

pTeX 実装の詳細

北川 弘典

2011/3/6 (日)

- 1 Packed Data
- 2 リスト
- 3 フォント
- 4 禁則処理・空白挿入制御
- 5 組方向
- 6 ベースライン補正
- 7 Category codes and tokens
- 8 DVI の拡張
- 9 数式
- 10 discretionary break
- 11 和文文字の list への追加
- 12 *adjust_hlist* procedure
- 13 A bug of ϵ -pTeX

pTeX 系列で不審な挙動があったら

- 1 L^AT_EX の範囲で再現する最小ソースを探す
- 2 plain T_EX の範囲で再現する最小ソースを探す
- 3 DVI や pdf で確認できる不具合なら, node 内容を見る.
`\showbox`, `\showlists` はこういうときに**非常に**有効.

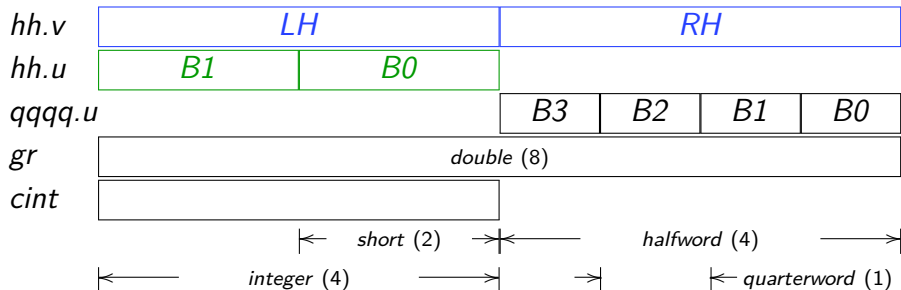
例 (radical_bug.tex, qa:55735)

```
\scrollmode\tracingonline=1
\showboxdepth\maxdimen\showboxbreadth\maxdimen
\setbox0=\hbox{\font\j=jis at20pt\textfont8\j
  $\jfam8\sqrt ひ\showlists$}
\end
```

- 4 ptex-base.ch などとにらめっこして, 原因を探す

texmfmem.h 内の実装: 通常

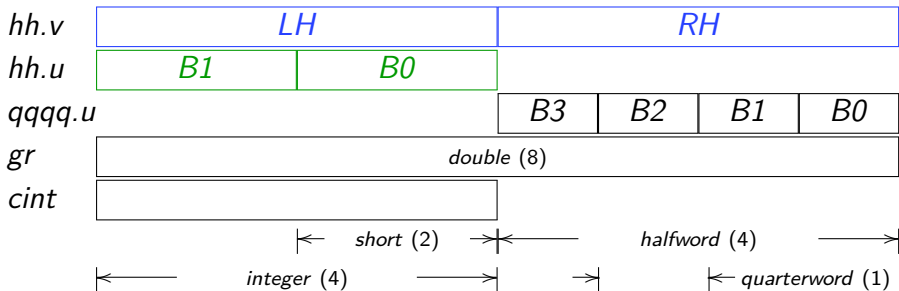
とりあえず Little Endian を仮定.



$$b? := u.B?, \quad lh := v.LH, \quad rh := v.RH$$

texmfmem.h 内の実装: 通常

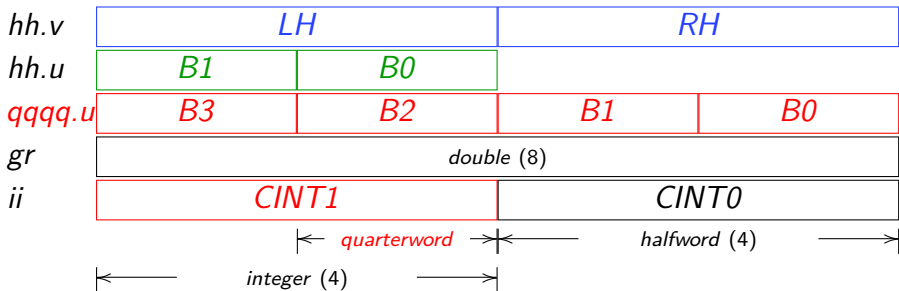
とりあえず Little Endian を仮定.



- `hh.lh`, `hh.rh` が 4 バイト: 配列 `mem` の要素数を 2^{16} 以上にしたい.
- `hh.b1`, `hh.b0` が 2 バイト: 256 個以上のフォントを使えるように.

texmfmem.h 内の実装: 変種

とりあえず Little Endian を仮定.



これは Ω , \aleph , ε -p $\text{T}_\text{E}\text{X}$, up $\text{T}_\text{E}\text{X}$ で使用される.

TEX の主な役目 = 垂直/水平リストを作ること

- 文字, box, glue, その他組版に関わるものは node で表現.

各 node の先頭 word は次のような構造:

<i>subtype</i>	<i>type</i> (種別)	<i>link</i> (次 node への参照)
----------------	------------------	---------------------------

但し, *char_node* (文字を表す) は:

<i>character</i>	<i>font</i>	<i>link</i>
------------------	-------------	-------------

- node p が *char_node* が否かは $is_char_node(p)$ で検出可能.
ちなみに, $is_char_node(null)$ は偽.
- node 達の linked list を用いてページの内容物を表す.

node の種類

- *hlist_node*, *vlist_node*, *dir_node*: box 用. 後述
- *rule_node* : rule (黒四角) 用. `\hrule`, `\vrule`
- *ins_node* : insertion
- *disp_node* : ベースライン補正用. 後述
- *mark_node* : mark
- *adjust_node* : `\vadjust`
- *ligature_node* : 合字用 (構成元の文字列も一緒に格納)
- *disc_node* : “discretionary” line break. 後述
- *whatsit_node* : `\write`, `\special`, `\pdfsavepos`, ...
- *math_node* : 数式境界, `\beginR ... \endR` 境界
- *glue_node* : glue (`\hskip`, `\vskip`)
- *kern_node*
- *penalty_node*
- *unset_node* : 表組 (`\halign`, `\valign`) 内部処理用

list_state_record (§16)

list_state_record: リストの状態を表現する構造体

- *mode_field*: リストのモード (垂直/水平/...)
- *head_field*, *tail_field*: リストのヘッダと最後.
link(head_field) がリストの実際の中身の先頭を表す.
- *dir_field*: 組方向
- *pnode_field*: 「リストの最後の *disp_node*」の一つ前の *node*
- *pdisp_field*: *pdisp_field* におけるベースライン補正值
- *last_jchr_field*: リスト中の最後の和文文字
- ...

→リストの最後 (*tail_field*) が *disp_node* だった場合, その直前の *node* には戻れる.

list_state_record (§16)

- 「現在のリスト」 (*cur_list*) に対しては省略形が用意:

$$\begin{aligned} \textit{head} &:= \textit{cur_list.head_field}, \\ \textit{tail} &:= \textit{cur_list.tail_field}, \\ \textit{direction} &:= \textit{cur_list.pdir_field}, \\ \textit{prev_node} &:= \textit{cur_list.pnode_field}, \\ \textit{prev_disp} &:= \textit{cur_list.pdisp_field}, \dots \end{aligned}$$

- *head_field* の値について:

- **TEX** の場合: 単なる長さ 1 の *node*. *is_char_node(head)* は真.
- **pTEX** の場合: *box* 用の *node* (数枚後参照) と同じ構造.
 - *is_char_node(head)* は偽.
 - *space_ptr(head)*: リスト内における `\kanjiskip` の値.
 - *xspace_ptr(head)*: リスト内における `\xkanjiskip` の値.

JFM (Japanese Font Metric)

和文フォントに対する TFM の対応物.

- 各和文文字は 0-255 のどれか一つの**文字クラス**に属する.
意図: 漢字は文字クラス 0 に属する.
- 文字の寸法や, 文字間に挿入される glue の情報は, 文字クラスごとに決まる.

例: jisg.tfm における文字クラス

0 下に挙がっている文字以外全部

1 開き括弧類 ‘ “ ([[{ < 《 「 『 【

2 閉じ括弧類 、 , ’ ”)]] } > 》 」 』]

3 中点類 ・ : ;

4 句点 。

5 ダッシュ — … …

TFM の模式図



DESIGNSIZE	QUAD	XHEIGHT	...
D	$1\text{ em}/D$	$1\text{ ex}/D$...
10 pt	0.962216	0.916443	...

char.	幅	高さ	深さ	italic
0	w_0	h_0	d_0	i_0
1	w_1	h_1	d_1	i_1
⋮	⋮	⋮	⋮	⋮

前	後	命令
0	0	/LIG 2
	2	kern $0.3D$
1	0	LIG/> 4
	1	kern $-0.2D$
	4	kern $0.25D$
⋮	⋮	⋮

JFM の模式図



DESIGNSIZE	QUAD	XHEIGHT	...	前	後	命令	
D	$1zw^1/D$	$1zh^1/D$...	0	0	$glue\ 0.0^{+0.25}_{-0.0} \cdot D$	
10 pt	0.962216	0.916443	...		2	kern $0.3D$	
<hr/>							
type	幅	高さ	深さ	italic	1	0	$glue\ 0.4^{+0.0}_{-0.5} \cdot D$
0	w_0	h_0	d_0	i_0		1	kern $-0.2D$
1	w_1	h_1	d_1	i_1		4	kern $0.25D$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

