ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

TESIS DE GRADO PREVIA A LA OBTENCION DEL TITULO

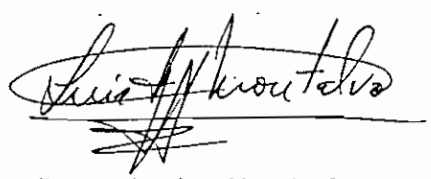
DE INGENIERO EN ELECTRONICA Y TELECOMUNICACIONES

ESTUDIO TEORICO-PRACTICO DE LOS CODIGOS DE DETECCION DE ERRORES POR CHEQUEO REDUNDANTE CICLICO (CRC)

ALICIA MEDINA VALLEJO

1.991

Certifico que la presente Tesis de Grado ha sido realizada en su totalidad por la señorita R. Alicia Medina V.



Ing. Luis Montalvo
DIRECTOR DE TESIS

INDICE

OBJETIVOS Y ALCANCE DE LA TESIS

Existen algunos códigos para el control de errores en transmisión de datos, siendo el CRC uno de ellos; éstos se describen en forma general en el capítulo uno, con el fin de conocer algunas de sus características y poder compararlos entre sí.

La presente Tesis tiene como uno de sus objetivos principales realizar un estudio teórico detallado sobre los códigos de detección de errores por chequeo redundante cíclico, de tal manera que permita justificar los algoritmos de cálculo del CRC.

Por tal razón, en el capítulo dos se analizan las propiedades y fundamentos matemáticos de los códigos cíclicos, que son la base para llegar a establecer los procesos de codificación y decodificación de mensajes.

En relación a los algoritmos, se dispone de la aplicación con cierto polinomio; en el capítulo tres se realizan los cambios necesarios para trabajar con el polinomio fijado en la Recomendación V.41 del CCITT. Además se definen tanto las condiciones iniciales como la forma en

que se han de introducir los datos para el cálculo.

Los algoritmos se diferencian en su forma de procesar el mensaje, es decir, se puede tomar la información por bytes o de bit en bit; lo que determina su velocidad. Los diagramas de flujo correspondientes se presentan en el capítulo cuatro.

Para poder observar la detección de errores por medio del CRC se integran todos los pasos de la transmisión de datos, ésto es desde la edición del mensaje hasta cuando es recibido en forma correcta. Lo que permite presentar una aplicación práctica al tema de la Tesis, siendo a la vez una ayuda a la parte teórica, para comprender mejor éstos códigos.

La descripción del programa y su diagrama de flujo se incluyen en el capítulo cuatro.

Es importante realizar una comparación del tiempo que necesitan los diferentes métodos para el cálculo del CRC sobre un mismo mensaje. También se desea conocer el área de memoria que ocupan; para lo que se ha desarrollado el software en base a los algoritmos estudiados, con el que se han hecho las pruebas. Los resultados se presentan en el capítulo cinco.

binarios, señales de sensores, etc. En definitiva son una onda continua o una secuencia de símbolos; si es del primer tipo se requiere un conversor analógico-digital (A/D).

