Índice de Contenidos

Definición de Sistema Informático

Un sistema informático (SI) es un sistema que permite almacenar y procesar información; es el conjunto de partes interrelacionadas: hardware, software y personal informático. El hardware incluye computadoras o cualquier tipo de dispositivo electrónico, que consisten en procesadores, memoria, sistemas de almacenamiento externo, de comunicación, etc. El software incluye al sistema operativo, firmware y aplicaciones. Por último, el soporte humano incluye al personal técnico que crean y mantienen el sistema (analistas, programadores, operarios, etc.) y a los usuarios que lo utilizan.[1]

Sistemas de Información digital

Las computadoras solo son capaces de almacenar y procesar secuencias de datos binarios (0 y 1). Dependiendo del soporte, estos datos de almacenarán en forma de campos eléctricos, magnéticos y ópticos entre otros. Sin embargo, nuestra apreciación del mundo es analógica (continua, no discreta), por lo que es necesario realizar una conversión entre los datos que queremos almacenar y procesar a su representación binaria o digital.

Conversión de decimal a binario

Convertir un número decimal al sistema binario es muy sencillo: basta con realizar divisiones sucesivas por 2 y escribir los restos obtenidos en cada división en orden inverso al que han sido obtenidos.[2]

Por ejemplo, para convertir al sistema binario el número 77_{10} haremos una serie de divisiones que arrojarán los restos siguientes:

77 : 2 = 38 Resto: 1

38 : 2 = 19 Resto: 0

19 : 2 = 9 Resto: 1

9 : 2 = 4 Resto: 1

4 : 2 = 2 Resto: 0

2 : 2 = 1 Resto: 0

1 : 2 = 0 Resto: 1

y, tomando los restos en orden inverso obtenemos la cifra binaria:

$77_{10} = 1001101_2$

Ejercicio:

Expresa, en código binario, los números decimales siguientes: 191, 25, 67, 99, 135, 276

El tamaño de las cifras binarias

La cantidad de dígitos necesarios para representar un número en el sistema binario es mayor que en el sistema decimal. En el ejemplo del párrafo anterior, para representar el número 77,

[1] https://es.wikipedia.org/wiki/Sistema_informático

[2] http://platea.pntic.mec.es/~lgonzale/tic/binarios/numeracion.html#Conversi%F3n_entre_n%FAmeros_decimales_y

que en el sistema decimal está compuesto tan sólo por dos dígitos, han hecho falta siete dígitos en binario.

Para representar números grandes harán falta muchos más dígitos. Por ejemplo, para representar números mayores de 255 se necesitarán más de ocho dígitos, porque $2^8 = 256$ y podemos afirmar, por tanto, que 255 es el número más grande que puede representarse con ocho dígitos.

Como regla general, con n dígitos binarios pueden representarse un máximo de 2^n, números. El número más grande que puede escribirse con n dígitos es una unidad menos, es decir, $2^n - 1$. Con cuatro bits, por ejemplo, pueden representarse un total de 16 números, porque $2^4 = 16$ y el mayor de dichos números es el 15, porque $2^4-1 = 15$.

Para saber de antemano cuantos bits necesitaremos para representar un número podemos calcular el logaritmo en base 2 de dicho número y tomar el entero inmediatamente posterior. Por ejemplo para representar el número 15873_{10} necesitaremos log2 15873=13,95-> 14 dígitos binarios (11111000000001).

Ejercicio:

Averigua cuántos números pueden representarse con 8, 10, 16 y 32 bits y cuál es el número más grande que puede escribirse en cada caso.

Ejercicio:

Dados dos números binarios: 01001000 y 01000100 ¿Cuál de ellos es el mayor? ¿Podrías compararlos sin necesidad de convertirlos al sistema decimal?

Conversión de binario a decimal

El proceso para convertir un número del sistema binario al decimal es aún más sencillo; basta con desarrollar el número, teniendo en cuenta el valor de cada dígito en su posición, que es el de una potencia de 2, cuyo exponente es 0 en el bit situado más a la derecha, y se incrementa en una unidad según vamos avanzando posiciones hacia la izquierda.