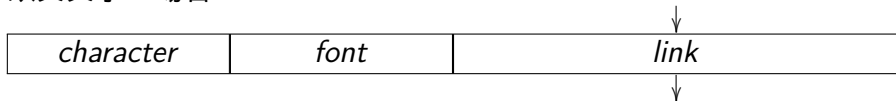
$\{char\} \longrightarrow \{type\}$: $\$ \mapsto 2$, $\text{あ} \mapsto 1$, $\text{く} \mapsto 3$, $\text{ぐ} \mapsto 2$, $\text{相} \mapsto 1, \dots$

TFM の場合とある程度同じ処理が使えるようになっている。

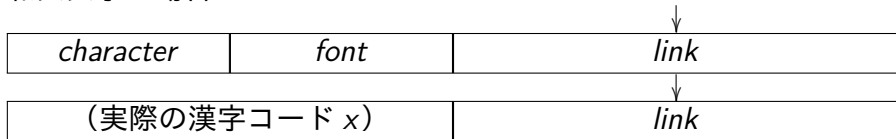
¹実際には、 $1zw$, $1zh$ は文字クラス 0 における寸法値が採用 (§55)。

character node

欧文文字の場合:



和文文字の場合:



character 部分は、実際には x の属する文字クラスの番号を格納。
「実際の漢字コード」のある node は殆ど読み飛ばせばよい。

「現在のフォント」 (§17)

- *cur_font*: 現在の欧文フォント
- *cur_jfont*: 現在の横組和文フォント
- *cur_tfont*: 現在の縦組和文フォント

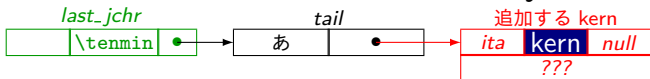
これらによって pT_EX は和文/欧文フォントを独立に変更可能。
また、これらの値は *ex*, *em*, *zw*, *zh* の長さを得るのにも用いられる。

問題

LuaT_EX では、当然「現在のフォント」として保持しているのは *cur_font* のみ。では、pT_EX のように「現在の和文フォント」も保持させるにはどうする？

イタリック補正 (§47)

- **T_EX** では, kern の種類を次の 3 種類としている:
 - *normal*: TFM や数式組版に由来.
 - *explicit*: `\kern` や `\/` (イタリック補正) によって挿入.
 - *acc_kern*: 非数式アクセントに由来.
- **pT_EX** では,
 - **イタリック補正由来の kern を *ita_kern* として, `\kern` 由来のものと区別.** この区別は *adjust_hlist* で活用.
 - 和文文字のイタリック補正の挿入時に *last_jchr* を利用:



「`\tenmin あ`」のイタリック補正量を知るには, *tail* の直前の *node* の場所がわからないといけない.

問題

どうやって LuaT_EX でイタリック補正と `\kern` 由来の kern を区別?

- **box** の内容をひたすらリストへ追加（後で）
 - 和文文字の *char_node* のリストへの追加
 - JFM 由来グルーの挿入
 - 禁則用 *penalty* の挿入
- ***adjust_hlist procedure***（後で）
 - 和欧文間空白の挿入
 - *\jcharwidowpenalty* の挿入
- （行分割処理）
- ***hpack function***: リストを *\hbox* の形にまとめる。
また、*box* の寸法測定や、*glue* 伸縮度 (*glue_set*) の決定も行う。
- **DVI 出力**

禁則テーブル

禁則処理は、文字の前後に`\penalty`を挿入することで行われる。

eqtb

⋮			
2	"A4A4		<i>kinsoku_base + r</i>
⋮			
1000			<i>kinsoku_penalty_base + r</i>
⋮			

- 左の状況のとき,
`\postbreakpenalty"A4A4=1000,`
 $kinsoku_type(r) = 2,$
 $kinsoku_code(r) = "A4A4,$
 $kinsoku_penalty(r) = 1000.$
- $0 \leq r < 256.$
- 赤字は `penalty` の種別。
0: (free), 1: pre, 2: post.

- 上のテーブルは `local` に代入が可能である。
- 同じ文字コードに対して pre, post 両方のエントリは持てない。
- 0番のエントリは「penalty 挿入せず」で固定。

禁則用 penalty 挿入場所

以下は「和文文字を現在の list に追加する」処理 (§55) で禁則用 penalty が挿入される場所:

- 和文文字の前後
- (和文文字連続の) 直前の欧文文字: `\postbreakpenalty` のみ
- (和文文字連続の) 直後の欧文文字: `\prebreakpenalty` のみ

他にも、欧文文字挿入処理 (§46) 中になぜか一箇所ある。

`qtrip.tex` で実行はされているので、無駄なコードではなさそうだが、どんなときに実行されるのかを私は忘れてしまった。

「現在の」 \kanjiskip

和文間空白 (\kanjiskip) の値は:

- 1 *kanji_skip*: 「ユーザーから」 \kanjiskip で見える値
- 2 *cur_kanji_skip*: グローバル変数. 内部処理で使われる.
 - box *b* の終了時
 - 数式終了時 } *adjust_hlist* 実行後 (*kanji_skip*)
 - 行分割処理開始時 (*space_ptr(head)*)
 - 表組 (\halign, \valign) 時に何回か (略)
 - アクセント上下移動量の計算時 (0.0 pt)
- 3 *space_ptr(head)*: 「現在のリストにおける」値
 - *adjust_hlist* 実行時 (*kanji_skip*)
- 4 *space_ptr(b)*: 「box_node *b* の中身における」値
 - *hpack* 実行時 (*cur_kanji_skip*)
 - *vpackage* 実行時 (0.0 pt)

和欧文間空白も同様 (*cur_xkanji_skip* 他).

和文間/和欧文間空白の違い

和欧文間空白 (`\xkanjiskip`) は, `adjust_hlist` procedure で `glue` の形で挿入される. この時の値は `xkanji_skip` が用いられる.

和文間空白 (`\kanjiskip`) は, `glue` の形でリスト中に挿入されることはない.

- `\hbox b` の DVI への出力時に, 和文文字連続時に `space_ptr(b)` だけ参照点を余計に移動させている.
- 行分割幅の計算時も, `cur_kanji_skip` の値が自動算入.
- `\hbox` の幅の計算時 (in `hpack`) も, `cur_kanji_skip` が自動算入.

`hpack` 実行時の処理のみ, 次で説明する.

\kanjiskip 算入

\kanjiskip 算入時は、どの文脈でも次の変数を用いている:

- *chain*: 直前（と現在）の node が和字を表す *char_node* か？

hpack 実行時 (§33):

各 *char_node* *p* に対し、*p* の幅を加算した後、次の処理を行い
\kanjiskip を算入:

```
k ← cur.kanji_skip;  
if font_dir[font(p)] ≠ dir_default then begin ← p の中身が和文文字  
   p ← link(p);                               ← を判定する常套句  
   if chain then begin x += width(k);  
     o ← stretch_order(k); total_stretch[o] += stretch(k);  
     o ← shrink_order(k); total_shrink[o] += shrink(k); end  
   else chain ← true;  
   end  
else chain ← false;
```

空白挿入の制御

pTeX では、`\kanjiskip` の自動挿入を制御するため、`\autospacing`, `\noautospacing` といった primitive がある。

これらの役割は単純:

- `\autospacing` : 内部パラメタ `auto_spacing` を 1 にする
- `\noautospacing`: 内部パラメタ `auto_spacing` を 0 にする
- 前に「`kanji_skip` の値を x に代入」と表現したところは、実は次のようになっている:

```
if auto_spacing > 0
  then  $x \leftarrow kanji\_skip$  else  $x \leftarrow (\text{zero glue});$ 
```

`\xkanjiskip` 側も同様.

\inhibitxspcode

和文文字ごとに、「直前/直後の欧文文字との間に\kxkanjiskipの自動挿入を行うか」も同じ方法で管理されている。

eqtb

⋮	
3	"A4A4
⋮	

inhibit_xsp_code_base + r

- 左の状況のとき,
`\inhibitxspcode"A4A4=3,`
$$inhibit_xsp_code(r) = 3.$$
- $0 \leq r < 256.$
- 0番のエントリは
「前後とも自動挿入許可 (3)」
で固定。

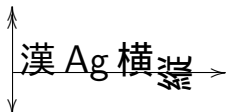
赤字の部分は空白自動挿入の指定。0-3のどれか。

- 上位 bit (2 の位) on: 直前の欧文文字との間に挿入を許可。
- 下位 bit (1 の位) on: 直後の欧文文字との間に挿入を許可。

dir_yoko (4)

dir_tate (3)

組版例



漢 Ag 横 漢

A diagram showing the horizontal layout of text. The characters '漢', 'Ag', '横', and '漢' are arranged horizontally. A vertical double-headed arrow is positioned to the left of the text, and a horizontal double-headed arrow is positioned below the text, indicating the direction of the layout.