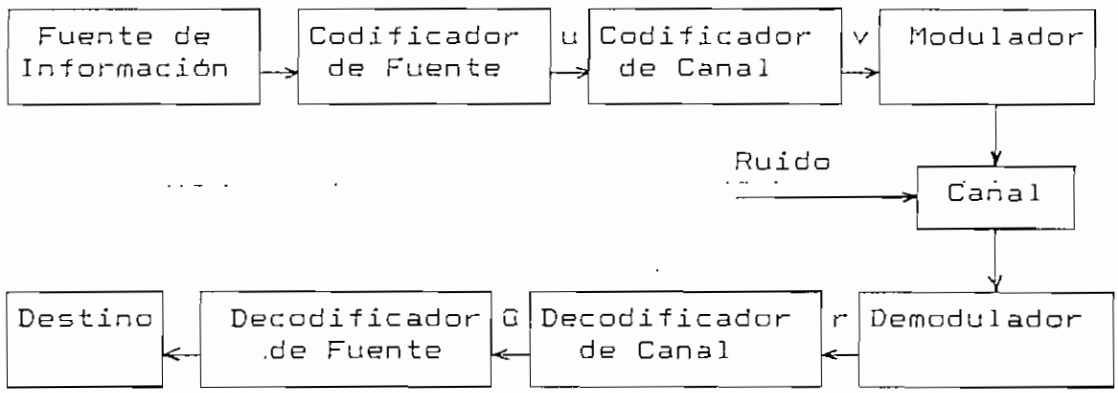


Fig. 1.1 Sistema de Transmisión

El codificador de fuente transforma la información de salida en una secuencia de digitos binarios a la que asignamos u, éste se diseña de manera que el número de bits por unidad de tiempo requerido para representar la fuente de salida sea el mínimo, y que a la vez asegure la reconstrucción de la información sin ambiguedad.

El codificador de canal transforma u en una secuencia discreta v llamada palabra código con la finalidad de combatir el ruido en el medio de transmisión.

El modulador transforma la secuencia binaria en una forma de onda más conveniente para enviarse por el canal fisico, el mismo que puede ser ser una linea telefónica, un enlace de radio de alta frecuencia, un enlace de

comunicación espacial, o un medio de almacenamiento.

En el lado del receptor el demodulador realiza la operación contraria al modulador; su salida es la secuencia codificada recibida **r**.

El decodificador de canal transforma r en una secuencia binaria $\hat{u}$ de nombre secuencia estimada. La estrategia de decodificación se basa en ciertas reglas y en el ruido característico del canal. Idealmente $\hat{u}$ debería ser la replica de u , pero el ruido puede generar errores.

El decodificador de fuente transforma esta secuencia estimada en la señal de salida que se entrega al destinatario.

El diagrama de bloques simplificado del sistema de transmisión se presenta en la fig. 1.2.

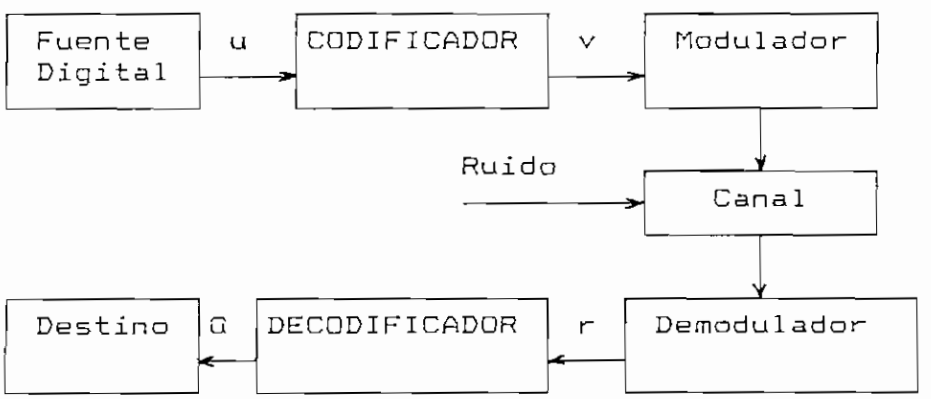


Fig.1.2 Sistema de Transmisión Simplificado

La capacidad de transferencia de un canal, que es el número de bits transmitidos por unidad de tiempo (bits/s),

se define, considerando las siguientes limitaciones: la presencia de ruido y otras interferencias, la sensibilidad del receptor, el número de estados de señalización que depende de la potencia máxima de la señal, etc. A su vez las fuentes tienen una tasa fija, que es el número de bits de información por símbolo transmitido que ingresan al codificador. Si la tasa de información es menor que la capacidad de transferencia es posible codificar de modo que en la decodificación la probabilidad de error sea pequeña.

1.2 Códigos de Detección y Corrección de Errores.

La transmisión de datos procesados a través de un sistema de comunicaciones no es inmune a errores que son causados por la presencia de ruido.

1.2.1 Fuentes de Ruido.

La teoría de la comunicación define el ruido como una perturbación eléctrica indeseable. Dependiendo de la fuente que lo genera se tienen los siguientes tipos:

a) Ruido de Fabricación.

b) Ruido Natural.

c) Ruido de Red.

a) **Ruido de Fabricación.-** Es producido por el mal funcionamiento del equipo. Algunas formas de ruido son generadas por filtros, amplificadores y otros circuitos sensitivos a la frecuencia, de manera que amplifican o

canales con memoria; sobre los cuales inciden los errores en forma diversa.

