1 Definitions

1.1 Misc

Let $m = \left\lceil \left(\frac{N}{93}\right)^{\frac{1}{d}} \right\rceil$, base of the counter

$$\begin{aligned}
MSR &= \text{ most signifcant digit region} \\
\mathcal{C}_0 &= \text{starting value of counter} \\
d &= \lceil \log_m \mathcal{C}_0 \rceil = \left\lfloor \frac{k}{2} \right\rfloor, \text{ number of digits per row} \\
\mathcal{C}_f &= m^d, \text{ final value of the counter} \\
\mathcal{C}_\Delta &= \mathcal{C}_f - \mathcal{C}_0, \text{ number of rows/ times to count} \\
l &= \lceil \log m \rceil + 2, \text{ bits needed to encode each digit in binary, plus 2 for MSR and MSD}
\end{aligned}$$

1.2 Determining the starting value $\mathcal{C}_0$

...therefore, let $d = \lfloor \frac{k}{2} \rfloor$, $m = \left\lceil \left(\frac{N}{93}\right)^{\frac{1}{d}} \right\rceil$, $l = \lceil \log m \rceil + 2$, $\mathcal{C}_0 = m^d - \left\lfloor \frac{N-3l-76}{3l+90} \right\rfloor$, where d is the number of digits per row of the counter, m is the base of the counter, l is the number of bits needed to encode each digit in binary plus 2 for indicating whether a digit is in the MSR and is the MSD in that region, and $\mathcal{C}_0$ is the start of the counter in decimal.

In general, the height of a digit region is $3l+90$. There are two cases when the height is different, namely in the first and last digit regions, where the height is $3l+91$ and $3l+75$, respectively. Let h be the height of the construction before any filler/roof tiles are added. If we define $\mathcal{C}_\Delta$ as the number of `Counter` unit rows, then $h = (\mathcal{C}_\Delta - 1)(3l+90) + (3l+91) + (3l+75)$, simplifying to $\mathcal{C}_\Delta(3l+90) + 3l + 76$. So then the maximum height of the counter is $m^d(3l+90) + 3l + 76$. Since our goal is to end with a rectangle of height N, we need to pick a base such that the counter can increment so many times that when it stops, it is at least N.

Lemma 1. *$N \leq m^d(3l+90) + 3l + 76$.*

Proof.

$$\begin{aligned}
N = 93\left(\frac{N}{93}\right) = 93\left(\left(\frac{N}{93}\right)^{\frac{1}{d}}\right)^d &\leq 93\left\lceil \left(\frac{N}{93}\right)^{\frac{1}{d}} \right\rceil^d \\
&= 93m^d \leq 3lm^d + 90m^d \leq 3lm^d + 90m^d + 3l + 76 \\
&= m^d(3l+90) + 3l + 76
\end{aligned}$$

$\square$

1.3 Filling in the gaps

...this means that the number of `Counter` unit rows $\mathcal{C}_\Delta$ is $m^d - \mathcal{C}_0$, where we have defined $\mathcal{C}_0$ as the starting value of the counter. To choose the best starting value, we find the value for $\mathcal{C}_\Delta$ that gets h as close to N without exceeding N. It follows from the equation $h = \mathcal{C}_\Delta(3l+90) + 3l + 76$, that $\mathcal{C}_\Delta = \left\lfloor \frac{N-3l-76}{3l+90} \right\rfloor$. Thus, $\mathcal{C}_0 = m^d - \left\lfloor \frac{N-3l-76}{3l+90} \right\rfloor$. As a result of each digit requiring a width of 2 tiles, if k is odd, one additional tile column must be added. The number of filler tiles needed for the width is $k \mod 2$, and the number of filler tiles for the height is $N - 3l - 76 \mod 3l + 90$.

2 Gadgets

- h, here
- t, top
- b, bottom
- p, page of float

and LaTeX will try to honour the placement with respect to the actual place, the top or bottom of the page, or a separate page of floats coming immediately after the present insertion point. For example, when using ht LaTeX will try to put the figure at the insertion point, then on the top of the next page if it happens to violate its typesetting rules. You may also force LaTeX to "insist" on these specifications by adding an exclamation mark (!) before the placement parameters, e.g. !htb

`Gadget`($\langle$Input$\rangle$, $\langle$Output$\rangle$)

