Representación numérica en el computador

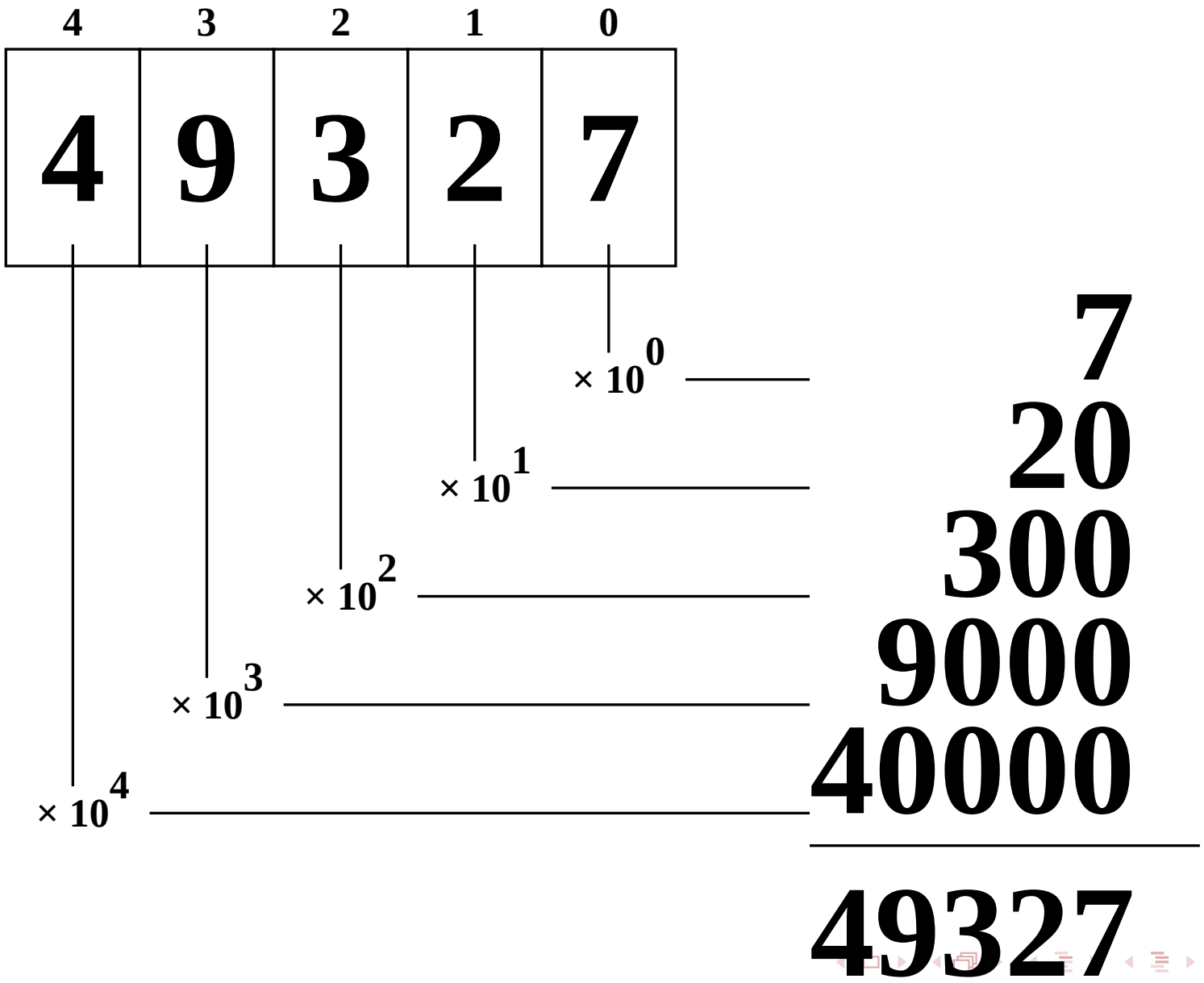
Fundamentos de Computación