漢
Ag
横
縦

A diagram showing the vertical layout of text. The characters '漢', 'Ag', '横', and '縦' are arranged vertically. A horizontal double-headed arrow is positioned above the text, and a vertical double-headed arrow is positioned to the right of the text, indicating the direction of the layout.

primitive
フォント
DVI *opcode*

`\yoko`
横
dirchg 0

`\tate`
縦
dirchg 1

モード判定

`\ifydir`
`\ifybox`

`\iftdir`
`\iftbox`

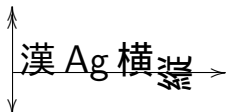
組方向

dir_yoko (4)

dir_tate (3)

dir_dtou (1)

組版例



primitive フォント	<code>\yoko</code> 横	<code>\tate</code> 縦	<code>\dtou</code> 横
DVI <i>opcode</i>	<code>dirchg 0</code>	<code>dirchg 1</code>	<code>dirchg 3</code>
モード判定	<code>\ifydir</code> <code>\ifybox</code>	<code>\iftdir</code> <code>\iftbox</code>	—

数式での「組方向」

- $direction = cur_list.dir_field$ のとり得る値:
 - $dir_yoko, dir_tate, dir_dtou$: 通常時 (地の文)
 - $-dir_yoko, -dir_tate, -dir_dtou$: 数式内の box
- $direction < 0$ のときは、地の文のモードと比較して,
 - 使用される和文フォントは横組のものとなる.
 - ベースライン補正も \lybaselineshift の値を用いる.

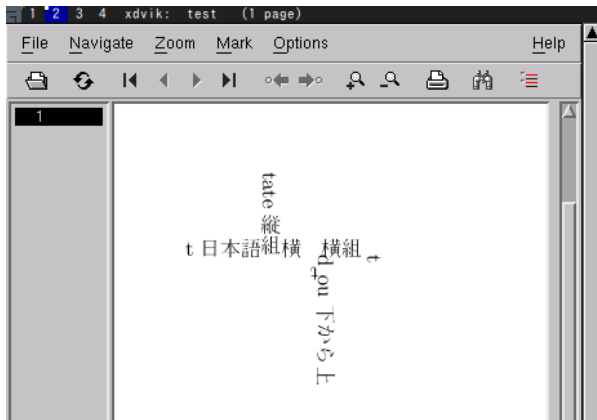
```
\vbox{\tate\Large 縦組  
$\hbox{aiu 漢字}$abc}
```

⇒

縦組
aiu 漢字
abc

\dtou には dviware のパッチが必要

```
\font\tenmin=jis\tenmin\font\tentmin=jis-v\tentmin
\tbaselineshift=3.8pt
t 日本語\hbox{\tate tate 縦組}横\hbox{\dtou dtou 下から上}横組 t\end
```



box 用の node

<i>box_dir</i>	<i>type</i>	<i>link</i>
<i>width</i>		
<i>depth</i>		
<i>height</i>		
<i>shift_amount</i>		
<i>glue_order</i>	<i>glue_sign</i>	<i>list_ptr</i> (中身)
<i>glue_set</i>		
<i>space_ptr</i>	<i>xspace_ptr</i>	

- *box_dir* ∈ {*dir_yoko*, *dir_tate*, *dir_dtou*}: box の組方向
- *shift_amount*: `\raise`, `\moveleft` 等による移動量
- *glue_order*, *glue_sign*, *glue_set*: 内部 glue の伸縮度

dir_node (§10)

box を表すための node として、次の 3 種類がある:

- *hlist_node* (0): 水平リストからなる box (`\hbox`)
- *vlist_node* (1): 垂直リストからなる box (`\vbox`)
- *dir_node* (2): 異なる組方向の box (`\hbox/\vbox` のコンテナ)

生成 (*new_dir_node*, §10)

b を box とし, $x' \neq x := \text{box_dir}(b)$ を組方向とする. このとき, b を「組方向 x' に合わせる」*dir_node* は, 次で作られる:

$$\begin{aligned} \text{box_dir}(b') &= x', & \text{list_ptr}(b') &= b, \\ (w', h', d') &= \begin{cases} (h + d, w/2, w/2) & \text{if } (x, x') = (\text{yoko}, \text{tate}), \\ (w, d, h) & \text{if } \{x, x'\} = \{\text{dtou}, \text{tate}\}, \\ (h + d, w, 0) & \text{otherwise.} \end{cases} \end{aligned}$$

```
\hbox{\yoko テスト \hbox{\tate あいうえお}テスト}
```

```
\hbox(52.68143+1.26434)x73.754, yoko direction
```

```
.\JY2/mc/m/n/10.95 テ
```

```
.\JY2/mc/m/n/10.95 ス
```

```
.\JY2/mc/m/n/10.95 ト
```

```
.\dirbox(52.68143+0.0)x10.53629, yoko direction
```

```
..\hbox(5.26814+5.26814)x52.68143, tate direction
```

```
...\JT2/mc/m/n/10.95 あ
```

(中略)

```
...\JT2/mc/m/n/10.95 お
```

```
.\JY2/mc/m/n/10.95 テ
```

(後略)

box レジスタの構造

box レジスタの中身は、実際には次の list である:

- 実際の box b (`\hbox` or `\vbox`): 組方向を x とする.
- 任意個の *dir_node*: 組方向が x でないときの「 b の寸法」保持. 寸法のみ保持していて, 中身への pointer *list_ptr* は null 値.

組方向 x' での寸法:

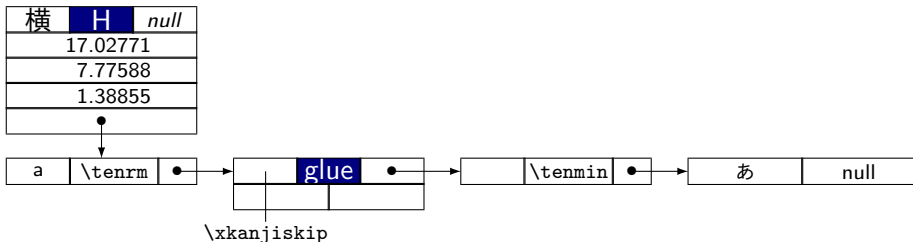
- $x = b$ のとき: 何もする必要はない. b の寸法そのまま.
- 組方向 x' の *dir_node* b' が b の後ろに存在:
 - b' が保持する寸法.
 - 組方向 x' で寸法設定が過去に行われたことを意味する.
- そうでないとき:
 - 組方向 x' に合わせるために, *dir_node* b' を b から作成.
 - b' の寸法が求める寸法値となる.

異なる組方向での寸法

例:

```
\setbox0\hbox{\yoko a あ}\the\wd0\hbox{\tate\the\ht0  
x\ht0=2pt}
```

\box0 の中身:

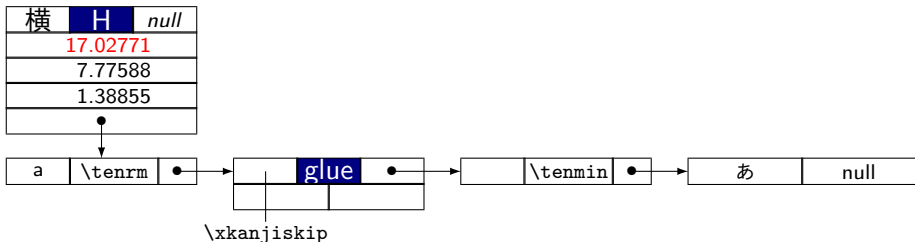


異なる組方向での寸法

例:

```
\setbox0\hbox{\yoko a あ}\the\wd0\hbox{\tate\the\ht0  
x\ht0=2pt}
```

\box0 の中身:

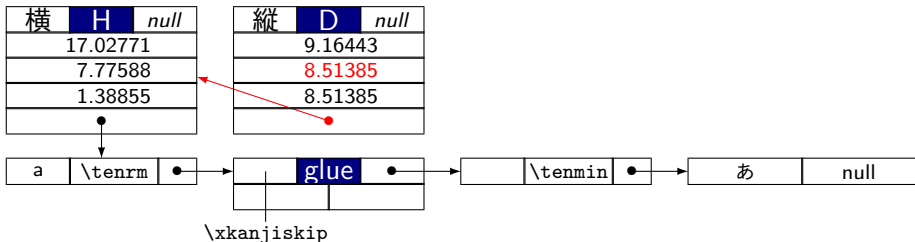


異なる組方向での寸法

例:

```
\setbox0\hbox{\yoko a あ}\the\wd0\hbox{\tate\the\ht0  
x\ht0=2pt}
```

\box0 の中身:

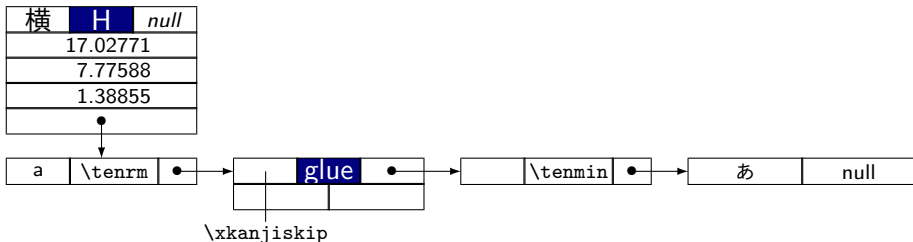


異なる組方向での寸法

例:

```
\setbox0\hbox{\yoko a あ}\the\wd0\hbox{\tate\the\ht0  
x\ht0=2pt}
```

\box0 の中身:

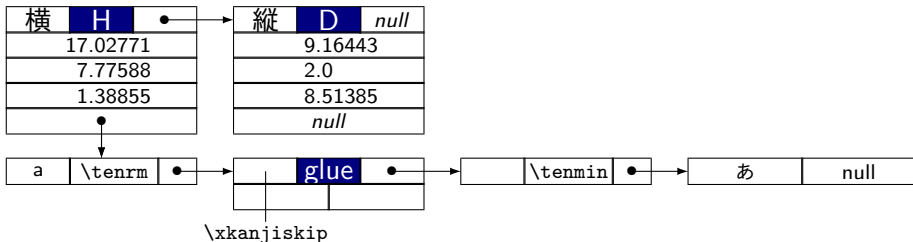


異なる組方向での寸法

例:

```
\setbox0\hbox{\yoko a あ}\the\wd0\hbox{\tate\the\ht0  
x\ht0=2pt}
```

\box0 の中身:



寸法取得/設定時のバグ

症状 (2011/2/24)

```
\setbox0=\hbox{\tate a}
```

```
\hbox{\yoko\ht0=30pt
```

```
\hbox{\dtou\dimen0=\ht0}
```

```
\the\ht0}
```

⇒ 5.00002 pt ← 30 pt では？

b から dir_node を作ると、以前に b より先 ($link(b)$ 以降) にあった node は切り離され、どこからも参照できなくなる。

```
\setbox0=\hbox{\tate a}
```

```
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```

$\box0$ の中身:

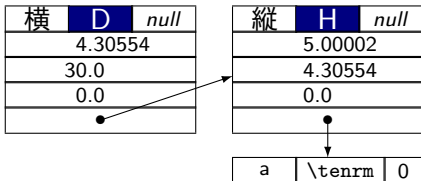
縦	H	null
5.00002		
4.30554		
0.0		
●		
↓		
a	\tenrm	0

原因

b から dir_node を作ると、以前に b より先 ($link(b)$ 以降) にあった node は切り離され、どこからも参照できなくなる。

```
\setbox0=\hbox{\tate a}  
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```

$\backslash box0$ の中身:



原因

b から dir_node を作ると、以前に b より先 ($link(b)$ 以降) にあった node は切り離され、どこからも参照できなくなる。

```
\setbox0=\hbox{\tate a}
```

```
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```

$\box0$ の中身:



原因

b から dir_node を作ると、以前に b より先 ($link(b)$ 以降) にあった node は切り離され、どこからも参照できなくなる。

```
\setbox0=\hbox{\tate a}  
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```

$\box0$ の中身:

du	D	<i>null</i>
5.00002		
0.0		
4.30554		
●		

縦	H	<i>null</i>
5.00002		
4.30554		
0.0		
●		

a	\tenrm	0
---	--------	---

orphaned		
横	D	<i>null</i>
4.30554		
30.0		
0.0		
<i>null</i>		

原因

b から dir_node を作ると、以前に b より先 ($link(b)$ 以降) にあった node は切り離され、どこからも参照できなくなる。

```
\setbox0=\hbox{\tate a}  
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```

$\box0$ の中身:

縦	H	null
5.00002		
4.30554		
0.0		
●		
↓		
a	\tenrm	0

orphaned		
横	D	null
4.30554		
30.0		
0.0		
null		

原因

b から dir_node を作ると、以前に b より先 ($link(b)$ 以降) にあった node は切り離され、どこからも参照できなくなる。

```
\setbox0=\hbox{\tate a}  
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```

$\backslash box0$ の中身:

横	D	null
4.30554		
5.00002		
0.0		
●		

縦	H	null
5.00002		
4.30554		
0.0		
●		

a	\tenrm	0
---	--------	---

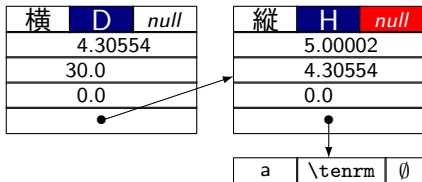
orphaned

横	D	null
4.30554		
30.0		
0.0		
null		

box の寸法取得/設定時の `new_dir_node` 実行時には、あらかじめ `link(b)` (への pointer) を待避させておき、`new_dir_node` の実行後に再設定すればよい。

```
\setbox0=\hbox{\tate a}
```

```
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```



box の寸法取得/設定時の `new_dir_node` 実行時には、あらかじめ `link(b)` (への pointer) を待避させておき、`new_dir_node` の実行後に再設定すればよい。

```
\setbox0=\hbox{\tate a}
\hbox{\yoko \ht0=30pt \hbox{\dtou \dimen0=\ht0} \the \ht0}
```



box の寸法取得/設定時の `new_dir_node` 実行時には、あらかじめ `link(b)` (への pointer) を待避させておき、`new_dir_node` の実行後に再設定すればよい。

```
\setbox0=\hbox{\tate a}
```

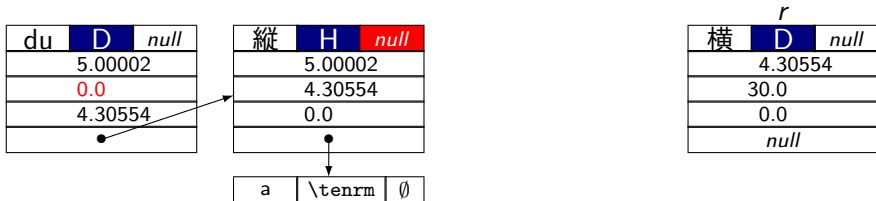
```
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```



box の寸法取得/設定時の `new_dir_node` 実行時には、あらかじめ `link(b)` (への pointer) を待避させておき、`new_dir_node` の実行後に再設定すればよい。

```
\setbox0=\hbox{\tate a}
```

```
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```



box の寸法取得/設定時の `new_dir_node` 実行時には、あらかじめ `link(b)` (への pointer) を待避させておき、`new_dir_node` の実行後に再設定すればよい。

```
\setbox0=\hbox{\tate a}
```

```
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```



box の寸法取得/設定時の `new_dir_node` 実行時には、あらかじめ `link(b)` (への pointer) を待避させておき、`new_dir_node` の実行後に再設定すればよい。

```
\setbox0=\hbox{\tate a}
```

```
\hbox{\yoko\ht0=30pt\hbox{\dtou\dimen0=\ht0}\the\ht0}
```



box の追加 (§47)

b : box とし, 現在の組方向を x' とする.

b は後ろに中身が空の *dir_node* 達を従えているかも.

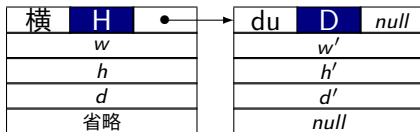
このとき, 「 b を現在の list に追加する」処理は:

- $box_dir(b) = x'$ のとき: 単に b を追加すればよい.
- *dir_node* b' で, 組方向が x' のものがあるとき:
 $list_ptr(b') \leftarrow b$ とし, b' を追加.
- そうでない時:
 b から「組方向 x' に合わせた」*dir_node* を作って, それを追加.
- 使われなかった *dir_node* 達は全部削除される.

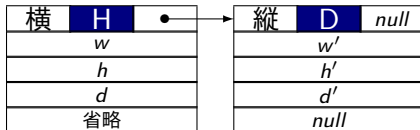
box の追加: 例

現在の組方向が *dir_tate* (縦組) と仮定する.

例 1:



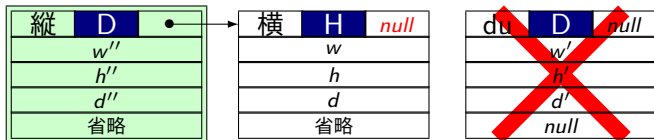
例 2:



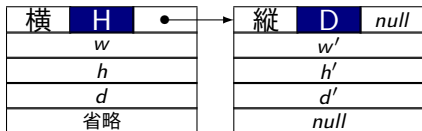
box の追加: 例

現在の組方向が *dir_tate* (縦組) と仮定する.

例 1:



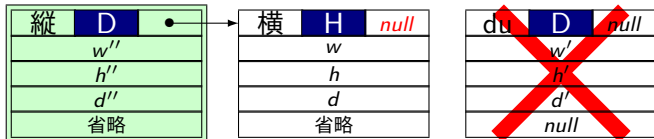
例 2:



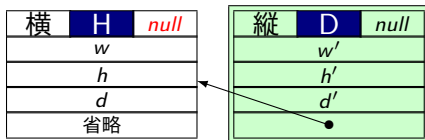
box の追加: 例

現在の組方向が *dir_tate* (縦組) と仮定する.

