# Códigos Binarios

Dr. Roberto Gómez Cárdenas

10 de marzo de 1999

## Introducción

La historia de la codificación de lenguajes humanos es muy antigua y no ha dejado de evolucionar desde su inicio. La aparición de las computadoras la hizo bastante más compleja. En un principio los lenguajes eran hablados y después fueron escritos en papel basandose en figuras. Una persona reconocía estas figuras y era capaz de reproducir los sonidos y darle sentido al mensaje (o no era capaz y el mensaje se convertia en todo un misterio).

Estas figuras fueron evolucionando y se convirtieron en caracteres, que formaban un alfabeto. La combinación de algunos de estos caracteres expresaba una idea o un mensaje. Con la llegada de las computadoras fue necesario convertir dichos caracteres en códigos de tal forma que pudieran ser almacenados en la memoria de la computadora y poco después reproducidos.

En otras palabras, la información representada en un alfabeto de entrada debía ser transformada (codificada) a un alfabeto de salida que entendiera la computadora. La codificación debe realizarse de manera que a cada elemento del primer alfabeto le corresponda un elemento distinto del segundo (ver figura 1).

| | | | | |
|---|---|---|---|---|
| INFORMACION<br>(Alfabeto de<br>entrada ) | $\Longrightarrow$ | Sistema<br>de<br>codificación | $\Longrightarrow$ | INFORMACION CODIFICADA<br>(alfabeto de<br>salida) |

Figura 1: *Esquema de codificación*

Todo sistema de codificación lleva consigio un código que se define como la ley de correspondencia biunívoca entre los datos que se van a representar y su codificación.

Entre las características más importantes de un código esta el número de caracteres que se pueden codificar, el número de bits que se utilizan para codificar un carácter (longitud del código). El número máximo del conjunto de caracteres que es posible representar es de $2^{\text{longitud}}$.

Las primeras computadoras estructuraron su memoria en varios trozos (ahora casi universalmente son llamados *bytes*) y cada uno de estos trozos almacenaba el código que representaba un carácter. El tamaño de estos trozos variaba de acuerdo a las diferentes computadoras que existían. Esto traía como consecuencia que el número de caracteres representables variaban de una computadora a otra, así como el hecho de que dos computadoras diferentes podían tener diferentes códigos para el mismo carácter.

Estas dos últimas circunstancias provocaron que el intercambio de información entre computadoras fuera una tarea difícil, por lo que se tuvieron que definir varios estándares.

En lo primero que se tuvieron que poner de acuerdo fue en el tamaño del "trozo" que almacenada el código del carácter. En un principio este variaba entre 4 y 6 bits. Poco después se adoptó el tamaño de 7 bits, para que en los 80's quedara en 8 bist.

Hoy en día la unidad básica de información computacional es el bit, y el término *pablabra* se refiere al conjunto de $n$ bits que conforman un carácter en un determinado código, y es de 8 bits. La tabla de la figura 2 presenta un resumen de las principales formas de representación de información binaria.

- Tipos Representación
  - Alfanuméricas
    - FIELDATAEBCDIC
    - ASCII
    - Gray
    - Reflejado
  - Decimales
    - BCD puro o natural
    - BCD exceso a tres
    - BCD 2421...
    - Biquinario
  - Numéricas
    - Números enteros
      - Punto fijo sin signo (enteros-positivos)
      - Punto fijo con signo (signo-magnitud)
      - Punto fijo com complementos a dos
    - Números Reales
      - Punto flotante (entera y fraccionaria)
      - Punto flotante con mantisa normalizada
      - Punto flotante exceso-128
      - .........
  - Redundantes (detección errores)
    - Cíclicos
    - Baudot
    - Paridad
      - Vertical o a nivel carácter
      - Horizontal o a nivel bloque
      - Entrelazada
    - Código de Hamming

Figura 2: *Diferentes tipos de representación binaria*

Es importante hacer la diferencia entre la representación decimal de un numero binario, y la codificación binaria de un número decimal. En efecto las representaciones binarias de un número pueden (y en la mayor parte de los casos lo hacen) llegar a diferir de su código en binario. Las representaciones y conversiones en diferentes bases de un número decimal ya se han visto anteriormente. El presente capítulo esta dedicado a diferentes códigos utilisados para la representación de caracteres o números; tanto para la codificación de información no computable, como para aquella sobre la cual se pueden aplicar las operaciones aritméticas básicas.

En este documento se tratan las dos primeras formas de representración binaria: la alfanumérica y la numérica. Se describen cuatro códigos para la representación de caracteres alfanuméricos y otros tantos códigos numéricos.

# 1 El código FIELDATA

Este código fue desarrollado por la Armada de los Estados Unidos como un código militar de comunicación de caracteres y fue espeficicado en la norma militar MIL-STD-118. Han aparecido tres revisiones del estándar MIL-STD-188 desde 1958 hasta 1969. FIELDATA fue definido como un código de 7 bits de 128 caracteres, donde los primeros 64 caracteres estan reservados para códigos de control (los cuales varian en las diferentes versiones del estándar). UNIVAC usa la segunda mitad del código FIELDATA el cual contiene todos los caracteres gráficos y códigos simples de control de teletipo.

Para poder apreciar la evolución de dicho código describiremos los diferentes cambios que se fueron dando en éste. En un principio los códigos de las tarjetas perforadas correspondían, en los casos donde FIELDATA y el teclado de la perforadora de tarjetas (keypunch) se traslapaban, al keypunch de la IBM 026. La siguiente keypunch IBM 029 asignaba diferentes patrones de hoyos a algunas marcas de puntuación, consistentes con el código EBCDIC (ver sección2) del sistema 360 de IBM. Muchos sistemas UNIVAC fueron modificacos más tarde para aceptar los patrones de hoyos de la IBM 029 como opción, convirtiendolos en códigos para los correspondientes caracteres FIELDATA.