Grado en Ingeniería en Tecnologías Industriales

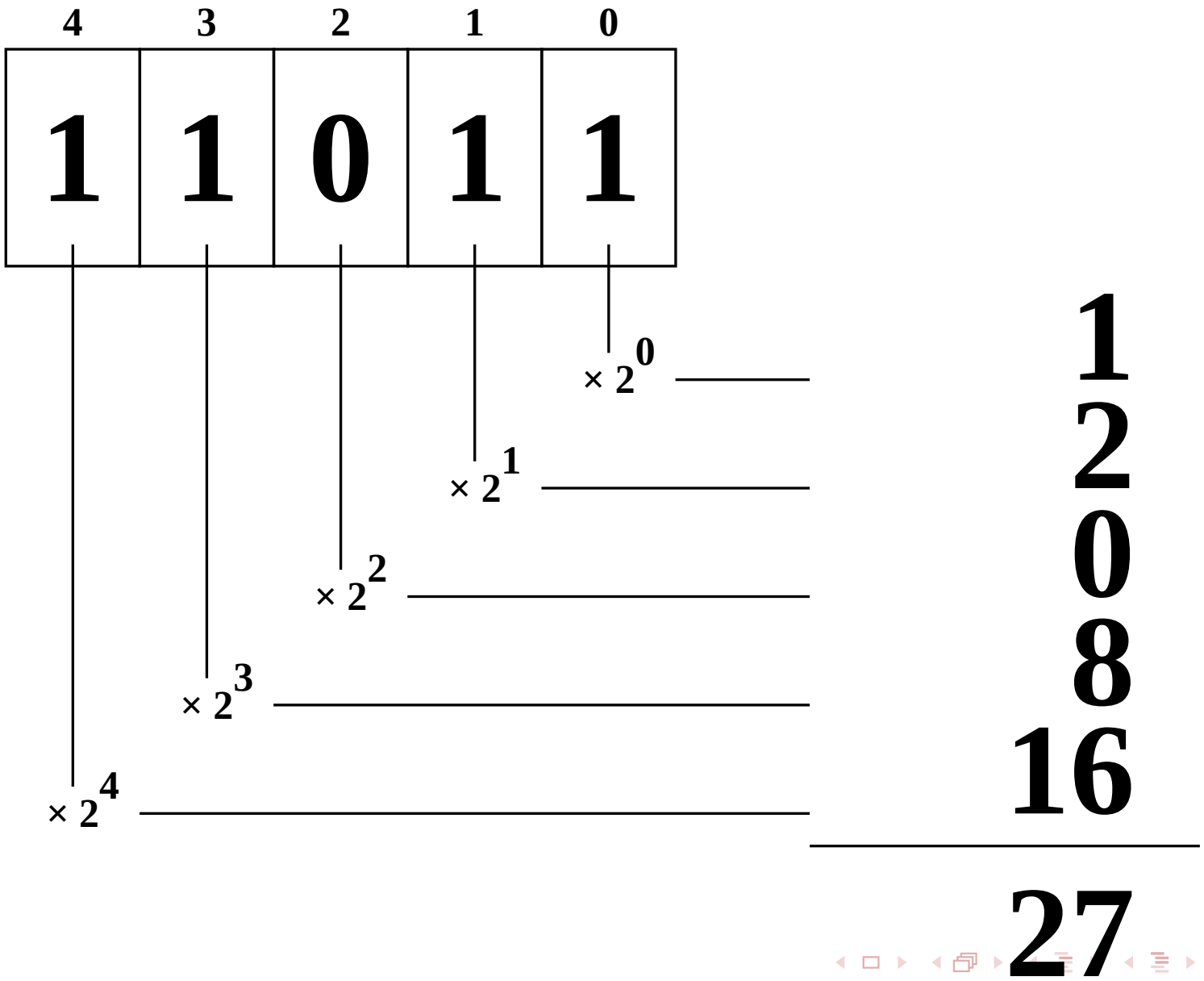
Septiembre 2021

Universidad de Cantabria

