Arquitectura del Computador y Sistemas Operativos

Primera Clase

SO

01

Lógica Booleana

02

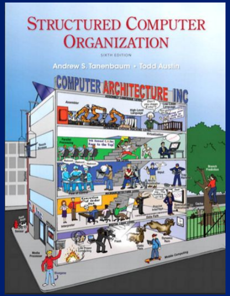
Arquitectura del Computador

03

Fundamentos de Sistemas Operativos

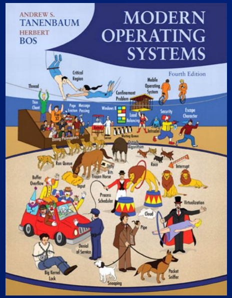
Bibliografía

01



Structured Computer Organization
Tanenbaum - Austin

02



Modern Operating Systems
Tanenbaum - Bos

SO

01

Lógica Booleana

SO

Representación numérica

Base 10: Diez símbolos, 0 - 9

000
001
002
003
004
005
006
007
008
009
Último símbolo
010
011
....

$023 = 0 \,.\, 10^2 + 2 \,.\, 10^1 + 3 \,.\, 10^0 = 23$

Base 2: Dos símbolos, 0 y 1

000
001
Último símbolo
010
011
....

$011 = 0 \,.\, 2^2 + 1 \,.\, 2^1 + 1 \,.\, 2^0 = 3$

Números Enteros (1/2)

Enteros sin signo: Se almacenan como una secuencia de bits significativos. Por ejemplo:

$$0100\ 0010 = 1 \,.\, 2^{6} + 1 \,.\, 2^{1} = 66$$

Enteros con signo: Se interpreta el bit más significativo como signo (0=positivo, 1=negativo). Sin embargo ésto requeriría un procedimiento especial para sumar:

$$\begin{array}{lll} & 0000\ 0001 & = +1 \\ + & & \\ & 1000\ 0001 & = -1 \\ \hline & 1000\ 0010 & \neq \ 0 \end{array}$$

No es lo deseado!

Números Enteros (2/2)

El complemento a la base menos uno (complemento a 1): Se propone que el negativo de un número se encuentre negando cada uno de los bits del mismo

```
                                                          1111|1
   0000 0001    = +1       0000 0100    = +4       0000 0100    = +4
 +                       +                       +
   1111 1110    = -1       1111 1011    = -4       1111 1110    = -1
   ---------               ---------               ---------
   1111 1111    ≠  0       1111 1111    ≠  0       0000 0010    ≠  3
```

El complemento a la base (complemento a 2): El negativo se encuentra negando todos los bits y luego sumando uno:

```
 11111 111                 1111 1                  1111 1
   0000 0001    = +1       0000 0100    = +4       0000 0100    = +4
 +                       +                       +
   1111 1111    = -1       1111 1100    = -4       1111 1111    = -1
   ---------               ---------               ---------
   0000 0000    =  0       0000 0000    =  0       0000 0011    =  3
```

SO

Notación Hexadecimal

Los números binarios, al tener una base chica, tienen muchos dígitos. Veamos un ejemplo:

1246 = 0b10011011110

Por ese motivo no son fáciles de recordar ni de decir. Entonces, lo que se utiliza es la notación hexadecimal, o sea base 16. Al ser $16 = 2^4$, es muy fácil pasar de base 2 a base 16, dado que 4 dígitos binarios se convierten en 1 dígito hexadecimal.

Antes de seguir adelante, base 16 implica que debe haber 16 símbolos. Se usan los diez dígitos (0-9) y se agregan las primeras 6 letras (A-F) para completar.

Para el caso del número anterior:

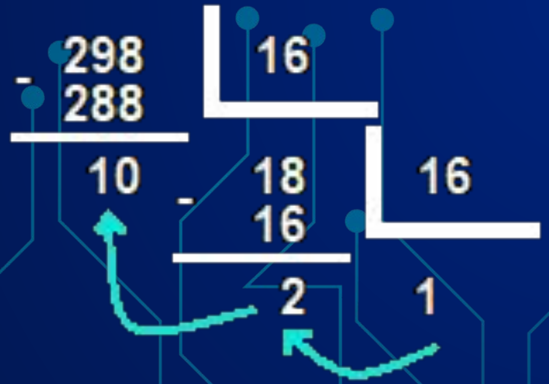
0b 100 1101 1110 = 0x4DE

Al ser base 16, los números tiene incluso menos dígitos que los de base 10.

Pasaje entre bases

Ya vimos cómo pasar de cualquier base a la decimal, haciendo una suma de los dígitos multiplicados por la base a su respectiva potencia.

Ahora veamos cómo pasar un número decimal a cualquier base. Ésto puede hacerse mediante una división sucesiva por la base:



288 = 0x12A

SO

Números fraccionarios (1/2)

Se podría usar un esquema de punto fijo, o sea almacenar un número entero y asumir que el separador decimal está en un lugar específico.

En cálculos complejos a menudo se opera con números grandes y pequeños, que tendrían formatos distintos (distinta posición del separador de decimales).

Resulta mucho más práctico utilizar un formato de punto flotante. En el cual cada número fraccionario se almacena de la forma:

$$\text{Signo} \,.\, \text{Mantissa} \,.\, \text{Base}^{\text{Exponente}}$$

Ahora exponentes negativos nos permiten números pequeños, mientras que exponentes positivos permiten almacenar números grandes.