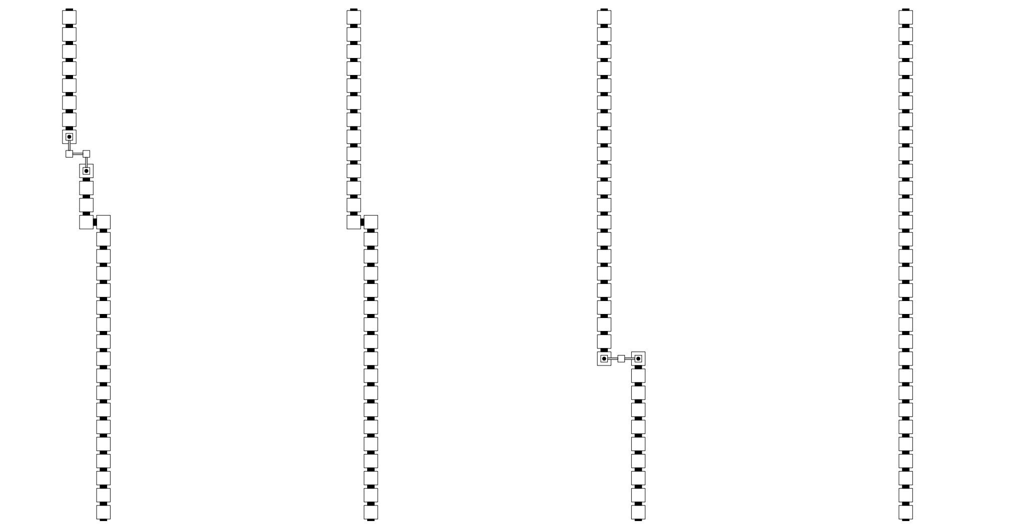
2.1 Counter Unit

2.1.1 Digit readers

2.1.2 Warping

For each $d \in \{0,1\}^l$, $i = 1, 2, 3$

– `Pre_Warp`($\langle d, i, \texttt{carry}\rangle$) $\approx O(1)$



(a) General (b) Case 1 – Digit 1 (c) Case 2 – Digit 1 (d) Case 2 – Digit 2

Figure 1: `Pre_Warp` gadgets

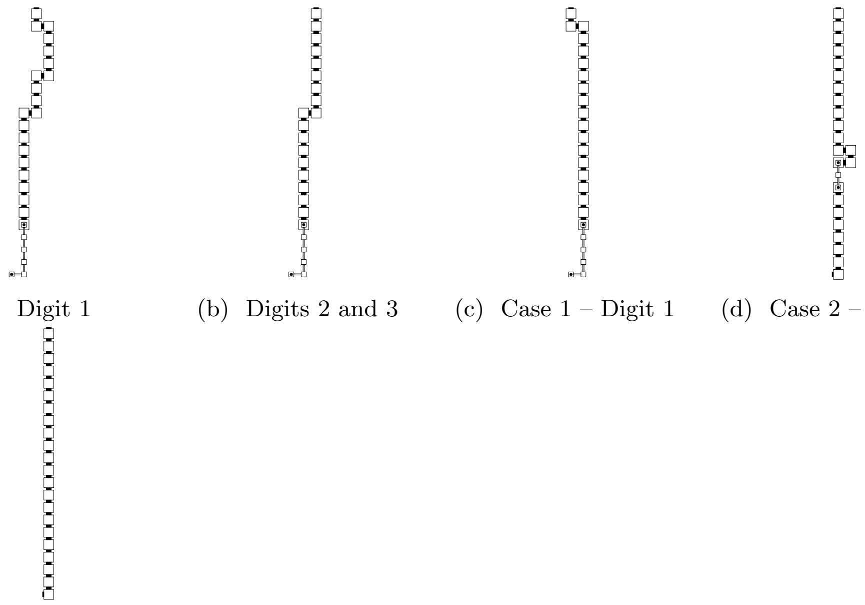
– `First_Warp`($\langle d, i, \texttt{carry}\rangle$) $\approx O(1)$

– `Warp_Bridge`($\langle d, i, \texttt{carry}\rangle$) $\approx O(1)$

– `Second_Warp`($\langle d, i, \texttt{carry}\rangle$) $\approx O(1)$

– `Post_Warp`($\langle d, i, \texttt{carry}\rangle$) $\approx O(1)$

Tiles created for the warp units $\approx O(\log m)$



(a) Digit 1 (b) Digits 2 and 3 (c) Case 1 – Digit 1 (d) Case 2 – Digit 1

(e) Case 2 – Digit 2

Figure 2: `Post_Warp` gadgets

2.1.3 Digit writers

2.1.4 Digit tops

The `Digit_Top` gadgets of have specific geometry, such that they allow `First_Warp` and `Second_Warp` units to "wake up" and end their warp journey. A `Digit_Top` is placed on the north end of a digit. These hold a increment/copy signal and the regional index of the next digit to read. $\approx O(\log m)$

For each `carry` $\in \{$`increment`, `copy`$\}$

- General digit tops common to all assemblies
  - create `Digit_Top`($\langle$`carry`$, 1, l\rangle$)

$$\text{input}\rightarrow\text{south} = \texttt{Digit\_Top}(\texttt{carry}, 1)$$
$$\text{output}\rightarrow\text{south} = \texttt{Return\_From\_Digit1\_Read\_Digit2}(\texttt{carry})$$