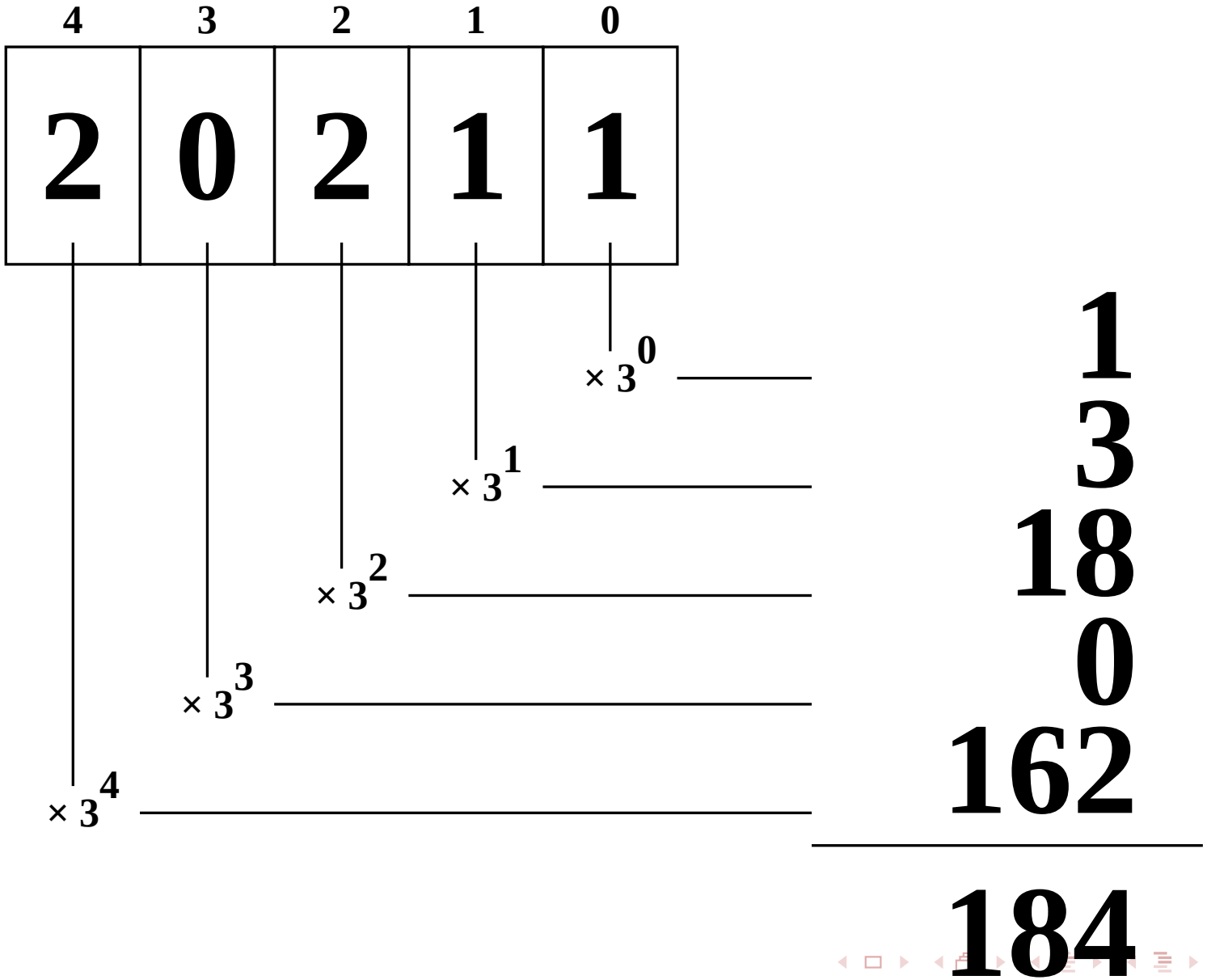
Sistema decimal



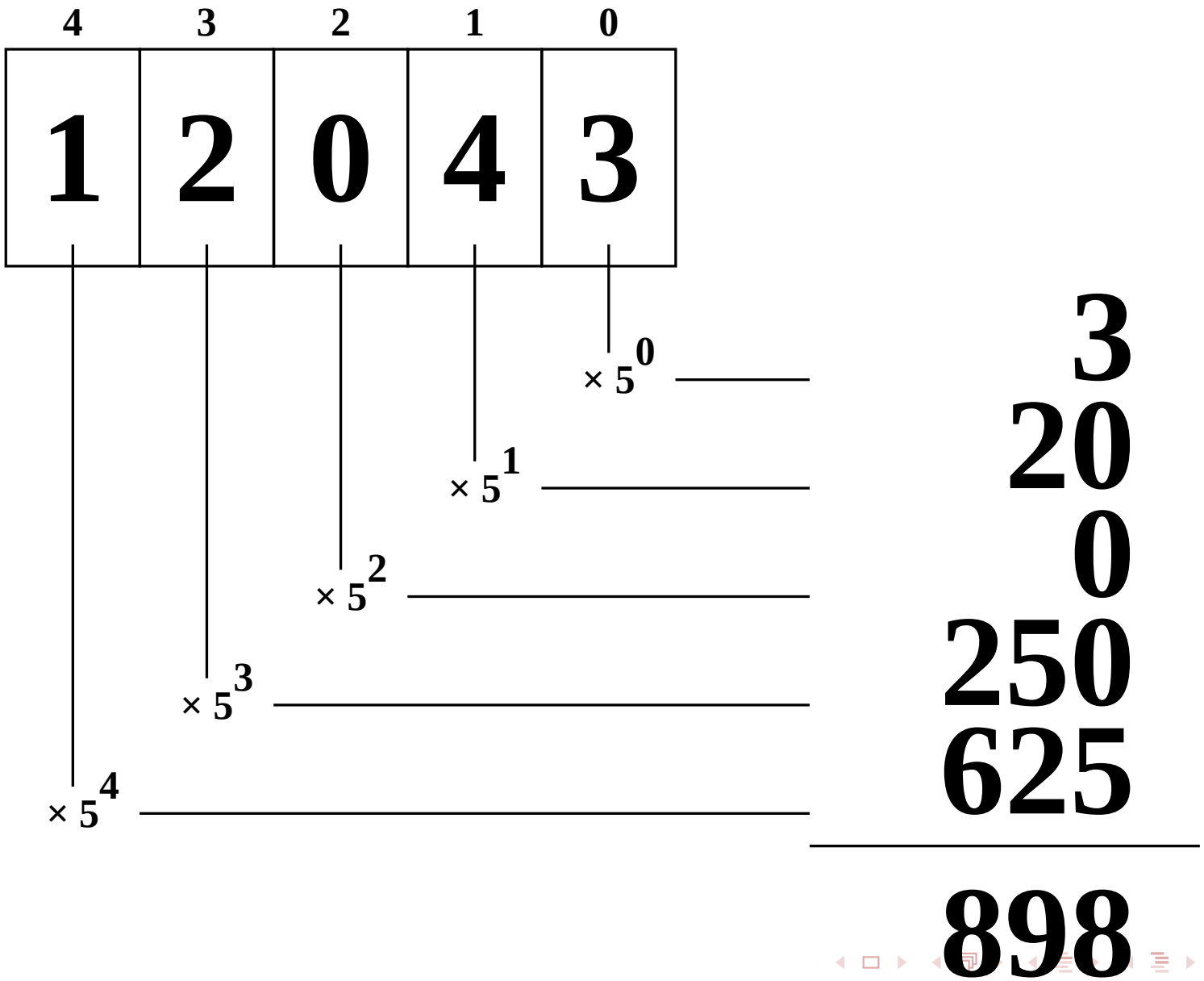