Por ejemplo, para convertir el número binario 10100112 a decimal, lo desarrollamos teniendo en cuenta el valor de cada bit:

1*26 + 0*25 + 1*24 + 0*23 + 0*22 + 1*21 + 1*20 = 83

$1010011_2 = 83_{10}$

Ejercicio:

Expresa, en el sistema decimal, los siguientes números binarios:

110111, 111000, 010101, 101010, 1111110

Sistema de representación hexadecimal

Para los humanos resulta fácil cometer errores a la hora de manejar cifras binarias, (imaginemos tener que transcribir por ejemplo el número 1001001010101010010101001010101010101001001010010111010101010100) es por ello que en las ocasiones que se han de manejar este tipo de datos, se suele hacer con el sistema de representación hexadecimal (La cadena anterior en notación hexadecimal es 92AA54AAA9297554).

Hay que tener en cuenta que los datos binarios no solo se emplean para representar números decimales (menos aún enteros) sino cualquier tipo de información.

En el sistema hexadecimal los números se representan con dieciséis símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. Se utilizan los caracteres A, B, C, D, E y F representando las cantidades decima-les 10, 11, 12, 13, 14 y 15 respectivamente, porque no hay dígitos mayores que 9 en el sistema decimal. El valor de cada uno de estos símbolos depende, como es lógico, de su posición, que se calcula mediante potencias de base 16.

Calculemos, a modo de ejemplo, el valor del número hexadecimal 1A3F16:

$1A3F_{16} = 1*16^3 + A*16^2 + 3*16^1 + F*16^0$

$1*4096 + 10*256 + 3*16 + 15*1 = 6719$

$1A3F_{16} = 6719_{10}$

Ejercicio 7:

Expresa en el sistema decimal las siguientes cifras hexadecimales: 2BC516, 10016, 1FF16

Ensayemos, utilizando la técnica habitual de divisiones sucesivas, la conversión de un número decimal a hexadecimal. Por ejemplo, para convertir a hexadecimal del número 173510 será necesario hacer las siguientes divisiones:

1735 : 16 = 108 Resto: 7

108 : 16 = 6 Resto: C es decir, 12_{10}

6 : 16 = 0 Resto: 6

De ahí que, tomando los restos en orden inverso, resolvemos el número en hexadecimal:

$1735_{10} = 6C7_{16}$

Ejercicio 8:

Convierte al sistema hexadecimal los siguientes números decimales: 3519_{10}, 1024_{10}, 4095_{10}

Conversión de números binarios a hexadecimales y viceversa

Del mismo modo que hallamos la correspondencia entre números octales y binarios, podemos establecer una equivalencia directa entre cada dígito hexadecimal y cuatro dígitos binarios, como se ve en la siguiente tabla:

DECIMAL	BINARIO	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A

11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Las operaciones más frecuentes que tendremos que realizar serán conversiones decimal-binario y viceversa y hexadecimal-binario y viceversa.

Unidades de medida

Los sistemas informáticos son capaces de procesar cada vez más información (miles de millones de bits) en menos tiempo (décimas de segundo). Para describir tales unidades de información empleamos los múltiplos.

Es usual hacer una distinción dependiendo de si estamos trabajando con unidades de almacenamiento o con velocidades de transferencia.

Para las unidades de almacenamiento se emplean los múltiplos de *byte*, un byte es la cantidad mínima de información que se suele encontrar en cualquier sistema informático y es una secuencia de 8 *bits* (dos dígitos hexadecimales).

Para las velocidades de transferencia en una red de datos digital se emplean múltiplos de bit.

Múltiplos del byte[3]

Los prefijos utilizados para los múltiplos del byte normalmente son los mismos que los prefijos del Sistema Internacional (SI), también se utilizan los prefijos binarios, pero existen diferencias entre ellos, ya que según el tipo de prefijo utilizado los bytes resultantes tienen valores diferentes.

Esto se debe a que los prefijos del SI se basan en base 10 (sistema decimal), y los prefijos binarios se basan en base 2 (sistema binario), por ejemplo:

kibibyte = 1024 B = 2^{10} bytes.

kilobyte = 1000 B = 10^3 bytes.