例 1:



例 2:



ベースライン補正

(特に縦組のとき) 和文と欧文・数式のベースラインをずらす必要が出てくる²:

漢
字

Ay
∫
dω

カ
ナ

補正なし

漢
字

Ay
∫
dω

カ
ナ

補正時
(4.1665pt)

→ pTeX では欧文・数式のベースラインをシフトさせることができる.

```
\hbox{\tate\normalsize 漢字 Ay  
$\int d\omega$カナ}
```

²横組の時は補正しなくても、あるいはしたとしても少量で済むだろう.

disp_node (§10)

未使用	<i>type</i>	<i>link</i> (次 node)
<i>disp_dimen</i>		

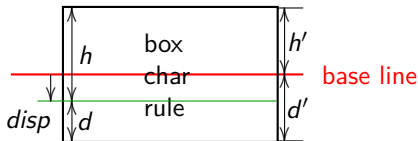
- 水平リストにのみ出現。これ以降の node は全て *disp_dimen* だけ下（行送り方向）だけシフトされる。
- （構築中の）リストはこの *disp_node* で終わることが多い。

問題

ベースライン補正の機能を LuaTeX に実装させるにはどうする？
和文フォントのベースラインを直すのは容易だが、欧文側を直すには？

hpack function (§33)

- 連続する和字間に *cur_kanji_skip* が挿入されているものとして box の幅を計算 (説明済み).
- box の高さ/深さの計算処理も、ベースライン補正のために変わっている:
 - ローカル変数 *disp*: 「現時点での」ベースライン補正值
 - *disp_node p* に会う度に, $disp \leftarrow disp_dimen(p)$.
 - box, char, rule について:



本来の高さは h , 深さは d としても, $disp$ 分の補正のため, ここでは高さ $h' = h - disp$, 深さ $d' = d + disp$ とみなされる.

disp_node の例

漢
字
A
y
J
d
w
カ
ナ

```
\hbox(5.06262+6.62024)x84.0413, tate direction
.\displace 0.0
.\JT1/mc/m/n/10.95 漢
.\JT1/mc/m/n/10.95 字
.\displace 3.80211
.\glue(\xkanjiskip) 2.7349 plus 1.64243 minus 0.65697
.\OT1/lmss/m/n/10.95 A
.\kern-0.30418
.\OT1/lmss/m/n/10.95 y
.\glue 3.65 plus 1.825 minus 1.21666
.\mathon
  (中略)
.\mathoff
.\displace 0.0
.\glue(\xkanjiskip) 2.73749 plus 1.64243 minus 0.65697
.\JT1/mc/m/n/10.95 カ
.\JT1/mc/m/n/10.95 ナ
```

disp_node の挿入

disp_node を挿入するケースは、次のときに行われる:

- 欧文文字の連続の挿入処理の最初と最後 (§46)
- (非数式の) アクセントの前後 (§47)
- インライン数式の前後 (§48)
- 和文文字の連続の挿入処理の最初 (§55)

最後の場合を除き,

- 「補正エリアの開始」 *disp_dimen = disp* の *disp_node* を作成
 - 「補正エリアの終了」: *disp_dimen = 0* の *disp_node* を作成
- と、ペアで挿入される³.

³最後の場合は「補正エリアの終了」のみ。和文文字の補正の必要なし。

disp_node の「前に」挿入

pTeX では、一部のものは、「リストの末尾に追加される」のではなく、「リストの末尾が *disp_node* なら、その直前に挿入」という形になっている:

kern, *insertion*, *mark*, *penalty*, 単語間空白

Peter Breitenlohner 氏による修正 (2011/1/19):

↑の事情により、`\lastpenalty`, `\lastkern`, (`\lastskip?`) がうまく動作しないことに対する修正. 具体的には,

```
if (type(tail) = disp_node) ∧ ¬is_char_node(prev_node)
  then q ← prev_node else q ← tail;
```

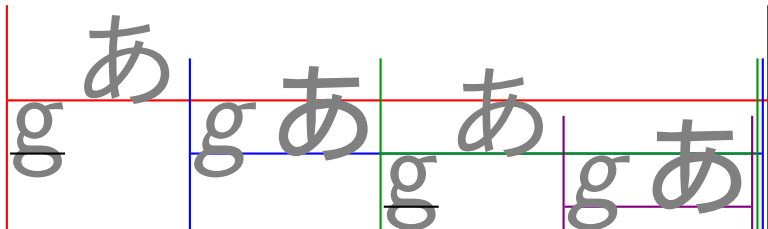
により、*tail* が *disp_node* なら、その直前の *node* を調べることにした. 実はまだ不十分. 詳細は後で.

数式内の\hbox

私にはこれが bug として認識すべきものかは分からないが、不自然な挙動なので述べておく。

```
{\ybaselineshift=5pt\font\g=jis at10pt\textfont15=\g
```

```
g あ $\jfam15g あ \hbox{g あ $g あ$} $ }
```



Category code (§15)

- | | | | |
|---|-------------------------|----|--|
| 0 | Escape character (\) | 10 | 空白 (<i>spacer</i> , <code>\ </code>) |
| 1 | グループ開始 ({) | 11 | 文字 (A..Z, a..z) |
| 2 | グループ終了 (}) | 12 | その他 |
| 3 | 数式モード境界 (\$) | 13 | アクティブ文字 |
| 4 | 表組時セル区切り (&) | 14 | コメント (%) |
| 5 | 改行文字 (<i>car_ret</i>) | 15 | 無効な文字 |
| 6 | マクロ引数 (#) | 16 | 漢字 |
| 7 | 上添字 (^) | 17 | かな文字等 |
| 8 | 下添字 (_) | 18 | その他 2 バイト文字 |
| 9 | 無視する文字 | | |

catcode 16–18 の違い

- (初期での) 文字範囲
 - 16 16 区以降 (JIS 第 1 水準/第 2 水準)
 - 17 3–6 区 (全角数字, 全角アルファベット, ギリシャ文字, かな)
 - 18 その他
- control sequence の名前 (`\csname` 時は除く)
 - catcode 11, 16, 17 以外の文字 1 文字
 - catcode 11, 16, 17 の文字の (できるだけ長い) 連続
- `\kcatcode` は pTeX にも存在するが……良い実装ではない⁴.

```
{\def\うう{A}\def\う{B}  
\kcatcode'あ=18\def\い{C}\うう\いう}
```

⇒ B う C う

⁴ptexenc の下では, 和文文字の catcode は「JIS コードに直したときの上位バイト」で定まる.

トークンの表現 (§20)

T_EX 内部において t が token T を表す自然数のとき,

- $0 \leq t < cs_token_flag$: T は文字トークンである.
 - $t = 2^8 m + c < "0F00$ のとき, T は文字コード c 番で, category code が m であるような文字トークンである.
 - そうでないとき, t はある和文文字の内部コード (Shift JIS or EUC) であり, T はそのコード番号の文字を表す文字トークン.
→和文文字の catcode は内部処理のたび文字コードから決定
- $t \geq cs_token_flag$: T はなんかの control sequence.

入力中の空白 (§24)

T_EX が入力を読み込むとき、動作モードは次の 4 つである：

- *mid_line*: 行の途中. 通常の動作モード.
- *mid_kanji*: 行の途中. 直前に読んだ文字が和文文字の場合.
- *skip_blanks*: 空白読み飛ばしモード
- *new_line*: 行の始め.

特筆すべき場合（空白が入ってこないことに注意）：

```
\xkanjiskip0pt あ{  
a あ}           ⇒ あaあf  
f
```

入力中の空白 (§24)

mode	読んだ文字の catcode	処理後の モード	処理内容
<i>mid_kanji</i>	0	<i>skip_blanks</i>	control sequence を scan
<i>mid_kanji</i>	1	<i>mid_kanji</i>	グループ開始
<i>mid_kanji</i>	2	<i>mid_kanji</i>	グループ終了
<i>mid_kanji</i>	5	<i>new_line</i>	<i>skip_mode</i> が偽 ⁵ なら空白発生
<i>mid_kanji</i>	10	<i>skip_blanks</i>	空白発生
<i>mid_kanji</i>	3, 4, 6–8, 11, 12	<i>mid_line</i>	
<i>mid_kanji</i>	13	<i>mid_line</i>	active 文字を scan
任意	16–18	<i>mid_kanji</i>	

⁵いつそうなる？

DVI フォーマット (§31)

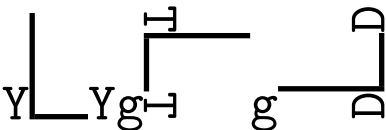
DVI は仮想機械への命令列と捉えられる:

- h, v, w, x, y, z, f, dir というレジスタが存在.
 - h : 現在の参照点の水平位置 (「右」へ増加)
 - v : 現在の参照点の垂直位置 (「下」へ増加)
 - w, x : 「右」方向の移動量
 - y, z : 「下」方向の移動量
 - dir : 現在の組方向
- ファイルサイズが小さくなるように工夫されている。
以下の命令では出力と同時に参照点が「右」にずれる:
 - $set_char_n, set\ m$ 命令 ($0 \leq n < 128, 1 \leq m \leq 4$): 文字を出力
 - set_rule 命令: rule (黒四角) を出力

pTeX では組方向を変える dir_chg (255) 命令が追加されており、これらは「右」「下」方向を変えるという意味を持つ。

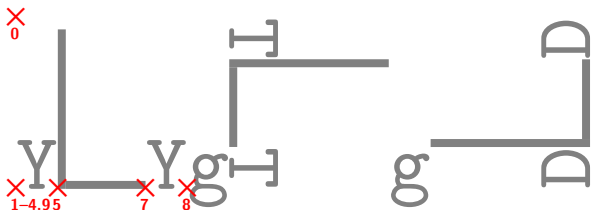
dir_chg 命令の例

```
\output={\shipout\box255}\tentt
\def\x{\vrule height 20pt width 1pt depth 0pt
  \vrule width 10pt height 1pt depth 0pt}\noindent
\hbox{\yoko Y\x Y}g\hbox{\tate T\x T}~%
\hbox{\dtou D\x D}g\end
```

