S1-SistemasNumeracion

October 13, 2025

1 ICCD332 - Arquitectura de Computadores

Fecha: 2024-10-07

Profesor: Lenin G. Falconí

Horario: LU 7-9 y SA 7-9

1.1 Sistemas de Numeración

- Un número se puede representar como una cadena de dígitos $b_{n-1} \dots b_2 b_1 b_0 \cdot b_{-1} b_{-2} \dots b_m$
- La posición relativa i del dígito posee un **peso** r^i, donde r es la **base** del sistema de numeración
- El alfabeto de dígitos posibles en un sistema de base r es $0 \leq b_i \leq r-1$
- Por tanto un número N en la base r, notado como N_r se representa como:

$$N_r = \sum_{i=-m}^{n-1} b_i \times r^i$$

1.2 Tipos de Sistemas de Numeración

Denominación	base
Decimal	10
Octal	8
Binario	2
Hexadecimal	16

1.3 Ejercicio 1 - Conversión de Número en base r a decimal -

- Escriba un número en base 7
- Escriba un número en base 4
- ¿Qué valor sería el correspondiente en decimal?

[1]:
```python
#35.23 en base 7
print("35.23 en base 7 equivale a:")
print(3*7**1+5*7**0+2*7**(-1)+3*7**(-2))
#123.30 en base 4
print("123.30 en base 4 equivale a:")
print(1*4**2+2*4**1+3*4**0+3*4**(-1)+0*4**(-2))
```

```
35.23 en base 7 equivale a:
26.346938775510203
123.30 en base 4 equivale a:
27.75
```

1.4 Sistema de Numeración Decimal

- La base del sistema es $r = 10$
- Los dígitos son $0 \leq b_i \leq 9$ y $b_i \in \mathbb{Z}^+$

$123.45 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-1}$

1.5 Sistema de Numeración Binario

- La base del sistema es $r = 2$
- Los dígitos son $0 \leq b_i \leq 1$ y $b_i \in \mathbb{Z}^+$
- Los dígitos 1 y 0 en binario tienen el mismo valor en notación decimal i.e.

$0_2 = 0_{10}$

$1_2 = 1_{10}$

Por tanto:

$1101.01_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$

1.6 Conversión Sistema de base r a Decimal $N_r \rightarrow X_{10}$

Sea $N_r \rightarrow X_{10}$ la conversión de un número en base r a su equivalente Decimal. En este caso, se aplica:

$X_{10} = \sum_i b_i \times r^i$

1.6.1 Ejercicio 2 $N_r \rightarrow X_{10}$:

- $(4021.2)_5$
- $(127.4)_8$
- $(110101)_2$
- $(B65F)_{16}$

[ ]:
```
# 4021.2 en base 5 a base 10
4*5**3+2*5**1+1*5**0+2*5**-1
```

[ ]:
```
511.4
```

1.7 Conversión Decimal a Binario $N_{10} \rightarrow X_2$

Dado un numero decimal que dispone tanto de parte entera como fraccionaria, se convierten por separado la **parte entera** y la **parte fraccionaria**

1.7.1 Parte Entera

- Sea $X_2 = b_{m-1}b_{m-2} \dots b_2b_1b_0 \cdot b_{-1}b_{-2} \dots$ el número binario buscado
- Sea N el número decimal dado
- La parte entera es: $\lfloor b_{m-1}b_{m-2} \dots b_2b_1b_0 \rfloor = b_{m-1}\times 2^{m-1} + b_{m-2}\times 2^{m-2} + \dots b_2\times 2^2 + b_1\times 2^1 + b_0\times 2^0$
- Dividir N por la base 2 obtiene un cociente N_1 y un residuo R_0 i.e. $N = 2 \times N_1 + R_0$
- Repita el proceso anterior para cada cociente y guarde los residuos hasta alcanzar un cociente de 0
- El conjunto de residuos en orden inverso es el número buscado

1.7.2 Demostración

$N = 2 \times N_1 + R_0$

$N_1 = 2 \times N_2 + R_1$

$N_2 = 2 \times N_3 + R_2$

...