Base 2



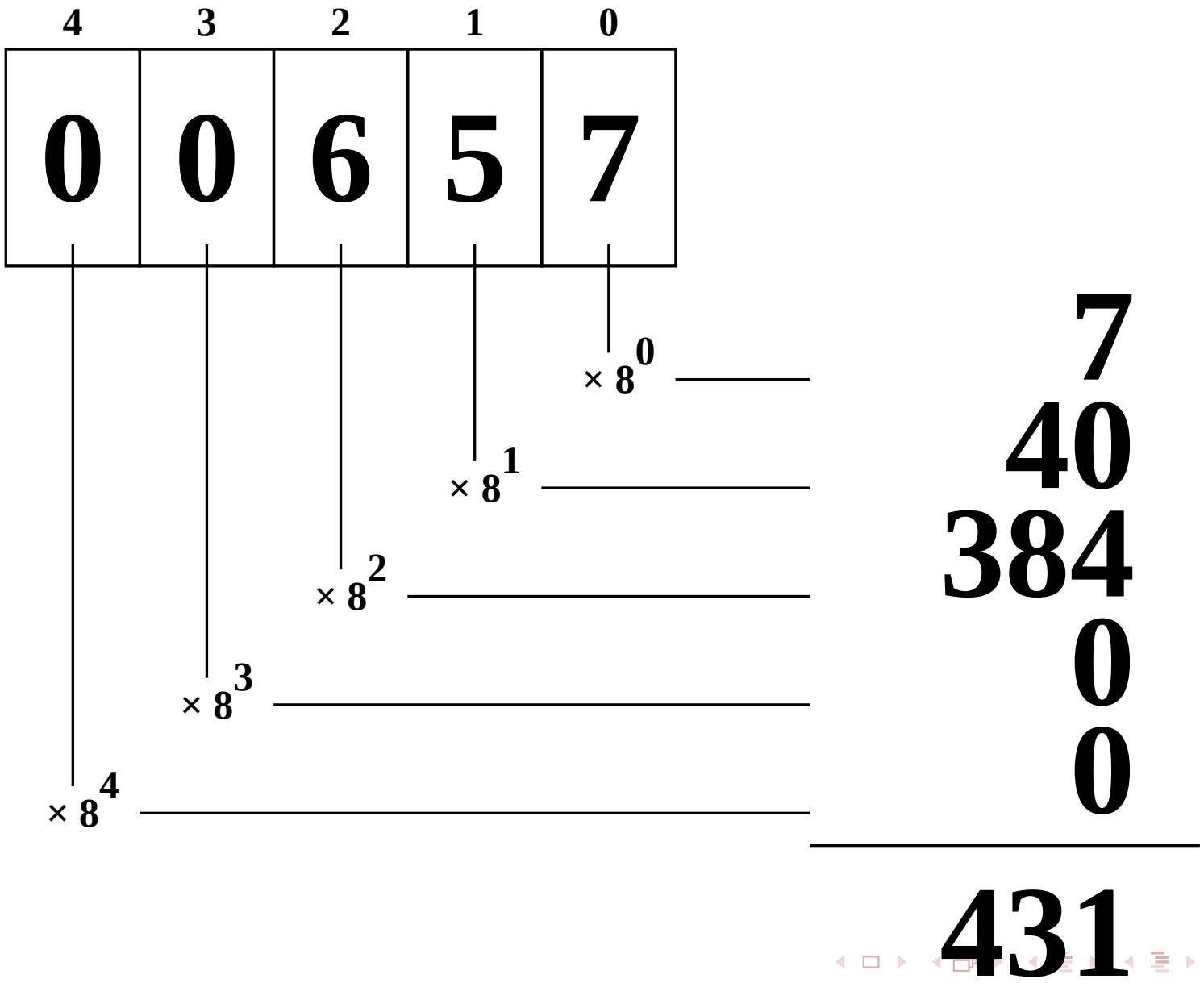
Base 3



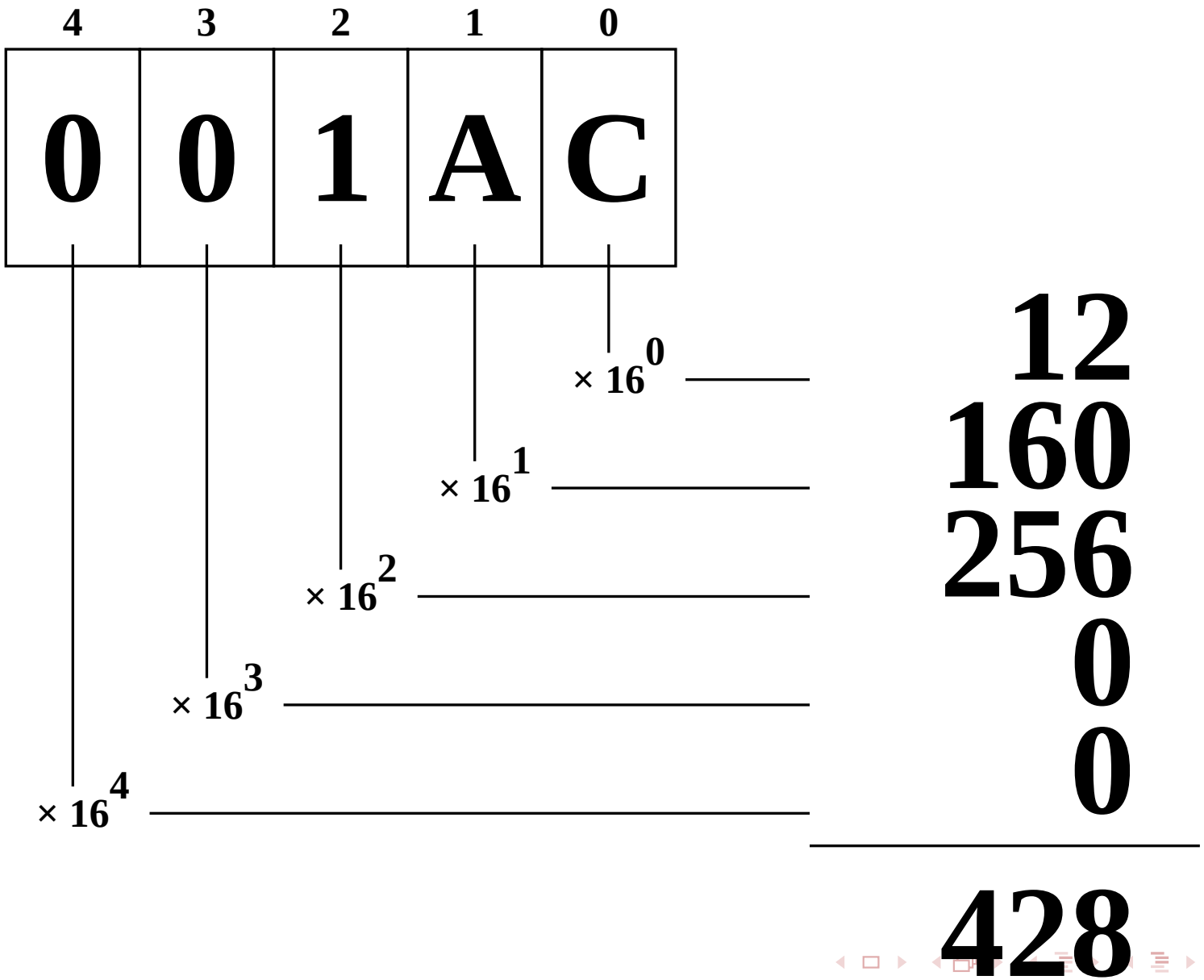
Base 5