[3] https://es.wikipedia.org/wiki/Byte

Múltiplos utilizando los prefijos del Sistema Internacional

Prefijo	Símbolo del prefijo	Nombre resultante del prefijo + *byte*	Símbolo del múltiplo del *byte*	Factor y valor en el SI
Valor de referencia		byte	B	$10^0 = 1$
kilo	k	kilobyte	kB	$10^3 = 1\,000$
mega	M	megabyte	MB	$10^6 = 1\,000\,000$
giga	G	gigabyte	GB	$10^9 = 1\,000\,000\,000$
tera	T	terabyte	TB	$10^{12} = 1\,000\,000\,000\,000$
peta	P	petabyte	PB	$10^{15} = 1\,000\,000\,000\,000\,000$
exa	E	exabyte	EB	$10^{18} = 1\,000\,000\,000\,000\,000\,000$
zetta	Z	zettabyte	ZB	$10^{21} = 1\,000\,000\,000\,000\,000\,000\,000$
yotta	Y	yottabyte	YB	$10^{24} = 1\,000\,000\,000\,000\,000\,000\,000\,000$

Múltiplos utilizando los prefijos ISO/IEC 80000-13

Actualmente los prefijos binarios al igual que el byte forman parte de la norma ISO/IEC 80000-131

Los primeros prefijos desde kibi a exbi fueron definidos por la Comisión Electrotécnica Internacional (IEC) en diciembre de 1998, e incluidos en el IEC 60027-2 (Desde febrero del año 1999), posteriormente en el año 2005 se incluyeron zebi y yobi.

Prefijo	Símbolo del prefijo	Nombre resultante del prefijo + *byte*	Símbolo del múltiplo del *byte*	Factor y valor en el ISO/IEC 80000-13
Valor de referencia		byte	B	$2^0 = 1$
kibi	Ki	kibibyte	KiB	$2^{10} = 1024$
mebi	Mi	mebibyte	MiB	$2^{20} = 1\,048\,576$
gibi	Gi	gibibyte	GiB	$2^{30} = 1\,073\,741\,824$
tebi	Ti	tebibyte	TiB	$2^{40} = 1\,099\,511\,627\,776$
pebi	Pi	pebibyte	PiB	$2^{50} = 1\,125\,899\,906\,842\,624$
exbi	Ei	exbibyte	EiB	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
zebi	Zi	zebibyte	ZiB	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$
yobi	Yi	yobibyte	YiB	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$

En la práctica, se suelen emplear KB, MB, GB, etc. como múltiplos en base 10 para las capacidades de almacenamiento comerciales (por ejemplo al comprar unidades de almacenamiento) mientras que los sistemas operativos emplean la misma notación KB, MB, GB, etc. como múltiplos en base 2.

Ejercicio: Calcula la capacidad de almacenamiento expresado según potencias de base 2 de un ~~disco duro que se vende como con capacidad de 2TB~~.

Múltiplos de bit[4]

[4] https://es.wikipedia.org/wiki/Kilobit

La definición decimal se utiliza comúnmente para expresar las velocidades de transmisión de datos. Por ejemplo, la velocidad de transmisión de un módem de 56K (56 Kbit/s) sería 56 000 bits por segundo (56 x 1000).

Kilobit (Kbit) Unidad informática de medida de información. 1 Kilobit = 1000 bits.

Múltiplos de bits					
Prefijo del SI (SI)			**Prefijo binario (IEC 60027-2)**		
Nombre	**Símbolo**	**Múltiplo**	**Nombre**	**Símbolo**	**Múltiplo**
Kilobit	kbit	10^3	Kibibit	Kibit	2^{10}
Megabit	Mbit	10^6	Mebibit	Mibit	2^{20}
Gigabit	Gbit	10^9	Gibibit	Gibit	2^{30}
Terabit	Tbit	10^{12}	Tebibit	Tibit	2^{40}
Petabit	Pbit	10^{15}	Pebibit	Pibit	2^{50}
Exabit	Ebit	10^{18}	Exbibit	Eibit	2^{60}
Zettabit	Zbit	10^{21}	Zebibit	Zibit	2^{70}
Yottabit	Ybit	10^{24}	Yobibit	Yibit	2^{80}

Ejercicio: Calcula cuánto tiempo se tardaría en transferir un archivo de 650MiB a través de una conexión de red de 50 Mbit/s (Suponiendo que no se emplea ninguna cantidad de datos para el control de la transmisión)

¿Cómo se almacena una imagen (sin compresión, mapas de bits) de manera digital?