El código define un carácter de retorno, (CR, LF, etc) en el MIL-STD-188 pero asigno gráficos en la variante UNIVAC de FIELDATA los cuales variaban de acuerdo al dispositivo al que eran conectados. Las consolas de la UNIVAC 1107 y 1108 incorporaron la impresora Teletype 35, la cual utilizaba los códigos CR y LF para el propósito para el que fueron pensados, e imprimía un signo de paro total para el código STOP (057 octal). En la terminal de consola de la UNIVAC 1108 (Type 4009, basada en una UNISCOPE 300) había discrepancias en los símbolos del teclado, lo que aparecía en el monitor y la salida de la impresora PAGEWRITER. Algunas impresoras de línea UNIVAC terminaban la salida de una línea apenas encontraban el primer carácter IDLE (077 en octal) mientras que otras. incluyendo la impresora/lectora de tarjetas UNIVAC 1004, impriía dicho carácter como un signo de desigualdad $\neq$ e imprimia los subsecuentes caracteres en una línea. Hay que imanganerse lo “gracioso” de estas discrepancias cuando se movían programas de una máquina a otra, o cuando un sitio instalaba una nueva pieza de hardware.

En la especificación original de la Computer Sciences Corporation 1107 Monitor System, el cual fue eventualmente adoptado por la UNIVAC como EXEC II, el “Master Space” de FIELDATA (octal 0) fue usado en los streams de entrada para distinguir comandos y tarjetas de datos. En sus manuales, CSC y después UNIVAC usaron el carácter $\nabla$, una letra griega delta invertida, para representar el “Master Space”. Notando la semejanza entre este carácter y la Delta impresa por el carácter del código 4 de la UNIVAC 1004 algunos sitios, incluyendo el Case Institute of Technology and Chi Corporation, intercambiaron los códigos de impresión dando como resultado que el código de CPU 00 se imprimió como $\triangle$ y el código 04 como `@` .

En los 70's UNIVAC adoptó otro código como estándar, el código ASCII (ver sección 3 y acopló la mayor parte de su software a ese código de ocho bits. Aún años después si se miraba con atención se podían encontrar pequeños bits de FIELDATA viajando en el moderno sistema 1100.

La tabla de la figura 3 presenta los códigos de los caracteres más importantes del código FIELDATA.

Para poder encontrar el código que corresponde a un carácter es necesario conocer el valor de los bits que componen su código. Los bits de la parte superior de la tabla representan los bits 5,4 y 3 del código, mientras que los bits de la parte izquierda son los bits 2, 1 y 0. A continuación se presentan algunos ejemplos:

- El código del carácter `#` es 000 011 = 3
- El código del carácter `G` es 001 100 = 12

| **210 ⋱ 543** | **000** | **001** | **010** | **011** | **100** | **101** | **110** | **111** |
|---|---|---|---|---|---|---|---|---|
| **000** | @ | $C$ | $K$ | $S$ | ) | $*$ | 0 | 8 |
| **001** | [ | $D$ | $L$ | $T$ | $-$ | ( | 1 | 9 |
| **010** | ] | $E$ | $M$ | $U$ | $+$ | % | 2 | , |
| **011** | # | $F$ | $N$ | $V$ | $<$ | : | 3 | ; |
| **100** | $\wedge$ | $G$ | $O$ | $W$ | $=$ | ? | 4 | / |
| **101** | SP | $H$ | $P$ | $X$ | $>$ | ! | 5 | . |
| **110** | $A$ | $I$ | $Q$ | $Y$ | & | $'$ | 6 | ó |
| **111** | $B$ | $J$ | $R$ | $Z$ | $ | \ | 7 | _ |

Figura 3: *Código Fieldata de 6 bits*

- El código del carácter `Z` es 011 111 = 31
- El código del carácter `?` es 101 100 = 44
- El código del carácter `5` es 110 101 = 53
- El código del carácter `/` es 111 100 = 60

# 2 El código EBCDIC

En los 60's avances significativos en la codificación de caracteres para la comunicación resultaron en el desarrollo de caracteres de 8 bits. En 1962 IBM decide lanzar una serie de computadoras de diferente potencial que serían compatibles entre ellas y su tecnología de base, sin ser la más avanzada, marcaría la pauta en el plano comercial. La serie 360 impone varias normas, entre las que podemos mencionar la organización de memoria, la adopción de la unidad más chica posible en memoria (el carácter) y dejar atras la palabra, la normalización del sistema operativo de la máquina y el hecho de que un programa que funcionaba en alguna de las máquinas de la serie 360 podía funcionar sobre cualquier otra máquina de la familia (siempre y cuando el tamaño de la memoria lo permitiera).

Otras de las innovaciones de la serie 360 fue la creación cró y promoción de un estándar de codificación conocido como Extended Binary-Coded-Decimal Interchange Code, o EBCDIC para abreviar. Hasta ese momento se utilizaban solo 6 bits para representar el conjunto de caracteres usados (ver sección 1), lo cual proporcionaban 64 posibles caracteres. Sin embargo 64 caracteres no eran suficientes para almacenar la información de la vida común y corriente. Para aplicaciones ordinarias el conjunto de simbolos usados en un servicio de contabilidad, por ejemplo, se necesitan al menos 120 signos.

Este sistema de codificación definía caracteres de 8 bits, definiendo hasta 256 caracteres, (cuatro veces más que con seis bits). El inconveniente que presentaba era el aumento en el tamaño de la memoria necesaria (en promedio de un tercio), pero esto no dura mucho, debido a la baja permanente del precio de las memorias.

Mientras que el mundo hubiera estado mejor con un código de solo ocho bits, otro estándar llamado American Standard Code for Information Interchange (ASCII) fue adoptado en 1963 y al final ganó la batalla de los estándares.

El códgio EBCDIC es un código de 8 bits con 256 caracteres sin bit de paradidad, muy utilizado en arquitecturas IBM. La tabla 4 presenta una muestra de lo que sería el código de algunos caracteres. Los cuatro bits de la izquierda corresponden a los bits 3,2,1 y 0 de la palabara y los bits de la parte de arriba a los bits 7,6,5,y 4. Por ejemplo el código EBCDIC del carácter $a$ sería 00011000).

| bits | 0000 | 1000 | 1001 | 1010 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|
| **0000** | *nul* | | | | { | } | | 0 |
| **0001** | *soh* | *a* | *j* | | *A* | *J* | | 1 |
| **0010** | *stx* | *b* | *k* | *s* | *B* | *K* | *S* | 2 |
| **0011** | *etx* | *c* | *l* | *t* | *C* | *L* | *T* | 3 |
| **0100** | *pf* | *d* | *m* | *u* | *D* | *M* | *U* | 4 |
| **0101** | *ht* | *e* | *n* | *v* | *E* | *N* | *V* | 5 |
| **0110** | *lc* | *f* | *o* | *w* | *F* | *O* | *W* | 6 |
| **0111** | *del* | *g* | *p* | *x* | *G* | *P* | *X* | 7 |
| **1000** | | *h* | *q* | *y* | *H* | *Q* | *Y* | 8 |
| **1001** | *rlf* | *i* | *r* | *z* | *I* | *R* | *Z* | 9 |
| **1010** | *smm* | | | | | | | |
| **1011** | *vt* | | | | | | | |

Figura 4: *Tabla Código EBCDIC*

El incoveniente del código es el aumento del tamaño de memoria necesaria (en promedio de un tercio); pero a la larga esto no representó ningún problema debido a la baja pemanente del precio de las memorias.

