Ich definiere folgende eigene Makros:

1: Die Ungleich–Null–Abfrage: IF $x_i \neq 0$ THEN P END

```
x_t := x_empty + 0;
LOOP x_i DO x_t := x_empty + 1; END;
LOOP x_t DO P END
```

2: Die Subtraktion: $x_i := x_j - x_k$

```
x_i := x_j + 0;
LOOP x_k DO x_i := x_i − 1 END
```

3: Die Variable x_{empty} ist eine ungebrauchte Variable, die somit den Wert 0 hat und für die Operation $a := c$ herhalten muss (mit $a := x_{empty} + c$).

Aufgabe 1.

a) Exponentialfunktion $x_i := x_j^{x_k}$

```
x_i := x_empty + 1;
LOOP x_k DO x_i := x_i · x_j
```

b/c) Dieses LOOP–Programm errechnet sowohl den ganzzahligen Anteil ($x_d := x_j$ DIV x_k), als auch den Rest der Division ($x_m := x_j$ MOD x_k). Sollte $x_k = 0$ sein, macht das Programm natürlich keinen Sinn, da ja nicht durch Null geteilt werden darf.

```
// Variablen mit 0 belegen
x_d := x_empty + 0;
x_m := x_empty + 0;
// WHILE–Schleife simulieren (x_j ist auf jeden Fall groß genug)
LOOP x_j DO
.   IF x_j ≠ 0 THEN
.   .   x_d := x_d + 1;
.   .   x_m := x_j + 0;
.   .   x_j := x_j − x_k;
.   END;
END;
x_j := x_k − x_m; // x_j hat ausgedient, benütze sie jetzt als Hilfsvariable
IF x_j = 0 THEN x_m := x_empty + 0; END;
IF x_j ≠ 0 THEN x_d := x_d − 1; END
// Ergebnisse in x_d und x_m
```

Da ein LOOP–Programm so definiert ist, dass das Ergebnis der Berechnung am Schluss in x_0 gespeichert ist, muss man in diesem Programm – welches ja zwei verschiede Ergebnisse produziert – entweder den Index d oder den Index m als 0 wählen, je nachdem, ob die DIV oder die MOD Funktion errechnet werden soll.

Aufgabe 2.

a) Zwei natürliche Zahlen können in einer kodiert werden, indem man die Darstellung zu einer definierten Basis wählt, und dann eine Zahl konstruiert, deren Stellen abwechselnd mit denen der zu kodierenden Zahlen aufgefüllt werden. Hierzu ein paar Beispiele (ich wähle die Basis 10):

$$\tau(1111, 3333) = 31313131 \quad (1)$$
$$\tau(444, 2) = 40424 \quad (2)$$
$$\tau(5, 55) = 5055 \quad (3)$$

Oder allgemein

$$\tau(a_n \dots a_1 a_0, b_k \dots b_1 b_0) = \dots a_1 b_1 a_0 b_0 \quad (4)$$

Die Umkehrfunktionen π_1 und π_2 gehen analog vor, indem sie entweder die geraden oder die ungeraden Stellen einer Zahl zu einer neuen konstruieren.

b) LOOP–Programm für $\tau(x_1, x_2)$. Das Programm rechnet zur Basis 2. Ich habe das Programm zwei mal abgedruckt. Einmal korrekt nur mit primitiven Makros, und einmal etwas großzügiger, dafür besser zu lesen :-)
$x_0 := \tau(x_1, x_2)$

$x_e := x_1 + x_2$; // x_e als ultimative Hilfsvariable → wird alle paar Zeilen überschrieben
$x_z := x_{empty} + 2$; // Die 2 in x_z speichern
LOOP x_e DO
. $x_e := x_c + 1$; // Exponent
. $x_e := x_z^{x_e}$; // Zwei „hoch Schleifendurchlauf“
. $x_5 := (x_1 \text{ MOD } x_e)$; // Ungleich 0, wenn Bit Nr. x_c gesetzt ist
. $x_6 := (x_2 \text{ MOD } x_e)$; // dito
. $x_e := x_c \cdot 2$;
. $x_e := x_z^{x_e}$;
. IF $x_5 \neq 0$ THEN $x_0 := x_0 + x_e$; END;
. $x_e := x_e \cdot 2$; // Ein Bit nach links...
. IF $x_6 \neq 0$ THEN $x_0 := x_0 + x_e$; END;
. $x_1 := x_1 - x_5$;
. $x_2 := x_2 - x_6$;
. $x_c := x_c + 1$; // Counter um 1 erhöhen
END

$x_0 := \tau(x_1, x_2)$; Mit „großzügiger“ Syntax...

```
LOOP (x_1 + x_2) DO
.   x_5 := (x_1 MOD 2^(x_c+1)); // Ungleich 0, wenn Bit Nr. x_c gesetzt ist
.   x_6 := (x_2 MOD 2^(x_c+1)); // dito
.   IF x_5 ≠ 0 THEN x_0 := x_0 + 2^(x_c·2); END;
.   IF x_6 ≠ 0 THEN x_0 := x_0 + 2^(x_c·2+1); END;
.   x_1 := x_1 − x_5;
.   x_2 := x_2 − x_6;
.   x_c := x_c + 1; // Counter um 1 erhöhen
END
```

LOOP–Programm für $x_0 := \pi_1(x_1)$. Um das entsprechende Programm für π_2 zu bekommen, muss man einfach x_0 und x_x vertauschen.
Einmal „besser leserlich“...

```
LOOP x_1 DO
.   x_5 := (x_1 MOD 2^(x_c·2+1));
.   x_1 := x_1 − x_5;
.   x_6 := (x_1 MOD 2^(x_c·2+2));
.   x_1 := x_1 − x_6;
.   IF x_5 ≠ 0 THEN x_0 := x_0 + 2^x_c; END;
.   IF x_6 ≠ 0 THEN x_x := x_x + 2^x_c; END;
.   x_c := x_c + 1; // Counter um 1 erhöhen
END
// Ergebnisse in x_0 für π_1 und x_x für π_2
```

... und einmal „korrekt“

```
x_z := x_empty + 2;
LOOP x_1 DO
.   x_e := x_c · 2;
.   x_e := x_e + 1;
.   x_e := x_z^x_e;
.   x_5 := (x_1 MOD x_e);
.   x_1 := x_1 − x_5;
.   x_e := x_e · 2; // Ein Bit nach Links...
.   x_6 := (x_1 MOD x_e);
.   x_1 := x_1 − x_6;
.   x_e := x_z^x_c;
.   IF x_5 ≠ 0 THEN x_0 := x_0 + x_e; END;
.   IF x_6 ≠ 0 THEN x_x := x_x + x_e; END;
.   x_c := x_c + 1; // Counter um 1 erhöhen
END
// Ergebnisse in x_0 für π_1 und x_x für π_2
```