$N_{m-1} = 2 \times N_m + R_{m-1}$

Entonces N es:

$N = 2 \times (2 \times (\dots + R_2) + R_1) + R_0$

$N = 2^m N_m + 2^{m-1} R_{m-1} + \dots + 2^2 R_2 + 2^1 R_1 + R_0$

Pero $N_m = 0$ y $R_{m-1} = 1$

1.7.3 Ejercicio 3 - Conversión de $N_{10} \rightarrow X_r$

1. Convertir 41 a binario
2. Convertir 153 a octal
3. Convertir 256 a hexadecimal

[3]:
```python
# 41 a binario
print("Convertir 41 a binario.")
print(41//2, 41%2)
print(20//2, 20%2)
print(10//2, 10%2)
print(5//2, 5%2)
print(2//2, 2%2)
print(1//2, 1%2)
resp = '0b101001'
resp==bin(41)
#153 a octal
print("Convertir 41 a binario.")
print(153//8, 153%8)
print(19//8, 19%8)
print(2//8, 2%8)
resp = '0o231'
resp==oct(153)
```

```python
#256 a hexadecimal
print("Convertir 256 a hexadecimal.")
print(256//16, 256%16)
print(16//16, 16%16)
print(1//16, 1%16)
resp = '0x100'
resp==hex(256)
```

```
Convertir 41 a binario.
20 1
10 0
5 0
2 1
1 0
0 1
Convertir 41 a binario.
19 1
2 3
0 2
Convertir 256 a hexadecimal.
16 0
1 0
0 1
```

[3]:
```
True
```

1.7.4 Parte Fraccionaria

Sea $F = 0 \cdot b_{-1} b_{-2} \ldots$ la parte fraccionaria buscada en binario. Entonces

$0 \cdot b_{-1} b_{-2} \cdots = b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \ldots$

Se observa que la parte fraccionaria puede obtener factor común

$2^{-1} \times (b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + \ldots) \ldots))$

Si se multiplica por la base 2, entonces

$2 \times F = b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + \ldots) \ldots)$

Que se puede escribir como

$2 \times F = b_{-1} + F_1$

con $F_1 = 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + \ldots) \ldots)$

Entonces:

1. Multiplique la parte fraccionaria por la base y tome la parte entera
2. Repita 1 con la nueva parte fraccionaria hasta obtener 0 o la precisión deseada
3. El valor binario deseado es el conjunto de partes enteras obtenidos

1.7.5 Ejercicio 4 - Conversión de Parte Fraccionaria -

1. $(0.6875)_{10} \to X_2$
2. $(0.513)_{10} \to X_8$

[ ]:
```
v1 = int(0.513*8), 0.513*8
v2 = int(0.104*8), 0.104*8
v3= int(0.832*8), 0.832*8
v1,v2,v3
```

[ ]:
```
((4, 4.104), (0, 0.832), (6, 6.656))
```

1.8 Sistema Hexadecimal

A continuación la tabla de valores de dígitos hexadecimales

[ ]:
```
print(f"decimal \tbinario \toctal \thexadecimal")
for i in range(16):
    print(f"{i} \t\t{bin(i)} \t\t{oct(i)}\t\t{hex(i)}")
```

```
decimal         binario         octal   hexadecimal
0               0b0             0o0             0x0
1               0b1             0o1             0x1
2               0b10            0o2             0x2
3               0b11            0o3             0x3
4               0b100           0o4             0x4
5               0b101           0o5             0x5
6               0b110           0o6             0x6
7               0b111           0o7             0x7
8               0b1000          0o10            0x8
9               0b1001          0o11            0x9
10              0b1010          0o12            0xa
11              0b1011          0o13            0xb
12              0b1100          0o14            0xc
13              0b1101          0o15            0xd
14              0b1110          0o16            0xe
15              0b1111          0o17            0xf
```

1.9 Conversión Binario - Octal - Hexadecimal

Para esta conversión es importante recordar:

- $2^3 = 8$
- $2^4 = 16$

En consecuencia para convertir un número binario a octal o decimal, notado esto por $N_2 \to X_8$ y $N_2 \to X_{16}$, respectivamente, se ha de localizar el punto decimal y apartir de el se forman grupos de 3 (octal) o grupos de 4 (hexadecimal).