El código EBCDIC anticipa por poco una nueva norma que se estaba "poniendo a punto" en los EUA para la transferencia de información en las telecomunicaciones: el código ASCII (ver sección 3), la cual preveía la normalización de un código a siete cifras bancarias al cual IBM podía adaptarse.

Aunque el código ya se considera obsoleto IBM lo sigue utilizando en algunos mainframes por motivos de compatibilidad.

# 3 Código ASCCII (American Standard Code for Information Interchange)

Es el código más utilizado hoy en día. Utiliza siete bits que permite $2^7$ o 128 posibles combinacione de 1'si y 0's e igual número de caracteres. Estos 128 caracteres se dividen en 94 que pueden imprimirse y 34 no imprimibles. Entre los que pueden imprimirse encontramos:

- 26 letras mayusculas, (A ... Z)
- 26 letras minusculas, (a ... z)
- 10 dígitos numericos (0 ... 9), y
- 32 caracteres especiales, (por ejemplo: `%, #, &, @` ,etc), todos imprimibles a excepción del espacio y del borreado (delete)

Los 34 caracteres no imprimibles son utilizados como funciones de control para direccionar datos y formar el texto impreso en un formato preestablecido. Se dividen en:

- *afectadores de formatos*, controlan la distribución de la impresión; incluye controles parecidos a los de una máquina de escribir común y corriente.
- *separadores de información*, sirven para separar los datos en divisiones como párrafos y páginas, (p.e. RS y FS)

- *control de comunicación*, utiles en la transmisión de texto en terminales remotas; por ejemplo STX y ETX usuados para enmarcar un mensaje texto cuando se transmite a través de algún medio de comunicación.

En la tabla 5 se encuentran definidos los códigos de los diferentes caractes ASCII. Para encontrar el código de un determinado carácter debe juntar los bits correspondientes. Los de la izquierda corresponden a los bits 3,2,1 y 0 (en ese orden) y los de arriba a los bits 6,5 y 4. Una vez la palabra binaria formada basta con encontrar su equivalente en base diez. Algunos ejemplos son:

- El carácter `A` es $100\ 0001_2 = 65_{10}$
- El carácter `4` es $011\ 0100_2 = 52_{10}$
- El carácter `j` es $110\ 1010_2 = 106_{10}$
- El carácter `=` es $011\ 1101_2 = 61_{10}$
- El carácter CR (retorno de carro) es $000\ 1101_2 = 52_{10}$

| **3210 ⋱ 654** | **000** | **001** | **010** | **011** | **100** | **101** | **110** | **111** |
|---|---|---|---|---|---|---|---|---|
| **0000** | NUL | DLE | SP | 0 | @ | *P* | ‘ | *p* |
| **0001** | SOH | DC1 | ! | 1 | *A* | *Q* | *a* | *q* |
| **0010** | STX | DC2 | ” | 2 | *B* | *R* | *b* | *r* |
| **0011** | ETX | DC3 | # | 3 | *C* | *S* | *c* | *s* |
| **0100** | EOT | DC4 | $ | 4 | *D* | *T* | *d* | *t* |
| **0101** | ENQ | NAK | % | 5 | *E* | *U* | *e* | *u* |
| **0110** | ACK | SYN | & | 6 | *F* | *V* | *f* | *v* |
| **0111** | BEL | ETB | ′ | 7 | *G* | *W* | *g* | *w* |
| **1000** | BS | CAN | ( | 8 | *H* | *X* | *h* | *x* |
| **1001** | HT | EM | ) | 9 | *I* | *Y* | *i* | *y* |
| **1010** | LF | SUB | ∗ | : | *J* | *Z* | *j* | *z* |
| **1011** | VT | ESC | + | ; | *K* | [ | *k* | { |
| **1100** | FF | FS | , | < | *L* | \ | *l* | \| |
| **1101** | CR | GS | − | = | *M* | ] | *m* | } |
| **1110** | SO | RS | . | > | *N* | ∧ | *n* | ∼ |
| **1111** | SI | US | / | ? | *O* | _ | *o* | DEL |

Figura 5: *Código Ascii de 7 bits*

A continuación se presentan algunas notas sobre los más comunes caracteres de control.

- BEL: cuando la salida es una terminal, se supone que BEL emite un “beep” audible.
- BS: este es el carácter oficial del backspace del código ASCII. Muchos sistemas operativos (como Unix) o editores de texto (como emacs) utilizan DEL para esta función, pero esto no es estándar.
- CAN: *Cancel* en inglés, o `<ctrl-X>`, originalmente fue pensado para señalar que el usuario deseaba cancelar toda una línea de entradas. La mayoría de los sistemas Unix, utilizan `<ctrl-U> (NAK)` para este propósito.

- CR: *Carry Return,* retorno de carro. Historicamente, los primeros teletipos requerían dos caracteres al final de cada línea: LF (linefeed) que movía el cilindro para moverse a la siguiente línea, mientras que CR regresaba el mecanismo de impresión al principio de la línea. Los sistemas MS-DOS aún utilizan ambos caracteres para terminar una línea. Sin embargo los sistemas Unix esperan que la tecla CR (usalmente llamada `return` o `enter` del teclado es usada para terminar una línea, mientras que solo un carácter de fin de línea (linefeed) es almacenado al final de cada línea cuando el texto es guardado en archivos.
- DEL: *Delete.* Utilizado para borrar un carácter. Ahora bien se recomienda leer la parte de BS para una discusión de la variantes en la designación del carácter de backspace.
- ESC: *Escape.* Esta tecla es utilizada en una variedad de funciones, pero originalmente fue usada como prefijo para modificar el significado del carácter siguiente.
- FF: *Form Feed.* Usualmente obliga a las impresora a pasar a la siguiente hoja.
- HT: *Horizontal Tab.* En la mayor parte de los teclados, es tan solo una tecla al lado de la tecla `Q`, y no tiene un uso tan frecuente.
- LF: *Line Feed.* Mueve el cursor a la siguiente línea. En CR se puede leer una discusión acerca de los terminadores de línea.
- NUL: También conocido como <Control-@> o <Control-shift-2>. Tiene un código de cero, y es usado en C para delimitar cadenas de caracteres.