⇒ 

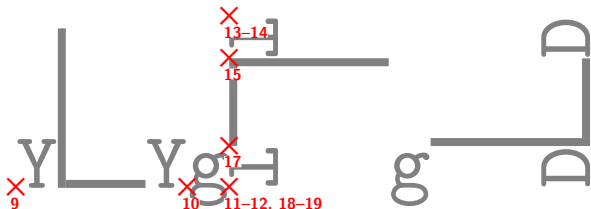
以下、この出力で得られる DVI ファイルの内容を確認していく。

dir_chg 命令の例



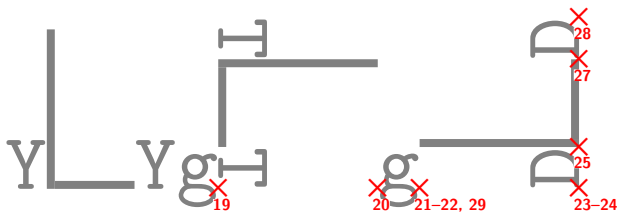
```
0                                     <- dir_yoko (initial)
1 down: 21.499908pt
2 push:
3   push:
4     fnt: cmtt10 at 10pt
5     set: 'Y'
6     setrule: 20pt 1pt
7     setrule: 1pt 10pt
8     set: 'Y'
9 pop:
```

dir_chg 命令の例



```
10 right: 21.499908pt
11 set: 'g'
12 push:
13   down: -21.499908pt
14   dir: 1 <- dir_tate
15   set: 'T'
16   setrule: 20pt 1pt
17   setrule: 1pt 10pt
18   set: 'T'
19 pop:
```

dir_chg 命令の例



```
20 w: 20pt
21 set: 'g'
22 push:
23   w0:
24   dir: 3 <- dir_dtou
25   set: 'D'
26   setrule: 20pt 1pt
27   setrule: 1pt 10pt
28   set: 'D'
29 pop:
```

DVI への出力 (§32)

box b を DVI に ship out する処理 (*ship_out*) では、以下の変数が用いられる（他にもある）：

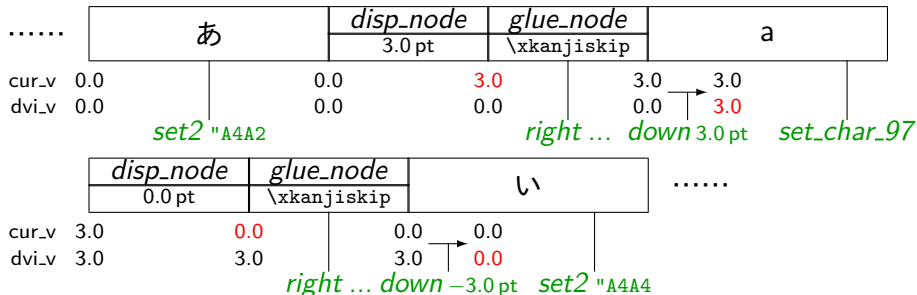
- *cur_h*, *cur_v*, *cur_dir*: T_EX における現在の参照点の位置
- *dvi_h*, *dvi_v*, *dvi_dir*: DVI ファイルにおける現在の位置
- *dvi_f*: DVI ファイルにおける現在のフォント
- *disp*: 欧文等のベースラインシフト量
- *base_line* (local): 現在のベースラインの縦位置

互換性のため、上の変数の *cur_dir*, *dvi_dir* の初期値は横組。また、 b は横組 (*dir_yoko*) と仮定して良い。そうでないときは、*ship_out* の最初に *dir_node* の中に b を放り込めば。

ベースライン補正 (\hbox 出力時)

cur_v を *base_line* 基準で *disp* だけずらすことによつて行う。

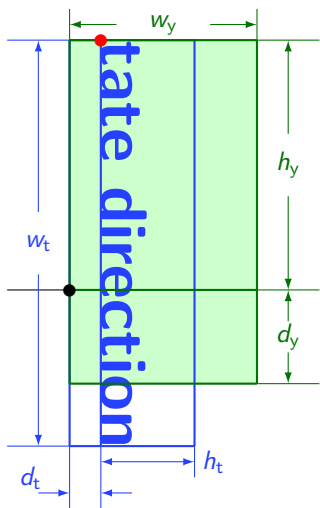
```
\hbox{...\ybaselineshift=3pt あ a い...}
```



上の図で、フォント回りは省略。また *base_line* = 0.0pt と仮定。

異方向 box の出力

例 1: 横組中に縦組の box 次のような状況を考える^a:



(direction: yoko)

```
\dirbox(h_y+d_y)x w_y, yoko direction  
.*box(h_t+d_t)x w_t, tate direction
```

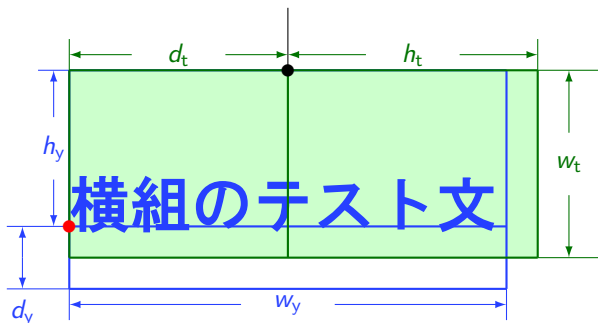
この *dir_node* d の DVI への出力処理は:

- 1 参照点 (cur_h, cur_v) は左図●の位置.
- 2 $cur_v -= h_y$, $cur_h += d_t$.
参照点は●の位置へ.
- 3 $list_ptr(d)$ の出力処理を呼び出す.

^a 「自然な状態」では, $h_y = w_t$, $d_y = 0$,
 $w_y = h_t + d_t$ だった.

異方向 box の出力

例 2: 縦組中に横組の box



(direction: tate)

```
\dirbox(h_t+d_t)x w_t, tate direction
```

```
.\*box(h_y+d_y)x w_y, yoko direction
```

参照点移動:

$$cur_v += d_t,$$

$$cur_h += h_y.$$

cur_v は右向きが正

「自然な状態」:

$$w_t = h_y + d_y,$$

$$h_t = w_y/2,$$

$$d_t = w_y/2.$$

数式のデータ構造 1 (§34)

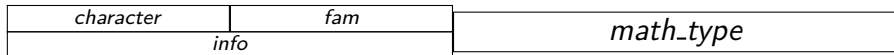
数式の構成要素は“noad”によって表され、noad のリストである“mlist”で数式全体を表す。以下は「基本形」の noad:

<i>subtype</i>	<i>type</i>	<i>link</i>
	<i>nucleus</i>	
	<i>supscr</i> (上添字)	
	<i>subscr</i> (下添字)	
	<i>kcode_noad</i>	
<i>math_kcode</i>		

- *type*: `\mathord`, `\mathop`, ..., `\mathinner`
- *subtype*: 1 (`\limits`) or 2 (`\nolimits`).
`\mathop` 時にのみ意味をもつ。
- 非基本形の noad: *radical_noad*, *accent_noad*, ...

数式のデータ構造 2 (§34)

nucleus, *supscr*, *subscr* は次のような構造:



■ *math_type*: 次のどれか

- *math_char*, (*math_text_char*):
fam 番フォントファミリの文字コード *character* の文字
- *math_jchar*, (*math_text_jchar*): *nucleus* にしか出現せず
fam 番の文字コード *math_kcode_nucleus* の文字⁶
- *sub_box*: *info* 番地が表す box
- *sub_mlist*: *info* 番地から始まる数式の list
- *math_text_char*, *math_text_jchar* は
内部処理でしか出てこない? らしい.

⁶この場所が前頁の *math_kcode* である!

オリジナルの pT_EX 3.1.11 で……

症状 (2010/3/23)

「 $\hat{\text{び}}$ 」をコンパイルすると、hat がつかず、次のメッセージが log ファイルに書かれる:

```
Missing character: There is no ^~d4 in font cmr10!
```

考察過程

- `\showlists` で観察すると、次のようになっていた。

```
\accent\fam0 ???
```

```
.\fam8 び
```

アクセント文字のコード (本来は"5E) が上書きされた。

- 「び」は EUC で"A4D4 である→これでやられたのでは？

原因

数式アクセント用の noad (*accent_noad*) の定義を見ると,

<i>type</i>	<i>subtype</i>	<i>link</i>
	<i>nucleus</i>	
	<i>supscr</i> (上添字)	
	<i>subscr</i> (下添字)	
	<i>accent_chr</i>	

……*accent_chr* が *kcode_noad* と同じ場所にある。上書きも当然。

解決策 (ptex-base.ch.0324.diff)

accent_noad の大きさを 1 つ増やし、*accent_chr* を後ろにずらせば良い。

同様の事態は *radical_noad* (根号等) でも起こる (2011/2/23)

<i>type</i>	<i>subtype</i>	<i>link</i>	
<i>nucleus</i>			
<i>supscr</i> (上添字)			
<i>subscr</i> (下添字)			
<i>left_delimiter</i>			
<i>large_char</i>	<i>large_fam</i>	<i>small_char</i>	<i>small_fam</i>

- pTeX では発現しない.

<i>math_kcode</i>				
	<i>l_c</i>	<i>l_f</i>	<i>s_c</i>	<i>s_f</i>

- ε-pTeX, upTeX では発現する.

<i>math_kcode</i>				
<i>large_char</i>	<i>large_fam</i>	<i>small_char</i>	<i>small_fam</i>	

<i>replace_count</i>	<i>type</i>	<i>link</i> (次 node)
<i>pre_break</i>		<i>post_break</i>

- *pre_break*: 「行末」部にくる内容の list への pointer
- *post_break*: 「行頭」部にくる内容の list への pointer
- この node で行分割が起こった場合、本 node に続く *replace_count* 個の node が無視され、代わりに前の 2 list が挿入される。
- これらの 3 つは、文字 (合字も含む), box, kern, rule, *disp_node* のみから構成されていないといけない。

\discretionary primitive

`\discretionary{⟨pre-break⟩}{⟨post-break⟩}{⟨no-break⟩}`

- この場所で改行しない場合: `⟨no-break⟩`
- 改行する場合: 行末には `⟨pre-break⟩`, 次行の頭には `⟨post-break⟩`

使用例: `\- = \discretionary{-}{}{} (hyphenation の指定).`

自由裁量の 不便な discretionary break

`\discretionary{}) {}{} ()` の後に挿入される JFM 由来の半角空き glue が ✖
⇒ ! Improper discretionary list.