Sea $N_2 = 10101001.10111$

1.9.1 Conversión a Octal

1. Se hacen grupos de 3 digitos a partir de la coma en ambas direcciones y se completa el dígito faltante con 0

010 101 001.101 110

2. A partir de la tabla de equivalencias realice la sustituciones de valores

2 5 1.5 6

Entonces: $N_2 = 10101001.10111_2 \rightarrow 251.56_8$

1.9.2 Conversión a Hexadecimal

1. Se hacen grupos de 4 digitos a partir de la coma en ambas direcciones y se completa el dígito faltante con 0

1010 1001.1011 1000

2. A partir de la tabla de equivalencias realice la sustituciones de valores

$A9.B8$

Entonces: $N_2 = 10101001.10111_2 \rightarrow A9.B8_{16}$

1.10 Órdenes de Magnitud de datos

- $2^0 \rightarrow bit$
- $2^3 \rightarrow byte$
- $2^{10} \rightarrow Kilo$
- $2^{20} \rightarrow Mega$
- $2^{30} \rightarrow Giga$
- $2^{40} \rightarrow Tera$

1.11 Ejercicio 5

¿Qué valores decimales corresponden los órdenes de magnitud antes indicados? Escribe un programa que devuelva los valores de las potencias de 2 antes indicadas.

[ ]:
```
def orden_magnitud2(potencias=[0,3,10,20,30,40], nombres=['bit', 'byte',␣
↪'Kilo', 'Mega', 'Giga', 'Tera']):
    for i, potencia in enumerate(potencias):
        print(f"{nombres[i]} -> {2**potencia}")
```

[ ]:
```
orden_magnitud2()
```

```
bit -> 1
byte -> 8
Kilo -> 1024
Mega -> 1048576
Giga -> 1073741824
Tera -> 1099511627776
```

1.12 Taller

1. Construir una función en python que permita convertir un número en una base r a su equivalente decimal
2. Construir una función en python que permita convertir un número decimal a binario. Divida el problema en dos partes 1. parte entera y 2. parte fraccionaria.
3. Generalice el programa anterior para convertir de decimal a un sistema con base r

[1]:
```python
def convertir_a_decimal(numero, base):
    return int(numero, base)

# Ejemplo:
print(convertir_a_decimal("1011", 2))
```

```
11
```

[2]:
```python
def decimal_a_binario(num, precision=10):
    #parte entera
    entera, fraccion = int(num), num - int(num)
    bin_ent = bin(entera)[2:]

    #parte fraccionaria
    bin_frac = ""
    while fraccion > 0 and len(bin_frac) < precision:
        fraccion *= 2
        bin_frac += str(int(fraccion))
        fraccion -= int(fraccion)

    #Combinar partes
    return bin_ent + ('.' + bin_frac if bin_frac else '')

# Ejemplo:
print(decimal_a_binario(10.625))
```

```
1010.101
```

[3]:
```python
def decimal_a_base(num, base, precision=10):
    digitos = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    # Parte entera
    ent, frac = int(num), num - int(num)
    bin_ent = "" if ent else "0"
    n = ent
    while n:
        bin_ent = digitos[n % base] + bin_ent
        n //= base

    # Parte fraccionaria
    bin_frac = ""
```

```python
    for _ in range(precision):
        frac *= base
        d = int(frac)
        bin_frac += digitos[d]
        frac -= d
        if frac == 0: break

    return bin_ent + ('.' + bin_frac if bin_frac else '')

# Ejemplos:
print(decimal_a_base(10.625, 2))   # 1010.101
```

```
1010.101
```