## 3.1 ASCII extendido (ocho bits)

Cuando la comunidad computacional decidió adoptar el tamaño de un byte o palabra como de ocho bits, al código ASCII le sobró un bit. Dicho bit tuvó diferentes usos, para proporcionar detección de error o como:

1. *algunas impresoras* reconocen los caracteres ASCII de ocho bits cuando el bit más significativo se desactiva en 0. Los 128 caracteres adicionales de 8 bits con el bit más significativo activado en 1 se utilizan para otros simbolos como el alfabeto griego o una fuente de tipo cursivo.
2. *comunicaciones* cuando se utiliza en la comunicación de datos, puede emplearse el octavo bit para indicar la paridad del carácter del código binario.

Otra aplicación se dió en la PC de IBM. Cuando la PC de IBM fue desarrollado, la tarjeta de video contenía un byte por cada carácter en el display de caracteres de 80x25. ¿Qué hacer con el bit extra por carácter? ¿Por qué no inventar 128 nuevos caracteres, para dibujos de líneas y símbolos especiales? El resultado fue, por supuesto. el conjunto de caracteres de ASCII extendido para la IBM PC. La figura de abajo (figura 6) muestra una parte de los caracteres generados por el monitor en la IBM PC original.

## 3.2 Estandarización de ASCII

ASCII fue definido por primera vez por la American National Standards Institute (ANSI) en el estándar ANSI X3.4 en 1968. El código ASCII también se describe en la ISO 636 (1973) y en el CCITT V.2, la cual hace referencia al código como estándar IA5 (International Alphabet #5). El código ASCII es un código de 7 bits, dando como resultado un máximo de 128 caracteres. Sin embargo, el estándar ANSI X3.16 (1976) y el estándar CCITT V.4 describen el uso de un octavo bit

| Dec | Hex | Binary | Symbol | Dec | Hex | Binary | Symbol |
|---|---|---|---|---|---|---|---|
| 182 | B6 | 10110110 | ╢ | 219 | DB | 11011011 | █ |
| 183 | B7 | 10110111 | ╖ | 220 | DC | 11011100 | ▄ |
| 184 | B8 | 10111000 | ╕ | 221 | DD | 11011101 | ▌ |
| 185 | B9 | 10111001 | ╣ | 222 | DE | 11011110 | ▐ |
| 186 | BA | 10111010 | ║ | 223 | DF | 11011111 | ▀ |
| 187 | BB | 10111011 | ╗ | 224 | E0 | 11100000 | α |
| 188 | BC | 10111100 | ╝ | 225 | E1 | 11100001 | ß |
| 189 | BD | 10111101 | ╜ | 226 | E2 | 11100010 | Γ |
| 190 | BE | 10111110 | ╛ | 227 | E3 | 11100011 | π |
| 191 | BF | 10111111 | ┐ | 228 | E4 | 11100100 | Σ |
| 192 | C0 | 11000000 | └ | 229 | E5 | 11100101 | σ |
| 193 | C1 | 11000001 | ┴ | 230 | E6 | 11100110 | µ |
| 194 | C2 | 11000010 | ┬ | 231 | E7 | 11100111 | τ |
| 195 | C3 | 11000011 | ├ | 232 | E8 | 11101000 | Φ |
| 196 | C4 | 11000100 | ─ | 233 | E9 | 11101001 | Θ |
| 197 | C5 | 11000101 | ┼ | 234 | EA | 11101010 | Ω |
| 198 | C6 | 11000110 | ╞ | 235 | EB | 11101011 | δ |
| 199 | C7 | 11000111 | ╟ | 236 | EC | 11101100 | ∞ |
| 200 | C8 | 11001000 | ╚ | 237 | ED | 11101101 | ø |
| 201 | C9 | 11001001 | ╔ | 238 | EE | 11101110 | ε |
| 202 | CA | 11001010 | ╩ | 239 | EF | 11101111 | ∩ |
| 203 | CB | 11001011 | ╦ | 240 | F0 | 11110000 | ≡ |
| 204 | CC | 11001100 | ╠ | 241 | F1 | 11110001 | ± |
| 205 | CD | 11001101 | ═ | 242 | F2 | 11110010 | ≥ |
| 206 | CE | 11001110 | ╬ | 243 | F3 | 11110011 | ≤ |
| 207 | CF | 11001111 | ╧ | 244 | F4 | 11110100 | ⌠ |
| 208 | D0 | 11010000 | ╨ | 245 | F5 | 11110101 | ⌡ |
| 209 | D1 | 11010001 | ╤ | 246 | F6 | 11110110 | ÷ |
| 210 | D2 | 11010010 | ╥ | 247 | F7 | 11110111 | ≈ |
| 211 | D3 | 11010011 | ╙ | 248 | F8 | 11111000 | ° |
| 212 | D4 | 11010100 | ╘ | 249 | F9 | 11111001 | ∙ |
| 213 | D5 | 11010101 | ╒ | 250 | FA | 11111010 | · |
| 214 | D6 | 11010110 | ╓ | 251 | FB | 11111011 | √ |
| 215 | D7 | 11010111 | ╫ | 252 | FC | 11111100 | ⁿ |
| 216 | D8 | 11011000 | ╪ | 253 | FD | 11111101 | ² |
| 217 | D9 | 11011001 | ┘ | 254 | FE | 11111110 | ■ |
| 218 | DA | 11011010 | ┌ | 255 | FF | 11111111 | |

Figura 6: *Algunos caracteres del monitor de la IBM PC orignal*

adicional como un bit de chequeo de paridad. Este bit toma su valor de tal fomar que la suma de 7 bits descritos que representan un carácter sea par o non. De esta forma solo es posible detectar un error en un bit durante la transmisión de caracteres. La paridad par es recomendada para comunicaciones asincronas, mientas que la paridad non para sistemas sincronos.