¿Cómo se almacena el sonido (sin compresión, PCM, WAV) de manera digital?

¿Cómo se almacena el texto (sin formato, UTF8, iso-8859-1) de manera digital?

Arquitectura de Von Neumann

La arquitectura von Neumann, también conocida como modelo de von Neumann o arquitectura Princeton, es una arquitectura de computadoras basada en la descrita en 1945 por el matemático y físico John von Neumann y otros, en el primer borrador de un informe sobre el EDVAC. Este describe una arquitectura de diseño para un computador digital electrónico con partes que constan de una unidad de procesamiento que contiene una unidad aritmético lógica y registros del procesador, una unidad de control que contiene un registro de instrucciones y un contador de programa, una memoria para almacenar tanto datos como instrucciones, almacenamiento masivo externo, y mecanismos de entrada y salida.[5]

[5] https://es.wikipedia.org/wiki/Arquitectura_de_von_Neumann

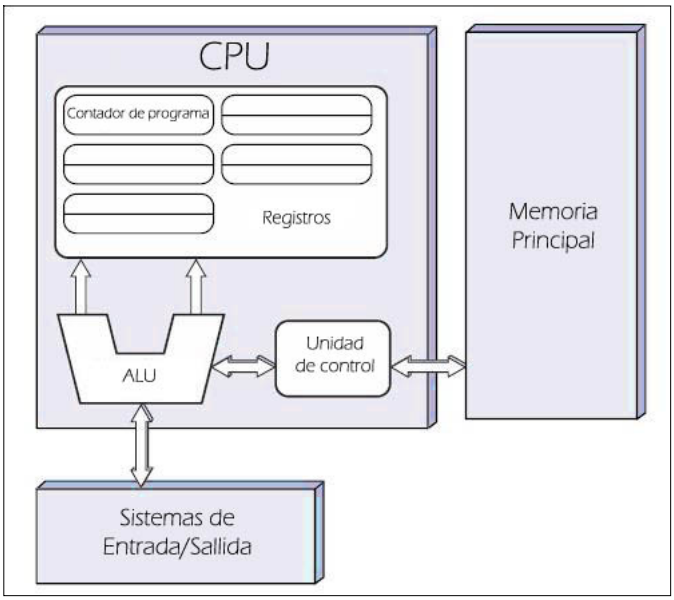

Ilustración 1. Arquitectura de Von Neumann

La importancia de ésta arquitectura radica en que tanto las instrucciones como los datos se encuentra en la memoria principal, que actualmente conocemos como memoria RAM.

La unidad de control se encarga de interpretar las instrucciones, y la ALU (Unidad Aritmético-Lógica) se encarga de procesarlas.

La ALU sólo es capaz de realizar operaciones matemáticas sencillas (sumas y restas básicamente) y operaciones lógicas (AND, OR, XOR, NOT).

Hoy en día las computadoras siguen un modelo muy similar a la arquitectura de Von Neumann, aunque sean capaces de realizar operaciones astronómicas en unos instantes, todo se reduce a las operaciones mencionadas anteriormente.

Ejercicio: Busca información sobre las operaciones lógicas con dígitos binarios. Comprueba cómo es posible realizar una operación simple, como sumar dos bits, mediante operaciones lógicas.

Simulador PIPPIN6

El simulador PIPPIN traduce un programa escrito en el lenguaje ensamblador PIPPIN y simula su ejecución, mostrando cómo se ejecuta el programa en la máquina. El objetivo perseguido con esta actividad es ofrecer al alumno una visión del funcionamiento real (aunque muy simplificado) y de más bajo nivel de un ordenador.