  - create `Digit_Top`($\langle$`carry`$, 2, l\rangle$)

$$\text{input}\rightarrow\text{south} = \texttt{Digit\_Top}(\texttt{carry}, 2)$$
$$\text{output}\rightarrow\text{south} = \texttt{Return\_From\_Digit2\_Read\_Digit3}(\texttt{carry})$$

  - create $\texttt{Digit\_Top}(\langle \texttt{carry}, 3, l\rangle)$

$$\begin{aligned} \text{input}{\rightarrow}\text{south} &= \texttt{Digit\_Top}(\texttt{carry}, 3) \\ \text{output}{\rightarrow}\text{south} &= \texttt{Return\_From\_Digit3\_Read\_Digit1}(\texttt{carry}) \end{aligned}$$

- MSR-specific digit tops.
  - if i is 1 and MSR contains 2 digits: create $\texttt{Digit\_Top\_Digit1\_Case2}(\langle \texttt{carry}, l\rangle)$

$$\begin{aligned} \text{input}{\rightarrow}\text{south} &= \texttt{Digit\_Top\_Digit1\_Case2}(\texttt{carry}) \\ \text{output}{\rightarrow}\text{south} &= \texttt{Return\_From\_Digit1\_Read\_Digit2\_Case2}(\texttt{carry}) \end{aligned}$$

  - if i is 2 and MSR contains 2 digits: create $\texttt{Digit\_Top\_Digit2\_Case2}(\langle \texttt{carry}, l\rangle)$

$$\begin{aligned} \text{input}{\rightarrow}\text{south} &= \texttt{Digit\_Top\_Digit2\_Case2}(\texttt{carry}) \\ \text{output}{\rightarrow}\text{south} &= \texttt{Return\_From\_Digit2\_Read\_Next\_Row}(\texttt{carry}) \end{aligned}$$

  - if i is 3 and MSR contains 3 digits: create $\texttt{Digit\_Top\_Digit3\_Case3}(\langle \texttt{carry}, l\rangle)$

$$\begin{aligned} \text{input}{\rightarrow}\text{south} &= \texttt{Digit\_Top\_Digit3\_Case3}(\texttt{carry}) \\ \text{output}{\rightarrow}\text{south} &= \texttt{Return\_From\_Digit3\_Read\_Next\_Row}(\texttt{carry}) \end{aligned}$$

2.1.5 Return paths between the MSD and LSD in different rows

The gadgets of this class hold a increment/copy signal. The height of these gadgets is dependent on l and the width is dependent of k. These gadgets are used to begin reading the first digit in the following row, once the MSD has been read in the current row.

For each $\texttt{carry} \in \{\texttt{increment}, \texttt{copy}\}$

1. `Return_From_Digit1_Read_Next_Row`

$$\begin{aligned} \text{input}{\rightarrow}\text{south} &= \texttt{Return\_From\_Digit1\_Read\_Next\_Row}(\texttt{carry}) \\ \text{output}{\rightarrow}\text{north} &= \texttt{Digit\_Reader}(\texttt{carry}, 1) \end{aligned}$$

2. `Return_From_Digit2_Read_Next_Row`

   input→north = `Return_From_Digit2_Read_Next_Row`(`carry`)
   output→north = `Digit_Reader`(`carry`, 1)

3. `Return_From_Digit3_Read_Next_Row`

   input→north = `Return_From_Digit3_Read_Next_Row`(`carry`)
   output→north = `Digit_Reader`(`carry`, 1)

2.1.6 Return paths between digits in the same row

The gadgets of this class hold a increment/copy signal and the regional index of the next digit to read. The height of these gadgets is dependent on l. These gadgets are used so that upon writing a digit, the counter is able to move back down to the next digit in the current row, and continue reading.

- `Return_From_Digit1_Read_Digit2`

  input→north = `Return_From_Digit1_Read_Digit2`(`carry`)
  output→north = `Digit_Reader`(`carry`, 2)

- `Return_From_Digit1_Read_Digit2_Case2`

  input→north = `Return_From_Digit1_Read_Digit2_Case2`(`carry`)
  output→north = `Digit_Reader`(`carry`, 2)

- `Return_From_Digit2_Read_Digit3`

  input→north = `Return_From_Digit2_Read_Digit3`(`carry`)
  output→north = `Digit_Reader`(`carry`, 3)

- `Return_From_Digit3_Read_Digit1`

  input→north = `Return_From_Digit3_Read_Digit1`(`carry`)
  output→north = `Digit_Reader`(`carry`, 1)

2.2 Seed Unit