En realidad los aplicaciones pueden implementar paridad par, paridad non, no paridad, paridad MARK o paridad SPACE. Los problemas de asignación de paridad son la causa de molestias para cualquiera que trabaje con diferentes BBSs. Aún en la cultura Internet de hoy en día, uno debe de estar informado de la correcta configuración para transmisiones de caracteres de 7 u 8 bits dentro del protocolo FTP (File Transfer Protocol).

Una extensión adicional del código utilizando el carácter `ESCAPE` fue definida en la norma ANSI X3.64-1979 como un esfuerzo de estandarizar representaciones de caracteres gráficos y control de cursor. De hecho, el código ASCII del sistema operativo DOS proporciona un conjunto de 256 caracteres, y los caracteres gráficos de ANSI tienen una representación dentro de los 128 caracteres extra que el sistema DOS permite.

La transmisión serial de ASCII es definida como ANSI X3.15 - X3.16 y CCITT V.4 and X.4. Un bit de inicio y uno de paso son añadidos al carácter para "delinear" el carácter para una transmisión asincrona, como en Código Baudot. Sin embargo, la transmisión sincrona de caracteres ASCII también es definida.

### 3.3 Las sonrisas ASCII

En el mundo del correo electrónico muchas veces es necesario expresar algún tipoo de comentario dentro del texto que se envia. Con un poco de imaginación y con la combinación de ciertos caracteres es posible realizar lo anterior.

En efecto, la combinación de ciertos caracteres da a conocer el sentido de la frase: broma, felicidad, tristeza, enojo o conformismo. Una lista completa de estas combinaciones se puede encontrar en `http://www.astro.umd.edu/~marshall/smileys.html`.

A continuación se listan algunas combinaciones de caracteres y su significado:

| | |
|---|---|
| :> | sonriendo, cara feliz, no me tomes muy en serio |
| B-) | carita feliz de alguien portando lentes |
| 8-( | sonrisa no feliz, triste |
| :-( | triste o cara decepcionada |
| :-o | Oh, nooooo, Wow, sorpresa |
| :-P | carita mostrando la lengua |
| :-# | mensaje que contiene algo que no debe de ser difundido |
| :ˆ) | de una persona con una nariz muy grande |
| &:-) | persona con cabello rizado |
| @:-) | persona con cabello ondulado |
| #:-) | persona con cabello enredado |
| O:-) | un angel |
| C=:-) | jefe, el usuario es un jefe |
| :-—K- | mensaje formal |
| ...—... | SOS |
| :-. | el usuario no tiene comentarios |
| ∼:o | usuario es un bebe |
| ¡3 | un corazón |
| 8— | macho |
| (0) | hembra |
| 8—(0) | teniendo sexo |
| }87(= | vampiro |

Figura 7: *Ejemplos sonrisas ASCII*

## 4 Código UNICODE

La propagación del uso de la computadora a nivel mundial y el desarrollo de Internet provoca que las computadoras deban de ser capaces de reconocer y manipular textos en diferentes lenguajes. El manejar diferentes lenguajes implica diferentes letras, pero no siempre es el caso. En el caso de lenguajes europeos el lenguaje latino, con la adición de diatricos, sirve para la mayor parte de ellos.

Sin embargo la mayor parte de las computadoras son utilizadas en un lenguaje, por lo que un conjunto de conjuntos de caracteres fue desarrollado por MS-DOS para cada uno de los lenguajes aprovechando el octavo bit no usado de ASCII. Esto fue hecho por Europa dejando los 128 caracteres bajos iguales a los del alfabeto inglés (conjunto ASCII) y modificando los 128 caracteres superiores.

Esto esta bien para aplicaciones que utilisan un solo lenguaje y corren en la misma máquina; sin embargo si es necesario enviar información entre computadoras se presenta un problema cuando el mismo código representa diferentes caracteres en diferentes conjuntos de caracteres.

El problema se intensifica en librerías y otros sistemas de bases de datos donde es necesario mezclar textos de diferentes lenguajes, representar bien el texo e intercalar varios textos sin provocar confusiones.

Otro problema se presenta en los lenguajes ideográficos del este-asiatico donde el número de caracteres es bastante grande (el chino tiene entre 7,000 y 13,000 caracteres básicos). Es lógico ver que no se pueden representar todos estos caracteres con 128 códigos. Por lo que es necesario otro metodo para codificar, el cual debe de resolver los siguientes problemas:

- codificación incompatible
- insuficiencia de espacio para codificar

El problema más serio era que el estándar de ocho bits no proveía el espacio suficiente para tener suficientes códigos para representar todos los caracteres. Fue por esto que el concepto de páginas de codificación fue inventado y que surgieron diferentes estándares de codificación.

La solución fue simple, ampliar el espacio de codificación.

Los conjuntos de caracteres CJK (Chino, Japones y Coreano) necesitan una gran cantidad de espacio, por lo que inventaron códigos de 16 bits para su codificación. Desafortunadamente, a pesar de que los ideogramas chinos son parte de todas las formas usadas por los chinos, japoneses y coreanos, se inventaron cuanto diferentes codificaciones. Esto significa que la transferencia de texto solo podía hacerse vía busqueda-masiva en tablas para convertir la codificación de caracteres extranjeros a locales antes de que el sistema despliega el mensaje.

El código Universal surgió como solución a los problemas anteriores. Este código es un superconjunto del conjunto de caracteres ASCII que utiliza dos bytes por cada carácter en lugar de uno. Puede manejar hasta 65,536 combinaciones de caracteres en lugar de solo 256. Puede adaptarse a la mayoría de los idiomas hablados en el mundo.

UNICODE es el resultado del trabajo en equipo de varias compañías y organizaciones de estandares, así como de intereses bibliográficos. Actualmente es mantenida y desarrollada por el UNICODE Consortium, que se encuentra en California USA (`http://www.unicode.org`).

Al mismo tiempo del desarrollo de UNICODE un comite técnico de la ISO[1] se enfocó a resolver el mismo problema llegando a conclusiones muy similares. En realidad las dos soluciones son oficalmente identicas, tal y como UNICODE es (y será) identico al BMP (Basic Multilingual Plabe) de la ISO 1046. En realidad BMP es un espacio de codificación[2] de 16 bits definido por la ISO 1046 (conocido como ISO 1046-2) el cual permite codificación de 32 bits (ISO 10646-4) que proporciona un espacio de codificación que parece excesivo para lo que se requiere hoy en día.