[6] http://www.dsie.upct.es/docencia/pippin/

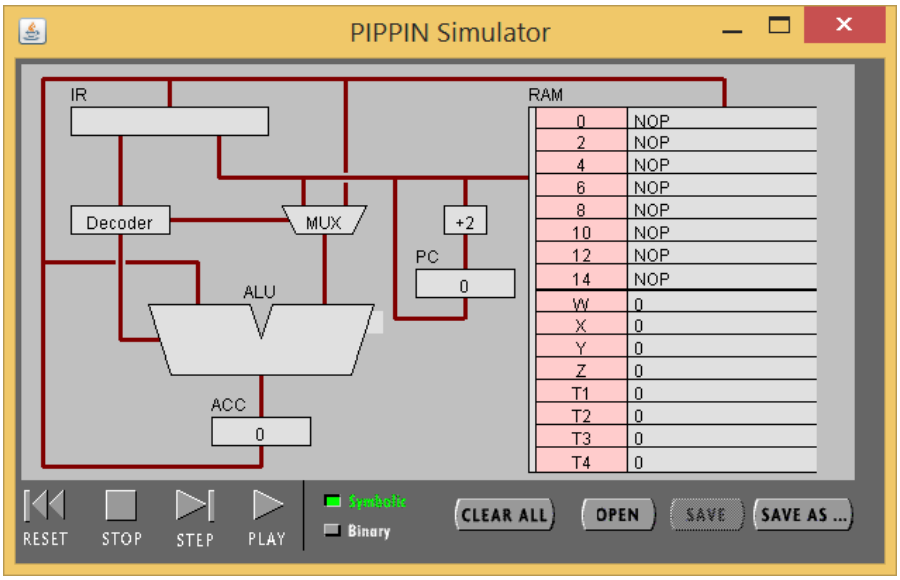

Ilustración 2. Simulador de CPU PIPPIN

Ejemplo: se quiere resolver la ecuación W = X * (2 + Y). Pruebe a introducir el siguiente programa en la parte superior derecha (posiciones RAM-0 a RAM-10), y tras esto fije los valores requeridos en la zona de registro (W a T4), por ejemplo ponga en X el valor 5 y en Y 8. Posteriormente pulse el botón Play para comenzar la simulación.

```
LOD #2
ADD Y
MUL X
STO W
HLT
```

Si selecciona el pulsador Binary podrá ver la representación de las instrucciones y el código en binario, tal y como lo ejecutaría un ordenador. Puede cambiar también directamente el contenido de los registros de la CPU como el Acc (acumulador) y el PC (contador de programa). Eso le puede ayudar a depurar antes sus programas. También puede introducir valores negativos en los registros y luego visualizarlos en binario. Haga unos programas que comprueben que PIPPIN sólo puede trabajar con números enteros comprendidos entre -128 y + 127 ... ¿por qué?

Lenguaje Ensamblador "PIPPIN"

OPCODE	SIGNIFICADO
`LOD R / LOD #X`	Cargar el valor del registro R (o el valor entero X) al Acumulador
`STO R`	Almacenar el valor del Acumulador en el registro R
`NOP`	Ninguna operación, avanza a la siguiente posición de memoria (perdiendo el tiempo en hacer nada)
`HLT`	Parar la ejecución del simulador PIPPIN
`ADD R / ADD #X`	Suma el contenido del Acumulador con el contenido del registro R (o el valor entero X) y sobrescribe el Acumulador
`SUB R / SUB #X`	Resta el contenido del Acumulador con el contenido del registro R (o el valor entero X) y sobrescribe el Acumulador

`MUL R / MUL #X`	Multiplica el contenido del Acumulador por el contenido del registro R (o el valor entero X) y sobrescribe el Acumulador
`DIV R / DIV #X`	Divide el Acumulador por el contenido del registro R (o el valor entero X) y sobrescribe el Acumulador
`NOT`	Si el contenido del Acumulador es 0, lo pone a 1. En caso contrario pone un 0
`JMP P`	Salta a la posición de memoria P
`JMZ P`	Si el valor del Acumulador es 0, salta a la posición de memoria P. En caso contrario, ejecuta la siguiente instrucción
`CPZ R/ CPZ #X`	Si el contenido del registro R (o el valor entero X) es cero, pone el Acumulador a 1. En caso contrario pone un 0
`CPL R/ CPL #X`	Si el contenido del registro R (o el valor entero X) es menor que cero, pone el Acumulador a 1. En caso contrario pone un 0