`\discretionary{} \inhibitglue {}{} ()` の前に挿入される禁則用の penalty が ✖
⇒ ! This can't happen (disc4).

```
\ybaselineshift=4pt\discretionary{a}{あ}{c}\showlists
```

⇒

```
\hbox(0.0+0.0)x20.0  
\discretionary replacing 4  
. \displace 4.0  
. \tenrm a  
. \displace 0.0  
| \displace 0.0  
| \tenmin あ  
\displace 4.0  
\tenrm c  
\kern 2.0  
\displace 0.0  
\displace 0.0
```

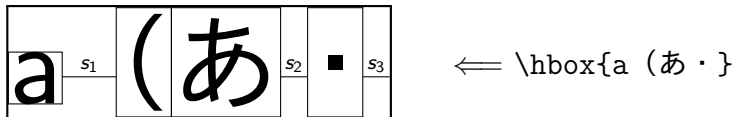
} 4 nodes

- 最後に $disp_dimen = 0\text{ pt}$ の $disp_node$ が 2 つ続いている.
 - 不用意にまとめてはいけない!
 - $\backslash lastkern$ の挙動が変
2.0 pt のはずだが, 0.0 pt が返る.
- 和文間/和欧文間空白はどうする.
glue を許容できるようにする?
無限に伸縮可能な glue ($\backslash hss$ 等) は?

$\backslash discretionary$ と日本語は相性悪.

JFM 由来 glue の挿入 (§55)

- 欧文文字の場合と同様に， $\text{T}_{\text{E}}\text{X}$ の中心部 *main_control* 内で行われる。
- 和文文字の連続の前後には，文字クラス 0 の文字が続いているものとして JFM 由来 glue が入る。



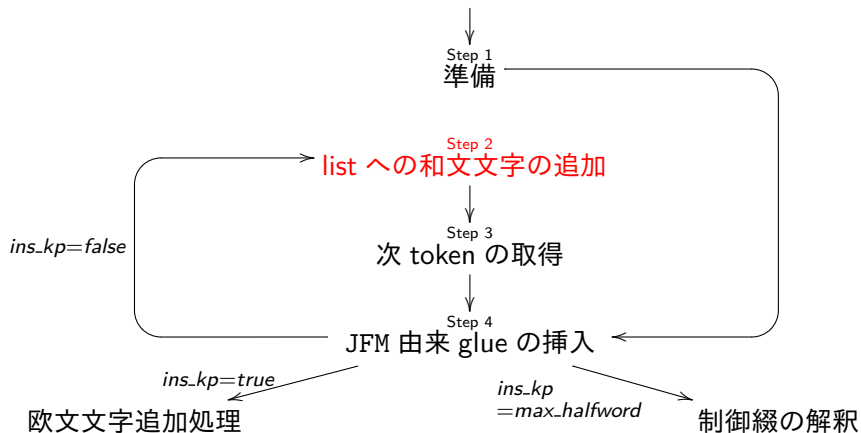
上で s_1 は文字クラス 0 と 1 の間に入る glue (半角幅)，
また s_3 は文字クラス 3 と 0 の間に入る glue (四分幅)。

- 行分割時には，当然，先頭/末尾の glue は取り除かれる (だから行の終わりの括弧や句読点は二分幅)。

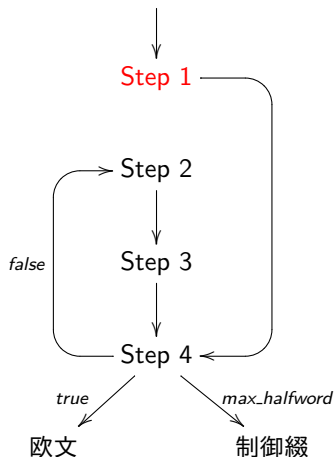
以下は実際のコード中の変数の意味とは若干異なる.

- *cur_chr*: 「これから list に追加する和文文字」の文字コード
- *cur_l*: 「これから list に追加する和文文字」の文字クラス
- *main_i*: 「直前に追加された和文文字」の文字クラス
- *ins_kp*: 次に挿入しようとする「文字」は何か?
false: 和文, *true*: 欧文, *max_halfword*: 制御綴 (文字でない)
- *main_f*: 現在の和文フォント

JFM 由来 glue の挿入: 概要

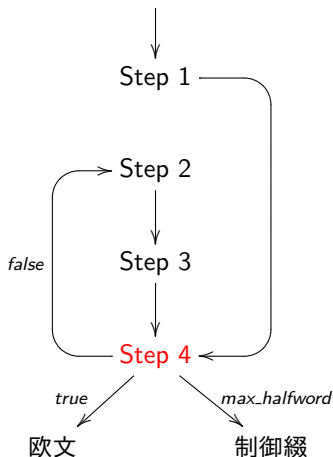


Step 1: 準備



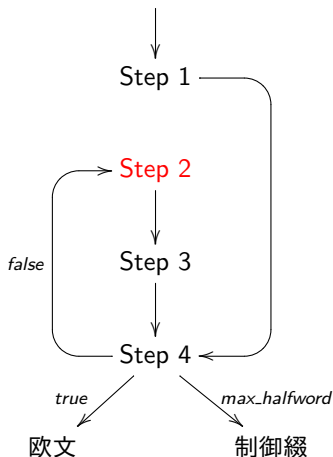
- 1 直前が欧文文字なら，その文字に応じた `\postbreakpenalty` を挿入。
- 2 *main-f* 設定 (*cur-jfont* or *cur-tfont*)
- 3 「ベースライン補正エリア」終了処理
- 4 *ins_kp* \leftarrow *false*. *cur-l* を *cur_chr* より計算
- 5 *main_i* \leftarrow 0.
最初の和文文字の前には
文字クラス 0 の文字があると見做す
- 6 Step 4 へ

Step 4: JFM 由来 glue の挿入

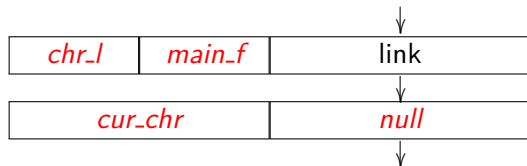


- 1 $ins_kp = true$ ならば,
それに応じた $\backslash prebreakpenalty$ を挿入.
- 2 $cur_q \leftarrow tail$.
- 3 $inhibit_glue_flag$ が真でないなら,
($main_i, cur_l$) に応じ JFM 由来 glue 追加.
- 4 $inhibit_glue_flag \leftarrow false$.
- 5 次の token に応じて行き先を設定:
 - $ins_kp = false \rightarrow$ Step 2 へ
 - $ins_kp = true$
 \rightarrow 欧文文字 cur_chr の挿入処理へ
 - $ins_kp = max_halfword$
 \rightarrow (今読み取った) 制御綴の処理へ

Step 2: リストへの追加



- 1 *char_node* 生成, *list* の末尾に追加.

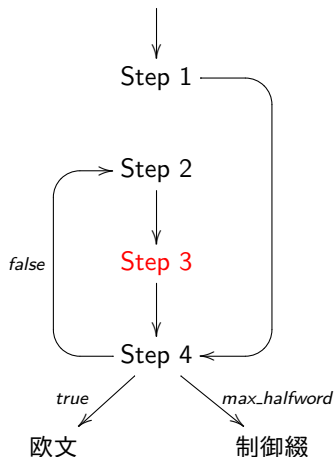


下の *node* が新しい *tail* となる.

- 2 禁則処理用 *penalty* の追加

- 後ろに *penalty* を追加する場合は, 単にリストの末尾に追加するのみ.
- 前に追加するときは, *cur_q* の直後に挿入.

Step 3: 次 token の取得



- 1 $main_i \leftarrow cur_l$.
- 2 次の token を取得, cur_chr 更新される.
- 3 次の token に応じて変数を設定:
 - 和文文字なら
 $ins_kp \leftarrow false$. cur_l を計算.
 - 欧文文字なら
 $ins_kp \leftarrow true$, $cur_l \leftarrow 0$.
 - 制御綴なら
 $ins_kp \leftarrow max_halfword$, $cur_l \leftarrow 0$.
 - $\backslash inhibitglue$ なら
 $inhibit_glue_flag \leftarrow true$, 2. へ戻る.

最後の和文文字の後には文字クラス 0 の文字があると見做す。

`\xkanjiskip` の挿入という，一番面倒くさい処理を行う⁷。

変数説明

- *p*: この node の前後に `\xkanjiskip` を挿入する。
- *q*: *p* の直前の node. 「*p* の直前に挿入」時に使う。
- *insert_skip*:
 - *insert_schar*: *p* が (直後に `\xkanjiskip` が挿入可能な) 欧文文字の直後であることを表す。
 - *insert_wchar*: *p* が和文文字の直後であることを表す。
 - *no_skip*: それ以外. *q* と *p* の間に `\xkanjiskip` は入れられない。

⁷`\jcharwidowpenalty` の挿入もここで行われるが，準備時間の都合で省略。

〈ASCII-KANJI 空白を挿入〉 cx : 漢字コード

- 和字 cx の前の `\xkanjiskip` 挿入が可能ならば、
node q と p の間に「`\xkanjiskip` 用 *glue_node*」を挿入.