Para efectos prácticos UNICODE e ISO 10646 son idénticos.

Unicode es un sistema de 16 bits para información en código y caracteres de todos los idiomas del mundo. Esta diseñado para permitir que los sistemas computacionales intercambien información sin ambiguedad ya que a cada carácter le corresponde un solo código. Con los 16 bits se pueden formar 65,536 caracteres en código. Los caracteres de 16 bits son llamados *caracteres ampliados*. Los primeros 128 códigos del Unicode son idénticos a los del ASCII. Solo añade otro byte cero a cada

[1]ISO es una organización internacional que establece ciertos estandares
[2]Espacio codificación=número de códigos posibles

carácter ASCII para convertirlo a Unicode. Unicode contiene más de 20,000 caracteres básicos, que son usados para representar todas las palabras o conceptos en chino, japonés o coreano.

La tabla 8 presenta la representación en unicode de los caracteres numéricos latinos:

| **Decimal** | **ASCII** | **Unicode** |
|---|---|---|
| 0 | 0110000 | 0000000000110000 |
| 1 | 0110001 | 0000000000110001 |
| 2 | 0110010 | 0000000000110010 |
| 3 | 0110011 | 0000000000110011 |
| 4 | 0110100 | 0000000000110100 |
| 5 | 0110101 | 0000000000110101 |
| 6 | 0110110 | 0000000000110110 |
| 7 | 0110111 | 0000000000110111 |
| 8 | 0111000 | 0000000000111000 |
| 9 | 0111001 | 0000000000111001 |

Figura 8: *Tabla representación dígitos en Unicode*

Es bueno remarcar que UNICODE no es nada de lo siguiente:

- una secuencia de ordenamiento de caracteres
- una definición glyph[3]
- un mecanismo de comparación de cadenas de caracteres
- un mecanismo de formateo de texto
- una definición de lenguaje
- un mecanismo de conversión de caracteres

Cada una de estas funciones es necesaria para el manejo de documentos de texto y existen estándares definidos para cada uno de ellos. En muchos casos (como en codificación de caracteres) existen varios estándares y estos deben de ser reconciliados.

Actualmente UNICODE solo es usado parcialmente en sistemas comerciales. Ha sido utilizado en el desarrollo de diferentes versiones del sistema operativo Unix así como el de Microsoft Windows NT. Dentro de este se encuentra la representación básica de todos los caracteres, aún aquellos que solo son usados en un sólo lenguaje y por lo tanto representan un único conjunto de caracteres.

Se dice que UNICODE soporta 34,168 diferentes caracteres codificados, derivados de 24 diferentes alfabetos de distintos lenguajes.

La tendencia es facilitar el desarrollo de aplicaciones multi-lenguajes dentro de los sistemas operativos. Sin embargo aún es necesario escribir la aplicación para reconocer y manipular los diferentes conjuntos de caracteres que ya estan disponibles. Como ejemplo del esfuerzo necesario para contar con estas aplicaciones, basta con saber que sólo un sistema de librería que soporta codificación de UNICODE esta disponible.

Los siguientes problemas no estan resueltos por UNICODE o por cualquier otro estándar de codificación:

[3]dentro de lo que es tecnología de la información un glyph es un simbolo gráfico que proporciona la apariencia o forma de un carácter. Puede ser un font alfabético o numérico que dibuja un carácter numérico.

- UNICODE solo resuelve los problemas triviales de los caracteres mundiales
  - *caracteres locales*; algunos símbolos, como chinos y koreanos, permiten al usuario inventar nuevos caracteres (lo equivalente a palabras en un diccionario de lenguaje latino) para condiciones locales y necesidades futuras.
  - *lenguajes faltantes*; si los símbolos del lenguaje que se quiere no estan incluídas entonces UNICODE no sirve de nada a menos que se utilizen páginas privadas.
- UNICODE es un esquema de codificación universal. Datos en sistemas locales deben convertirse a este código y esto es un proceso pesado, tanto en la actualización de las tablas como en la ejecución de la conversión
- Desplegado e impresión. Muchos sistemas operativos populares (MS-DOS, WINDOWS 3.1, etc) no reconocen UNICODE por lo que conversiones especiales deben ser realizadas o se debe de programar dispositivos especiales para el desplegado e impresión de caracteres
- Entrada. UNICODE no es muy amigable para las entradas. Esto se dificulta si se toma en cuenta que los usuarios esperan poder utilizar todos los caracteres disponibles en un sistema UNICODE, pero esto no es posible sin antes seguir un entrenamiento.
- Ordenamiento. En el alfabeto latino nadie discute que el carácter `b` antecede al carácter `c`; sin embargo esto no es tan evidente en otros lenguajes y/o alfabetos. UNICODE no indica nada a este respecto. Esto depende en el lenguaje y de la naturaleza de los caracteres.
- Localización. UNICODE solo transmite el contenido del texto plano, para que un documento puede entenderse mejor debe de existir información de formateo, de contenido antes de que el sistema receptor pueda darle sentido al documento.

# 5 Código BCD

Las siglas del código significan, *Binary Coded Decimal*, pero también es conocido como Packet-Decimal Representation. Es utilizado para representar los dígitos decimales del 0 al 9 y es un ejemplo de lo que se conoce como un código *pesado*. Esto es, cada posición de bit en el código tiene un valor numérico fijo o un peso asociado con él. El dígito representado por una palabra del código se puede encontrar sumando los bits de pesos. El código BCD utiliza cuatro bits por dígito, con los pesos elegidos de tal forma que coinciden con los de un entero binario de 4 bits. Este código también se conoce como 8-4-2-1 debido a los pesos utilizados.

La tabla 9 presenta la representación de los nueve dígitos decimales en BCD:

Como ejemplo de representación consideremos los números 314159 y 271823. La representación en BCD de estos números es:

```
            3    1    4    1    5    9
314159 = 0011 0001 0100 0001 0101 1001
            2    7    1    8    2    3
271823 = 0010 0111 0001 1000 0010 0011
```

Entre las desventajas del código es que desperdicia memoria ya que solo utiliza 10 de las 16 posibles combinaciones de bits posibles. (ver tabla figura 9).