La precisión del número, o sea la cantidad de dígitos significativos que se almacenan, depende de la cantidad de dígitos de la mantisa.

Números fraccionarios (2/2)

El estándar IEEE 754 define el estándar para el almacenamiento de números de punto flotante.

Define varios formatos, algunos con base 2 y otros con base 10. Además, varios tamaños estándar para la cantidad de dígitos en la mantisa. Los más usados son:

Precisión	Signo	Exponente	Mantissa
Half	1 bit	5 bits	10 bits
Single	1 bit	8 bits	23 bits
Double	1 bit	11 bits	52 bits
Quad	1 bit	15 bits	112 bits
Extended x87	1 bit	15 bits	63 bits

Circuitos combinacionales (1/3)

Se caracterizan porque la salida es función de la(s) entrada(s). En otras palabras no tienen memoria.

Dijimos que la salida es *función* de la(s) entrada(s). Para definir una función es necesario conocer primero las operaciones que pueden ser usadas para armar la función:

La compuerta NOT:

Entrada	Salida
0	1
1	0

Circuitos combinacionales (2/3)

La compuerta OR:

Entrada 1	Entrada 2	Salida
0	0	0
0	1	1
1	0	1
1	1	1

La compuerta AND:

Entrada 1	Entrada 2	Salida
0	0	0
0	1	0
1	0	0
1	1	1

Circuitos combinacionales (3/3)

La compuerta XOR:

Entrada 1	Entrada 2	Salida
0	0	0
0	1	1
1	0	1
1	1	0

Una función sencilla:

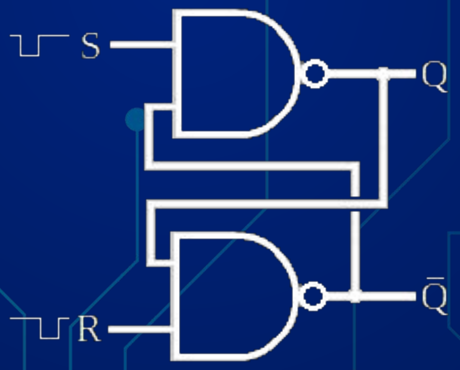
Entrada 1	Entrada 2	Salida
0	0	1
0	1	0
1	0	1
1	1	1

SO

Flip Flops (1/6)

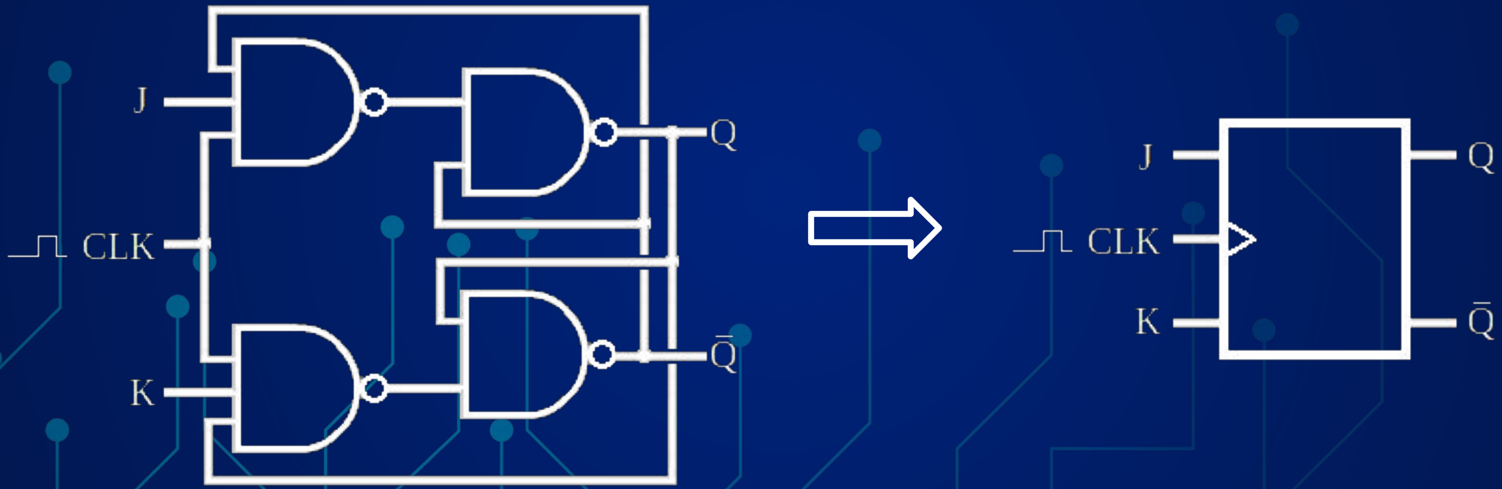
Los Flip Flops son conjuntos de compuertas en los que la salida es usada como una entrada más (o aunque no sea la entrada - por lo menos en alguna compuerta intermedia). Estos circuitos ya no se llaman combinacionales, porque la salida ya no depende solamente de las entradas, sino del estado anterior de la(s) salida(s). Decimos que estos circuitos tienen *memoria*.

El Flip Flop RS:



Flip Flops (2/6)

El Flip Flop JK:



Fin

¿Preguntas?