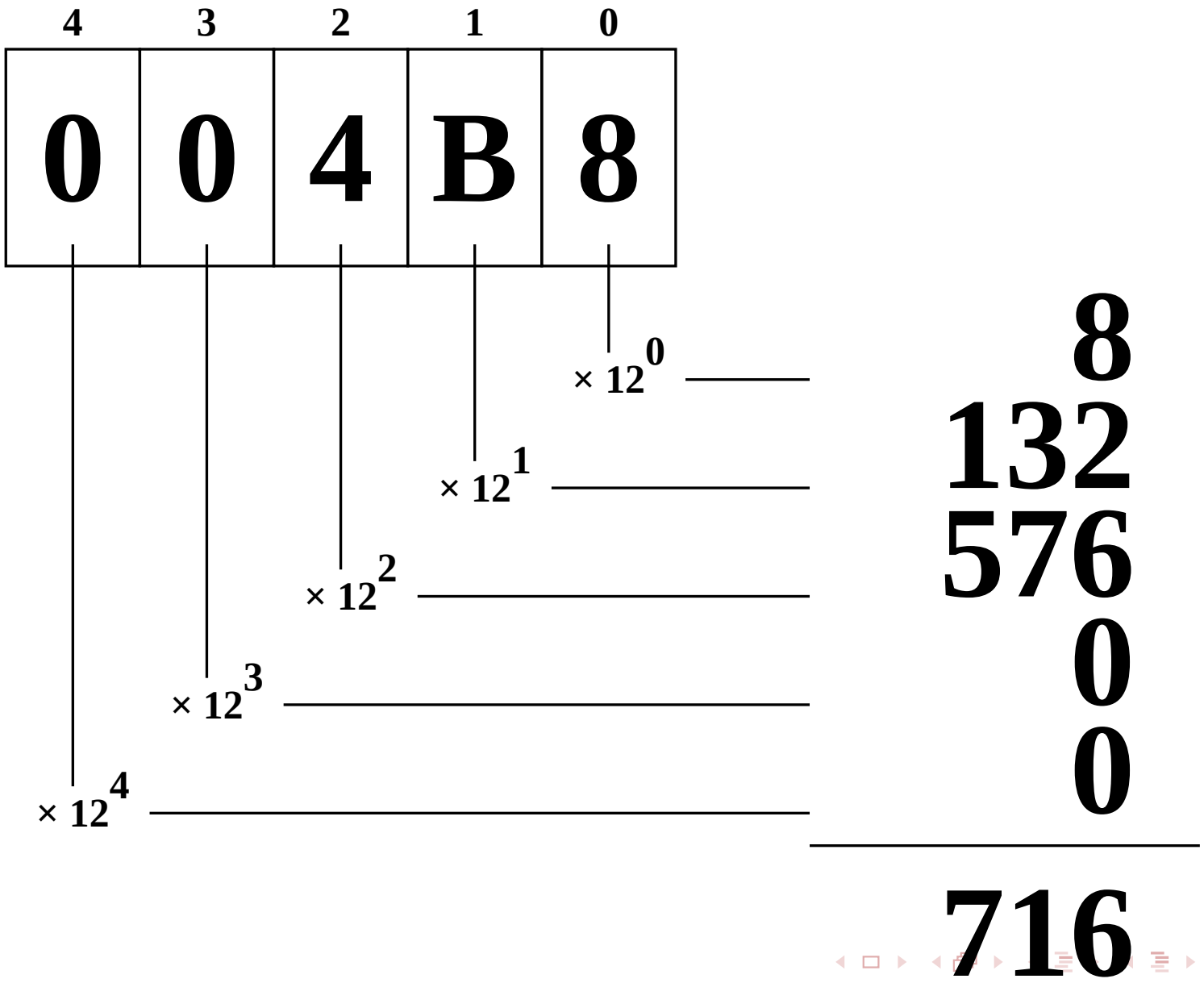
Base 8 (octal)



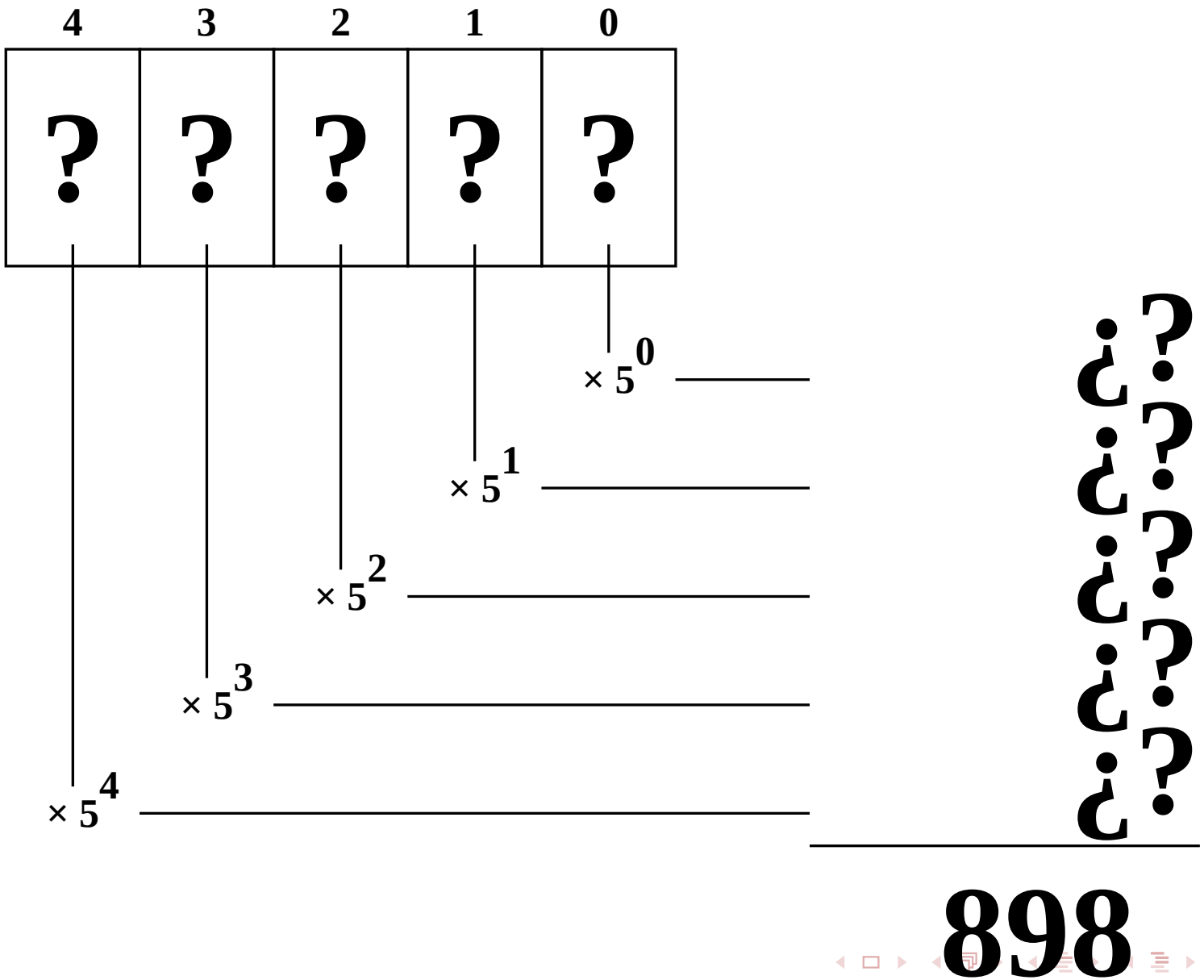
Base 16 (hexadecimal)