Existen seis combinaciones de bits que son inválidas ya que no represntran ninguno de los nueve dígitos. Estas son: $1010, 1011, 1100, 1101, 1110$ y $1111$. Otra desventaja es que los cálculos aritméticos se dificultan cuando se considera el signo.

| Dígito | Código BCD |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0111 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |
| **combinaciones** | 1010 |
| | 1011 |
| **de** | 1100 |
| **bits** | 1101 |
| | 1110 |
| **no empleadas** | 1111 |

Figura 9: *Tabla Código BCD 8421*

Para sumar dos números es necesario sumar la representación binaria de cada uno de los dígitos. En caso de que se produzca una combinación inválida se debe de sumar seis (0110) a dicha combinación. Si se produce un carry este se suma al dígito de la izquierda, en caso de que no se cuente con dicho dígito se considera como 0000.

Tomemos como ejemplo la suma de los números que se representaron anteriormente:

```
   0011 0001 0100 0001 0101 1001 (314159)
+  0010 0111 0001 1000 0010 0011 (271823)
-----------------------------------------
   0101 1000 0101 1001 0111 1100
```

Los números `1001` y `1100` son combinaciones inválidas, por lo que hay que sumar `0110` a cada uno:

```
   0101 1000 0101 1001 0111 1100
+                 0110      0110
--------------------------------
   0101 1000 0101 1001 1000 0010 (585982)
```

# 6 Código BCD exceso a 3

Es un código relacionado con el BCD, ya que es obtenido añadiendo 3 a cada dígito decimal y convirtiendo el reusltado de la sumas en un número binario de cuatro bits. Ya que no se asignan pesos definitivos a las cuatro posiciones de dígitos, el exceso a 3 es un código sin peso, (no ponderado).

Tomemos como ejemplo los decimales 2 y 9, su código en exceso a 3 sería

$2 + 3 = 5 \rightarrow 0101$
$9 + 3 = 12 \rightarrow 1100$

Para números de más de un dígito se tendría que tratar bit por bit. Tomemos como ejemplo el número 473.

```
   4     7     3
  +3    +3    +3
 ---   ---   ---
   7    10     6
0111  1010  0110

473 = 011 1010 0110
```

La tabla 10 presenta la representación de los diez dígitos en código BCD y en código exceso-3. Como puede constatarse también hay cominaciones no utilizadas. En exceso tres estas combinaciones son: $0000, 0001, 0010, 1101, 1110$ y $1111$.

| Dígito Decimal | Código BCD | Código Exceso-3 |
|---|---|---|
| 0 | 0000 | 0011 |
| 1 | 0001 | 0100 |
| 2 | 0010 | 0101 |
| 3 | 0011 | 0110 |
| 4 | 0100 | 0111 |
| 5 | 0101 | 1000 |
| 6 | 0110 | 1001 |
| 7 | 0111 | 1010 |
| 8 | 1000 | 1011 |
| 9 | 1001 | 1100 |
| **combinaciones** | 1010 | 1101 |
| | 1011 | 1110 |
| **de** | 1100 | 1111 |
| **bits** | 1101 | 0000 |
| | 1110 | 0001 |
| **no empleadas** | 1111 | 0010 |

Figura 10: *Tabla Código BCD y exceso-3*

# 7 Código 2421 o Aiken

En general, los códigos binarios requieren un mínimo de 4 bits por dígito. El código 2421 se puede formular disponiendo 4 bits en 10 posibles combinaciones de las 16 que se pueden disponer con 4 bits; las seis combinaciones que no se usan deben evitarse, ya que no tienen significado alguno.

Este código es tanto ponderado como autocomplementario, es un código autocomplementario porque tiene la propiedad de que el complemento a 9 de un número decimal se obtiene directamente cambiando unos por ceros y ceros por unos en el código. Por ejemplo el decimal 395 se representa en el código 2421 por: 0011 1111 1011. El complemento 9 de 395 es 604, el cual se representa por 1100 0000 0100 que se obtiene por el reemplazo de los 1's por 0's y los 0's por 1's.

Esta propiedad es muy útil cuando se hacen las operaciones internamente con números decimales (en código binario) y la sustracción se hace por medio del complemento a nueve.

El código 2421 es un código ponderado ya que a cada posición de bit se le asigna un factor de ponderación en tal forma que cada dígito se pueda evaluar sumando los valores ponderados de todos los unos de la combinación codificada. Por ejemplo, la combinaciόm de bits 1101, cuando se pondera con los dígitos respectivos 2421, da el equivalente decimal de $(2*1)+(4*1)+(2*0)+(1*1)=7$

Cabe señalar que algunos dígitos pueden codificarse en dos formas posibles en este código. Al 4 decimal se le pueden asignar las combinaciones de bits 0100 ó 1010, ya que ambas se suman para hacer un valor ponderado total de 4.

| **Código 2421** | **Dígito Decimal** |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 2 |
| 1001 | 3 |
| 1010 | 4 |
| 1011 | 5 |
| 1100 | 6 |
| 1101 | 7 |
| 1110 | 8 |
| 1111 | 9 |

Figura 11: *Tabla Código 2421*

# 8 Código Gray

El código Gray forma parte de los códigos que se conocen con el nombre de códigos cíclicos y tiene la característica de que el código para dos números consecutivos difiere tan solo en un bit. Esto es, la distancia entre dos palabras de código[4] es de un bit. No se considera un código numérico, ni ponderado y puede tener cualquier número de bits.

Su aplicación se da en sensores con cambios no graduales. Una de las aplicaciones de este código se presenta cuando los datos análogos se presentan mediante un cambio continuo de una posición relevante. La parte relevante se divide en segmentos y a cada segmento se le asigna un número. Si se hace que segmentos adyacentes correspondan a números en código Gray adyacentes, se reduce la ambiguedad cuando la posición relevante está en la línea que separa dos segmentos cualquiera.

Un ejemplo de código Gray a tres bits se presenta en la tabla de la figura 12.

Cuando los útimos bits de la mitad de los números del código gray coinciden con los últimos bits de la otra mitad se dice que el código es *reflejado*.

## 8.1 Convirtiendo de binario a gray

Representando los diez dígitos decimales con un código Gray usando cuatro bits, se tiene otra forma de código BCD, (ver sección 8.3). El código Gray puede extenderse a cualquier número de bits y la conversión entre código binario y Gray es útil en algunos casos.