En un canal de baja memoria, aquel que toma en cuenta el mensaje en el instante actual y no anteriores, el ruido afecta a cada símbolo independientemente.

Por ejemplo en el canal binario simétrico (BSC), de la fig.1.3 cada símbolo tiene una probabilidad p de recibirse incorrectamente, y **1-p** de recibirse correctamente. independiente de otros bits transmitidos.

Los errores ocurren en forma aleatoria en la secuencia. y los canales se denominan de error aleatorio.

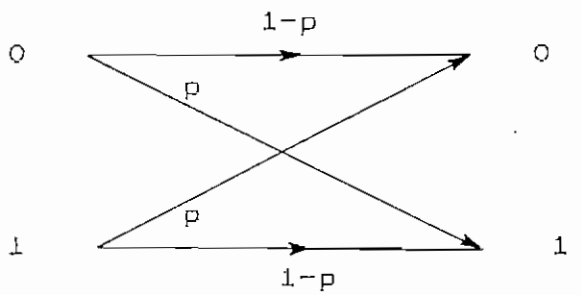


Fig.1.3 Canal Binario Simétrico

Errores aleatorios se producen en transmisión satelital, y principalmente en sistemas con línea de vista.

Sobre canales con memoria, aquellos que consideran el mensaje actual y anteriores, el ruido no es independiente de transmisión a transmisión.

Un modelo de éstos se encuentra en la fig. 1.4.

Hay dos estados, uno "bueno" (s_1) por su baja probabilidad de error $p1 \sim 0$, y otro "malo" (s_2) ya que los errores son altamente probables con $p2 \sim 0.5$. Y las probabilidades q_1 de pasar de s_1 a s_2 y q_2 de s_2 a s_1.

El sistema se encuentra en el estado favorable la mayor parte del tiempo con cambios ocasionales al otro debido a variaciones en sus características de transmisión, lo que produce errores tipo ráfaga, es decir de bits seguidos.

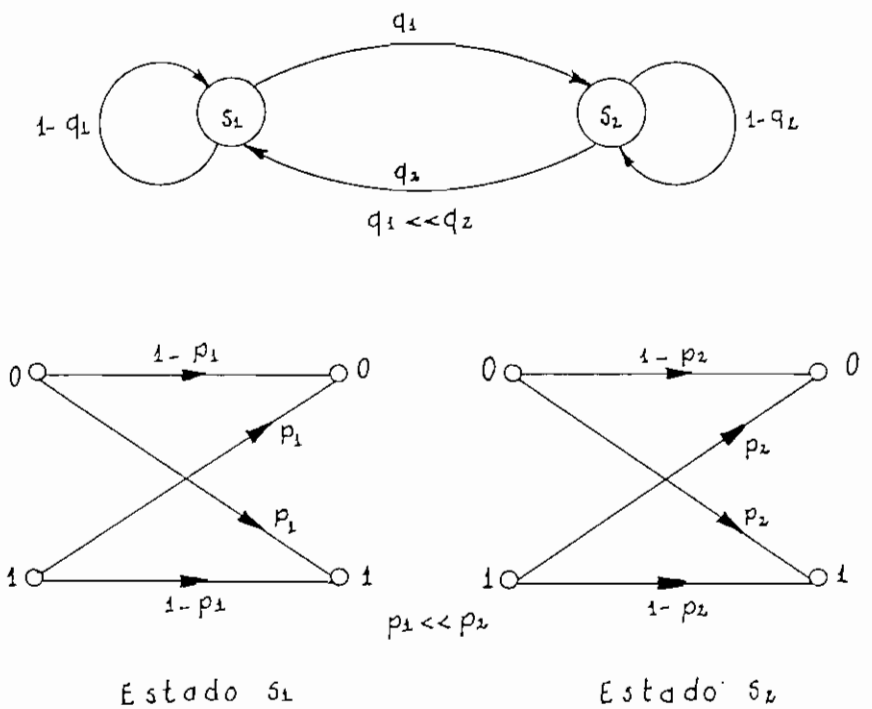


Fig.1.4 Modelo Simplificado de un Canal con Memoria

En los canales de radio los errores tipo ráfaga se generan por desvanecimiento de la señal, en transmisión por cable se deben al ruido impulsivo de conexión y crosstalk.