〈KANJI-ASCII 空白を挿入〉 cx : 漢字コード, ax : 文字コード

- 和字 cx の後の `\xkanjiskip` 挿入が可能で、
かつ文字 ax の前の `\xkanjiskip` 挿入が可能ならば、
node q と p の間に「`\xkanjiskip` 用 *glue_node*」を挿入.

adjust_hlist procedure

引数: p (リストの header), pf ($\backslash jchrwidowpenalty$ 挿入?)

- 1 $space_ptr(p) \leftarrow kanji_skip$, $xspace_ptr(p) \leftarrow xkanji_skip$.
- 2 先頭 ($link(p)$) が JFM 由来グルーならば削除.
- 3 $insert_skip \leftarrow no_skip$, $p \leftarrow link(p)$, $q \leftarrow p$
- 4 **main loop**: $p \neq null$ の限り回る.
 - p が $char_node$ なら, $\langle p$ が $char_node$ の場合 \rangle .
 - そうでなければ, $\langle p$ が $char_node$ でない場合 \rangle .
 - $q \leftarrow p$, $p \leftarrow link(p)$.
- 5 q (リスト最後の node) が JFM 由来グルーならば削除.

〈 p が $char_node$ の場合〉

- 1 p の中身が欧文文字か和文文字かで分岐:
 - 和文文字のとき: $cx \leftarrow info(link(p))$ (漢字コード) とする.
 - 1 $insert_skip = after_schar$ のとき, 〈ASCII-KANJI 空白を挿入〉.
 - 2 $p \leftarrow link(p)$ (読み飛ばし), $insert_skip \leftarrow after_wchar$.
 - 欧文文字のとき: $ax \leftarrow character(p)$ (文字コード) とする.
 - 1 $insert_skip = after_wchar$ のとき, 〈KANJI-ASCII 空白を挿入〉.
 - 2 ax の後ろの $\backslash xkanjiskip$ 挿入が可能か見る.
 - 可能ならば, $insert_skip \leftarrow after_schar$.
 - 不可能ならば, $insert_skip \leftarrow no_skip$.
- 2 $q \leftarrow p$, $p \leftarrow link(p)$ (次の node へ)
- 3 p が $char_node$ なら, 最初に戻る.

〈 p が `char_node` でない場合〉

p の種別で分岐:

- `hlist_node` (`\hbox`): 〈`\hbox` の周囲に挿入〉
- `ligature_node`: 略
- `penalty_node`, `disp_node`: 〈`penalty` の周囲に挿入〉
- `kern_node`: 〈`kern` の周囲に挿入〉
- `math_node`:
 - 数式開始: `ax := "0"` とし, 〈KANJI-ASCII 空白を挿入〉.
 - 数式終了: `ax := "0"` とし, `insert_skip ← after_schar`.
 - どちらでもないとき⁸: `insert_skip ← no_skip`.
- `mark_node`, `adjust_node`, `ins_node`, `whatsit_node`: 何もしない
これらの `node` は現在の水平リストからは消える運命にある.
- その他: `insert_skip ← no_skip`.

⁸TeX--X₃Tにおける`\beginR`, `\endR`等がこの場合にあたる.

\hbox の場合の補助ルーチン

<KANJI-KANJI 空白を挿入 >

- q と p の間に `\kanjiskip` を追加.

<KANJI-KANJI 空白を追加 >

- p と $link(p)$ の間に `\kanjiskip` を追加.

check_box function 引数: p (*box_node*)

- 内部に文字があれば *true*, そうでなければ *false*.
- *first_char* に p 内の最初の *char_node* (maybe null) を, *last_char* に p 内の最後の *char_node* (maybe null) をセット.
- box 内部は, *shift_amount* = 0 の `\hbox` のみ再帰的に探索.
- 合字も, 「合字の構成元の文字列」に対して再帰的に探索.

〈\hbox の周囲に挿入〉

\raise, \lower による上下移動が行われていない場合のみ考慮。もしも上下移動が行われていれば, $insert_skip \leftarrow no_skip$ で終了。

次に, $check_box(list_ptr(p))$ により, 内部に文字がないか調べる。文字がなければ, $insert_skip \leftarrow no_skip$ 。

p の中身に文字があった時は:

- 1 $first_char \neq null$ なら, q と $first_char$ の間に $\backslash xkanjiskip$ を挿入。
 - $first_char$ が和文文字のとき: $cx \leftarrow info(link(first_char))$ 。
 - 1 $insert_skip = after_schar$ のとき, [〈ASCII-KANJI 空白を挿入〉](#)。
 - 2 $insert_skip = after_wchar$ のとき, [〈KANJI-KANJI 空白を挿入〉](#)。
 - 3 $insert_skip \leftarrow after_wchar$ 。
 - $first_char$ が欧文文字のとき: $ax \leftarrow character(first_char)$ 。
 - 1 $insert_skip = after_wchar$ のとき, [〈KANJI-ASCII 空白を挿入〉](#)。
 - 2 ax の後ろの $\backslash xkanjiskip$ 挿入が可能か見る。
 - 可能ならば, $insert_skip \leftarrow after_schar$ 。
 - 不可能ならば, $insert_skip \leftarrow no_skip$ 。

〈\hbox の周囲に挿入〉

- 2 $last_char \neq null$ なら, .
 - $last_char$ が和文文字のとき:
 - 1 $insert_skip \leftarrow after_wskip$.
 - 2 $link(p)$ (p の次の node) が和文文字ならば⁹,
〈KANJI-KANJI 空白を追加〉し, $p \leftarrow link(p)$.
 - $last_char$ が欧文文字のとき: $ax \leftarrow character(last_char)$.
 ax の後ろの $\backslash xkanjiskip$ 挿入が可能か見る.
可能ならば, $insert_skip \leftarrow after_schar$.
不可能ならば, $insert_skip \leftarrow no_skip$.
- 3 $last_char = null$ なら, $insert_skip \leftarrow no_skip$.

⁹事前に $is_char_node(link(p))$ を呼ぶので, $link(p) = null$ も弾かれる.

〈penalty の周囲に挿入〉

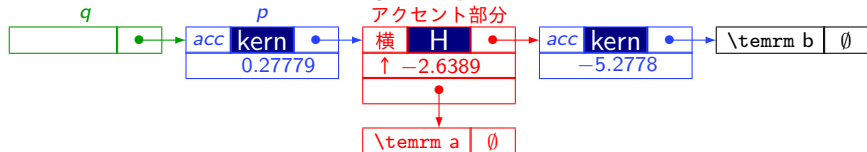
penalty も *disp_node* は、それぞれ行分割や DVI 出力時に使われるが、「実際に DVI にこの node の中身が出力される」わけではない。

- *link(p)* が *char_node* なら、
 $q \leftarrow p, p \leftarrow link(p)$ (次の node へ移動) した後で、
〈*p* が *char_node* の場合〉の 1. と同様の処理を行う。
- *link(p)* が *char_node* でなければ、何もしない。

〈kern の周囲に挿入〉

kern の種類 ($subtype(p)$) により場合わけ。

- *explicit* (明示的な `\kern`) のとき: $insert_skip \leftarrow no_skip$.
- *ita_kern* (イタリック補正) のとき: 何もしない。
- *normal* (TFM, 数式由来) のとき: 何もしない。
- *acc_kern* (アクセント由来) のとき:
アクセント部分 (下図青・赤) の node が存在しないものと考えて、 q の直後に `\xkanjiskip` を挿入する。



症状 (2011/2/26)

extended mode における ϵ -pTeX の TRIP test (2nd pass) において、赤字部分が出力されない:

```
.\hbox(0.0+0.0)x15.0, glue set 0.1875, shifted 5.0, display
```

背景: TeX--X₃T 拡張

- `\TeXXeTstate=1` で有効.
- `\beginR` と `\endR` の間の部分の水平リストをひっくり返す.
こうすることで、「右から左」の組版を実現させる.
- 但し、ディスプレイ数式はひっくり返さない.

例: `\beginR xy あ\beginL !\endL う bc\endR` \implies `cb う ! あ yx`

ϵ -TeX における *box_node* の先頭は

<i>subtype</i>	<i>type</i>	<i>link</i>
----------------	-------------	-------------

ここで, *subtype* の値の意味は:

- *min_quarterword*: 中身がひっくり返されるかもしれない.
- *reversed* := *min_quarterword* + 1:
既に中身がひっくり返された水平リスト.
- *dlist* := *min_quarterword* + 2:
ディスプレイ数式由来. 中身はひっくり返されない.

pTeX では, この領域は組方向の格納に用いるものであり, 衝突.

subtype の内,

- 上位 bit を T_EX--X₃T 拡張用 ($\in \{0, 1, 2\}$) に使用する.
- 下位 3 bit を pT_EX における組方向 ($\in \{0, 1, 3, 4\}$) に使用する.

	original	取得	設定
pT _E X	<i>box_dir(a)</i>	<i>box_dir(a)</i>	<i>set_box_dir(a)(dir_yoko)</i>
T _E X--X ₃ T	<i>subtype(a)</i>	<i>box_lrstat(a)</i>	<i>set_box_lrstat(a)(dlist)</i>