[4]En general la *distancia* entre dos palabras binarias es igual al número de bits en que difieren las dos palabras

| Gray a 3 bits | Gray a 4 bits |
|---|---|
| 000 | 0000 |
| 001 | 0001 |
| 011 | 0011 |
| 010 | 0010 |
| 110 | 0110 |
| 111 | 0111 |
| 101 | 0101 |
| 100 | 0100 |
| | 1100 |
| | 1101 |
| | 1111 |
| | 1110 |
| | 1010 |
| | 1011 |
| | 1001 |
| | 1000 |

Figura 12: *Ejemplo código gray reflejado con 3 y 4 bits*

Para la conversión de binario a gray se deben aplicar la siguientes reglas:

- el bit más significativo en el código Gray es el mismo que el correspondiente bit más significativo en el número binario
- de izquierda a derecha, se debe sumar cada par de bits del código binario para obtener el siguiente bit del código Gray. En caso de que la suma produzca un carry este se descarta.

Consideremos como ejemplo el convertir el número `100111` a código Gray:

El primer paso es asignar el valor del bit más significativo.

$$\begin{array}{lllllll} 1 & 0 & 0 & 1 & 1 & 1 & \text{binario} \\ \downarrow & & & & & & \\ 1 & & & & & & \text{Gray} \end{array}$$

El segundo paso es realizar las sumas de los diferentes bits:

$$\begin{array}{lllllllllllll} 1 & + & 0 & + & 0 & + & 1 & + & 1 & + & 1 & \text{binario} \\ & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 1 & & 1 & & 0 & & 1 & & 0 & & 0 & \text{Gray} \end{array}$$

Es decir `100111 = 110100`

## 8.2 Convirtiendo gray a binario

Se utiliza un metodo similar al de la conversión de binario a Gray, pero con algunas diferencias. En este caso se aplican las siguientes reglas:

- el bit más significativo en el número binario es el mismo que el correspondiente bit más significativo del código Gray

- cada bit del código binario es producido sumando cada bit binario generado en el paso anterior al siguiente bit adyacente del código Gray. Como en el caso anterior si la suma produce un carry este se descarta.

Tomemos como ejemplo la conversión del número `110100` El primer paso es el mismo que el de la conversión anterior:

$$\begin{array}{ccccccl} 1 & 1 & 0 & 1 & 0 & 0 & \text{Gray} \\ \downarrow & & & & & & \\ 1 & & & & & & \text{binario} \end{array}$$

El segundo paso consiste en *suma cruzadas*:

El segundo paso es:

$$\begin{array}{ccccccccccccccccccccccl}
1 & & & & 1 & & & & 0 & & & & 1 & & & & 0 & & & & 0 & \text{Gray} \\
 & & & \nearrow & & & & \nearrow & & & & \nearrow & & & & \nearrow & & & & \nearrow & & \\
 & & + & & \downarrow & & + & & \downarrow & & + & & \downarrow & & + & & \downarrow & & + & & \downarrow & \\
 & \nearrow & & & & \nearrow & & & & \nearrow & & & & \nearrow & & & & \nearrow & & & & \\
1 & & & & 0 & & & & 0 & & & & 1 & & & & 1 & & & & 1 & \text{binario}
\end{array}$$

Por lo que `110100 = 100111`

## 8.3 Definiendo un código para los números decimales del 0 al 15

Se necesitan cuatro bits para representar los números 0-15. Es posible construir un código asignando al bit *i* del código el valor de 0 si el bit *i* e *i+1* del número binario correspondiente son iguales y 1 en caso contrario. El bit más significativo del número debe de compararse con cero cuando se utliza esta técnica. El código resultante se muestra en la figura 13.

| **Decimal** | **Binario** | **Gray** |
|---|---|---|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0011 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0111 |
| 6 | 0110 | 0101 |
| 7 | 0111 | 0100 |
| 8 | 1000 | 1100 |
| 9 | 1001 | 1101 |
| 10 | 1010 | 1111 |
| 11 | 1011 | 1110 |
| 12 | 1100 | 1010 |
| 13 | 1101 | 1011 |
| 14 | 1110 | 1001 |
| 15 | 1111 | 1000 |

Figura 13: *Tabla binario-Gray*

# 9 Código Biquinario

El código biquinario es muy interesante, *Bi* significa dos y *quinario* significa cinco, de tal forma que el código se orienta a contener siete bits con sólo dos bits 1 en cada grupo de código. La ponderación de código se indica por los valores posicionales, por lo que también se le denomina código 5043210. Es un código de 7 dígitos con propiedades de detección de error. Cada dígito decimal cuenta con cinco ceros y dos unos colocados en las correspondientes columnas de carga.

Pertenece a la familia de los códigos ponderados. En este código a cada posición de bit se le asigna un factor de ponderación en tal forma que cada dígito se pueda evaluar suamando los valores ponderados de todos los unos de la combinación codificada. Por ejemplo, la combinación de bits 0101000, cuando se pondera con los dígitos respectivos 5043210, da el equivalente decimal:
$(0 * 5) + (1 * 0) + (0 * 4) + (1 * 3) + (0 * 2) + (0 * 0) = 3$.

La propiedad de la detección de error se debe al hecho de que se tienen exactamente dos bits 1 en el bit codificado. Cualquier aumento o disminución indica un error. El código del cero también contiene dos bits 1. Además de que sólo debe haber dos bits 1, uno de ellos debe de estan en alguna de las dos posiciones a la izquierda y el otro en cualquiera de las cinco posiciones a la derecha del número. Durante la transmisión de señales de un lugar a otro puede presentarse un error. Uno o más bits pueden cambiar de valor. Un circuito en el lado recepción puede detectar la presencia de más (o menos) dos bits con valor uno, y en el caso de que la combinación de bits no coincida con la combinación permitida, se detectará un error.

La tabla de la figura 14 presenta la representación de los nueve dígitos decimales.

| Dígito Decimal | Biquinario |
|---|---|
| 0 | 0100001 |
| 1 | 0100010 |
| 2 | 0100100 |
| 3 | 0101000 |
| 4 | 0110000 |
| 5 | 1000001 |
| 6 | 1000010 |
| 7 | 1000100 |
| 8 | 1001000 |
| 9 | 1010000 |

Figura 14: *Tabla del código Biquinario (5043210)*

Como ejemplo se presenta la representación del número decimal 285 en código biquinario:

$285_{10} = 0100010\ 1001000\ 1000001$