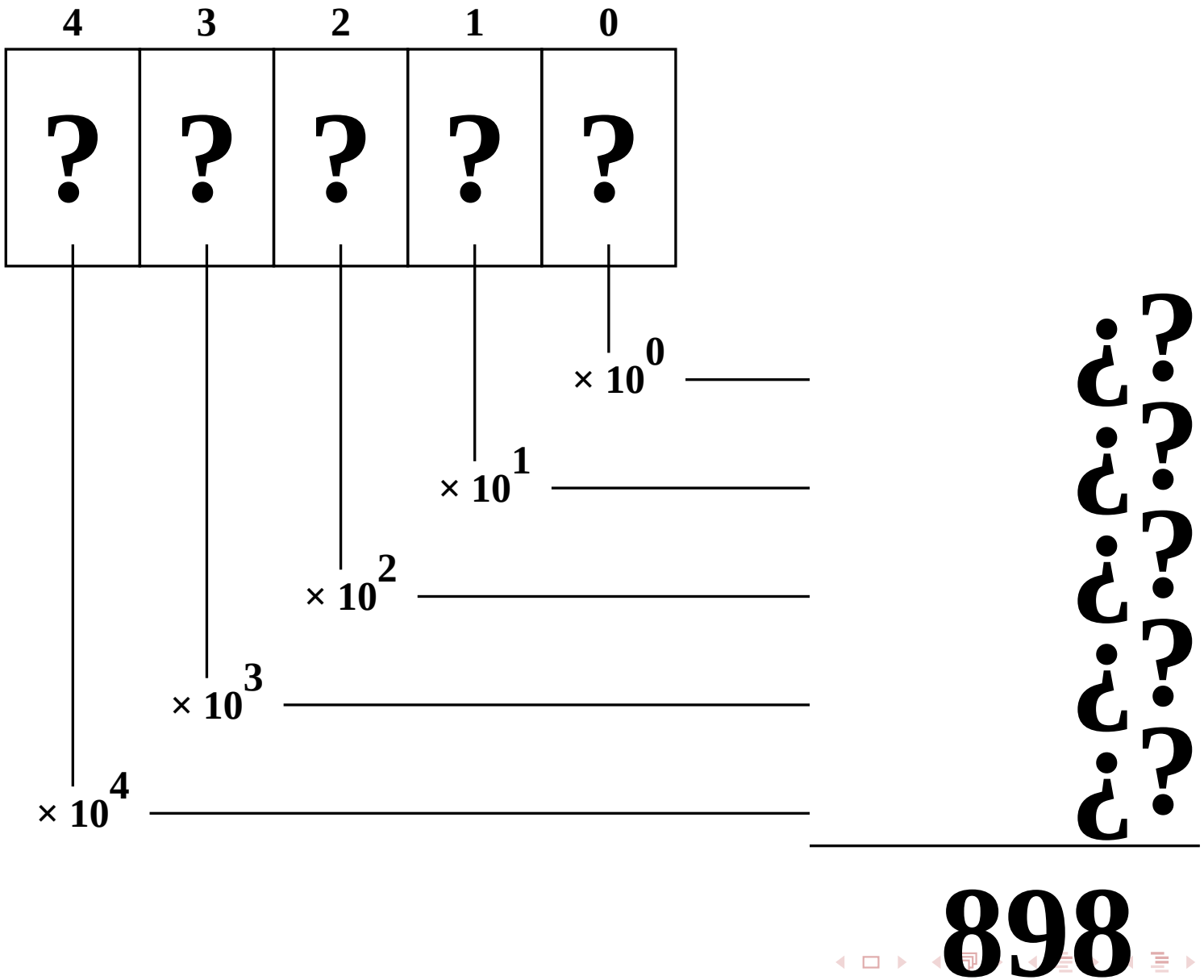
Base 12



Codificación



Codificación



División euclídea

ENTRADA: $\texttt{dividendo}, \texttt{divisor} \in \mathbb{Z}$
$\texttt{divisor} \neq 0$

SALIDA: $\texttt{cociente}, \texttt{resto} \in \mathbb{Z}$, tales que:

$$\texttt{dividendo} = \texttt{divisor} \times \texttt{cociente} + \texttt{resto}$$

$$0 \leq \texttt{resto} < |\texttt{divisor}|$$

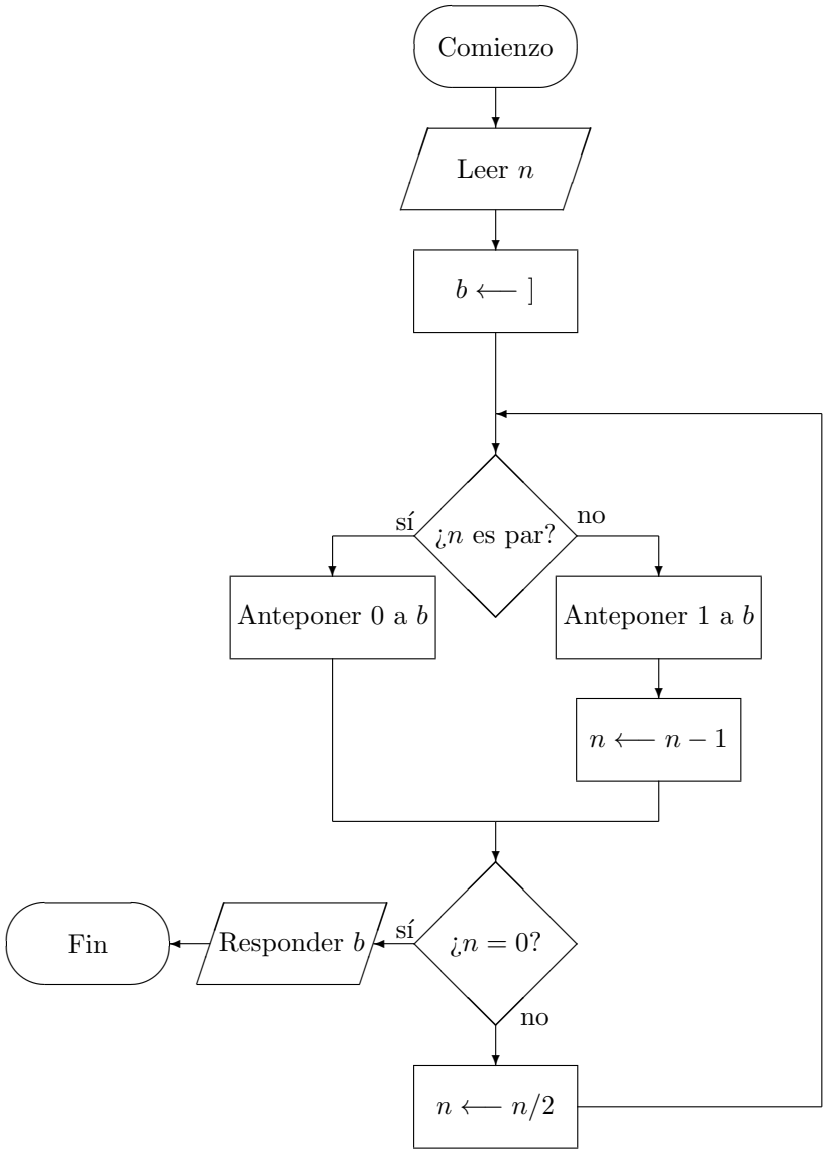
Codificación

- `resto`(898,10)=8 — La cifra menos significativa es 8.
- 898-8=890
- 890/10=89 — Añadimos, a la izq., la codificación de 89.
- `resto`(89,10)=9 — Tomamos la cifra 9.
- 89-9=80
- 80/10=8 — Añadimos, a la izq., la codificación de 8.
- `resto`(8,10)=8 — Tomamos la cifra 8.
- 8-8=0 — Hemos terminado.

Codificación

- resto(898,5)=3 — La cifra menos significativa es 3.
- 898-3=895
- 895/5=179 — Añadimos, a la izq., la codificación de 179.
- resto(179,5)=4 — Tomamos la cifra 4.
- 179-4=175
- 175/5=35 — Añadimos, a la izq., la codificación de 35.
- resto(35,5)=0 — Tomamos la cifra 0.
- 35-0=35
- 35/5=7 — Añadimos, a la izq., la codificación de 7.
- resto(7,5)=2 — Tomamos la cifra 2.
- 7-2=5
- 5/5=1 — Añadimos, a la izq., la codificación de 1.
- resto(1,5)=1 — Tomamos la cifra 1.
- 1-1=0 — Hemos terminado.

Codificación en binario



Funciones de conversión de Python

```
>>> bin(2022)
'0b11111100110'
>>> hex(2022)
'0x7e6'
>>> oct(2022)
'0o3746'
>>> int('111111',2)
63
>>> int('0x7e6',16)
2022
>>> int(bin(2022),2)
2022
>>> int('2022',2021)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: int() base must be >= 2 and <= 36, or 0
>>>
```

Longitud de la codificación

Un número positivo n ocupa exactamente d cifras en base 10 si y solo si:

$$10^{d-1} \leq n < 10^d$$

$$d - 1 \leq \log_{10} n < d$$

$$d - 1 = \lfloor \log_{10} n \rfloor$$

$$d = \lfloor \log_{10} n \rfloor + 1$$

Longitud de la codificación

Un número positivo n ocupa exactamente d cifras en base b si y solo si:

$$b^{d-1} \leq n < b^d$$

$$d - 1 \leq \log_b n < d$$

$$d - 1 = \lfloor \log_b n \rfloor$$

$$d = \lfloor \log_b n \rfloor + 1$$

Números decimales (float)

La representación del número real

$$z = \cdots + n_3 b^3 + n_2 b^2 + n_1 b^1 + n_0 b^0 + n_1 b^{-1} + n_2 b^{-2} + n_3 b^{-3} + \cdots$$

donde la base $b > 2$ y los $n_i \in A = \{0, 1, \ldots, b-1\}$ es

$$z)_b = \ldots . n_3 n_2 n_1 n_0 \cdot n_{-1} n_{-2} n_{-3} \ldots \ldots .$$

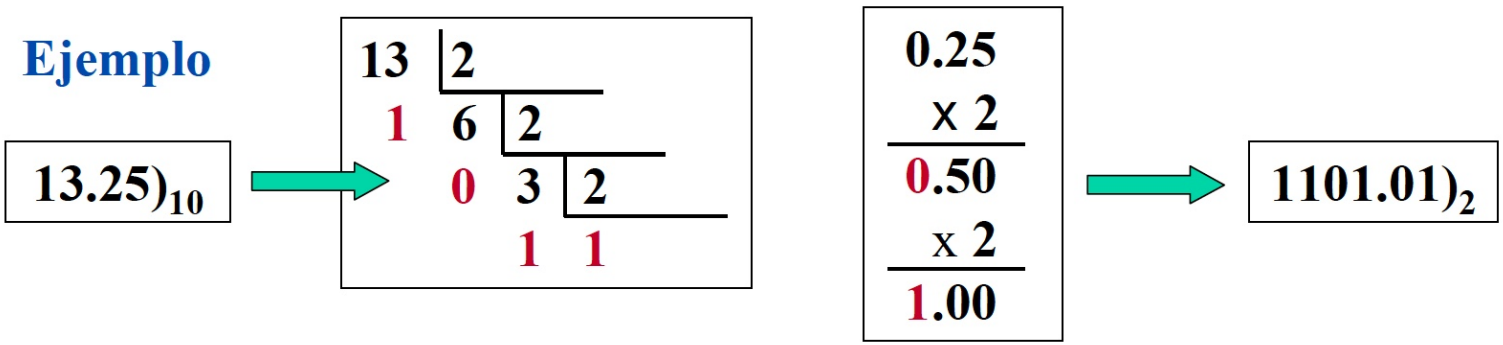
$$8503.25)_{10} = 8 \times 10^3 + 5 \times 10^2 + 0 \times 10^1 + 3 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$$

$$121.02)_3 = 1 \times 3^2 + 2 \times 3^1 + 1 \times 3^0 + 0 \times 3^{-1} + 2 \times 3^{-2} = 16.22222)_{10}$$

$$B2.A2)_{hex} = B \times 16^1 + 2 \times 16^0 + A \times 16^{-1} + 2 \times 16^{-2} = 178.6328125)_{10}$$

Transformar decimal en binario

- Codificamos la parte entera en binario.
- La parte fraccionaria o decimal del número binario se obtiene multiplicando por 2 sucesivamente la parte fraccionaria del número decimal de partida y las partes fraccionarias que se van obteniendo en los productos sucesivos. Las partes enteras (que serán ceros o unos) de los productos obtenidos son las cifras binarias, siendo el bit más significativo el del primer producto, y el menos significativo el del último producto.



Transformar decimal en binario

- Transformar el número $37.2)_{10}$ a binario:
  - $37)_{10} = 100101)_2$
  - $0.2)_{10} = 0.00110011001100110011...)_2$

$$37.2)_{10} = 100101.00110011001100110011...)_2$$

- Transformar el número $0.1)_{10}$ a binario:
  - $0)_{10} = 0)_2$

$$0.1)_{10} = 0.0001100110011.....)_2$$

Los lenguajes de programación convierten números decimales a binario. En general, los números decimales no se pueden representarse de forma exacta en binario, por lo que los resultados no pueden ser exactos.
La codificación utilizada en Python es el *IEEE standard 754* con precisión doble (53 bits).

Código intermedio Octal

Para transformar un número binario a octal se forman grupos de tres cifras binarias a partir del punto decimal hacia la izquierda y hacia la derecha. Se efectúa la conversión a octal de cada grupo individual.

Decimal	Binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

De octal a binario se pasa sin más que convertir individualmente a binario (tres bits) cada cifra octal, manteniendo el orden del número original.

$702.27)_8$ → $111\ 000\ 010\ .\ 010\ 111)_2$

$011\ 101\ 001\ .\ 110\ 100)_2$ → $351.64)_8$

Código intermedio Hexadecimal

Para transformaciones entre hexadecimal y binario se procede de forma análoga al caso octal pero formando grupos de cuatro cifras binarias a partir del punto decimal hacia la izquierda y hacia la derecha. La computadora utiliza este tipo de notaciones intermedias internamente o como entrada/salida.

Hexadecimal	Decimal	Binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

$$\text{CA7.B2})_{16} \rightarrow 1100\ 1010\ 0111\ .\ 1011\ 0010)_2$$

$$0110\ 1101\ 0011\ .\ 0110\ 0100)_2 \rightarrow \text{6D3.64})_{16}$$

Operaciones aritméticas en binario

Suma

```
1 111  111   ← acarreos
 1011 0111
+0111 0101
-----------
1 0010 1100  ← resultado
```

Resta

```
 1 0 0 1 0  1 1 0 0
-  1 0 1 1  0 1 1 1
-------------------
 1 1 1 1    1 1 1    ← adeudos
-------------------
 0 0 1 1 1  0 1 0 1  ← resultado
```

Producto

```
     1011 0111
          x101
-----------------
     1 0110 0111
 101 1011 0111
-----------------
 111 0010 0011   ← resultado
```

División

```
 10110111 | 101
-101      |-----
-------   | 100100.1001
 000101
   -101
 -------
  000110
    -101
 -------
  001000
```

Operaciones aritméticas en binario

```
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 0.1+0.1+0.1
0.30000000000000004
>>>
```

Python realiza las operaciones en binario y convierte el resultado de nuevo en decimal. En el redondeo (función round) los errores pueden acumularse.

Python solamente muestra una aproximación decimal al valor verdadero de la aproximación binaria almacenada por la máquina. En el caso 0.1 la fracción es

$$\frac{3602879701896397}{2^{55}} = 0.1000000000000000055511151231257827021181583404541$$

```
>>> round(0.1+0.1+0.1,1)
0.3
>>> 5/6
0.8333333333333334
>>> round(5/6,2)
0.83
>>>
```