Algunos canales presentan una combinación de los dos

tipos de errores y se conocen como canales compuestos.

1.2.2.1 Probabilidad de errores aleatorios.- Para errores aleatorios o "ruido blanco" la probabilidad de que no exista error en n posiciones es:

(1.1) $$(1-p)^n$$

La probabilidad de un error simple en n posiciones es:

(1.2) $$np(1-p)^{n-1}$$

La probabilidad de k errores está dada por el k-ésimo término de la expansión binomial:

(1.3)

$$1=[(1-p)+p]^n= (1-p)^n +np(1-p)^{n-1} +\frac{n(n-1)}{2}p^2 (1-p)^{n-2} +...+p^n$$

Por ejemplo, la probabilidad de que ocurran dos errores es:

$$\frac{n(n-1)}{2} p^2 (1-p)^{n-2}$$

Se puede obtener la probabilidad de un número par de errores sumando las dos expansiones binomiales siguientes y dividiendo por 2:

$$1=[(1-p)+p]^n = \sum_{k=0}^{n} C(n,k)\ p^k(1-p)^{n-k}$$

$$[(1-p)-p]^n = \sum_{k=0}^{n} (-1)^k\ C(n,k)\ p^k(1-p)^{n-k}$$

$$(1.4) \qquad \frac{1-(1-2p)^n}{2} = \sum_{m=0}^{[n/2]} C(n,2m)\, p^{2m}(1-p)^{n-2m}$$

La probabilidad de un número impar de errores es igual a la unidad menos el resultado anterior (1.4).

1.2.3 Tipos de Códigos de Detección y Corrección de Errores.

Los códigos para detección y corrección de errores se dividen en dos conjuntos: códigos bloque y códigos convolucionales.

Los códigos bloque a su vez se subdividen de acuerdo al cuadro de la fig.1.5.

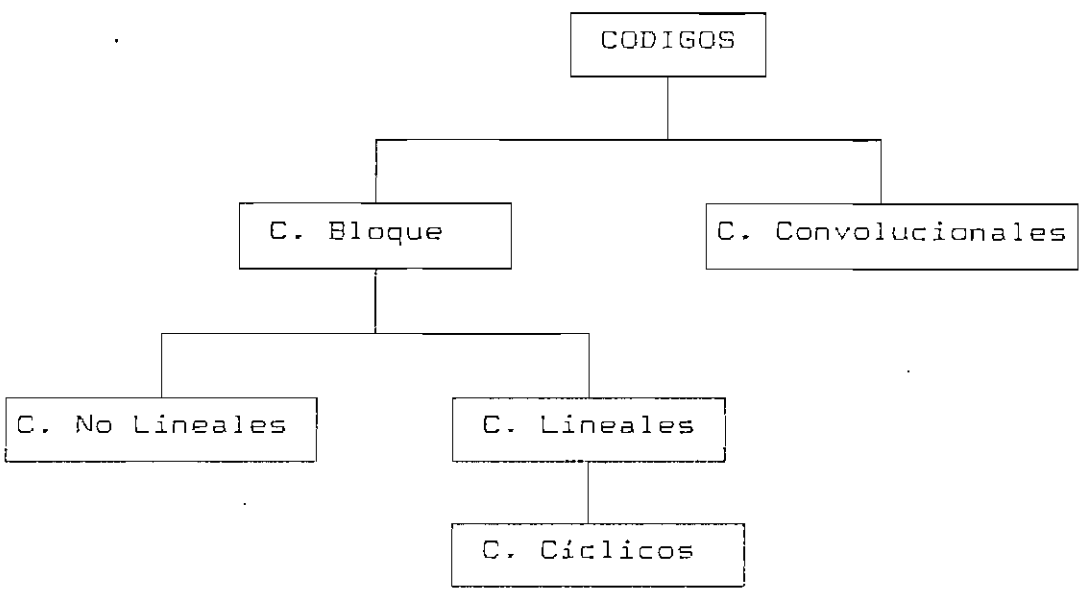


Fig.1.5 División de los Códigos

En un código bloque se divide la secuencia de información en bloques de k bits cada uno, que se denominan

memoria de orden m.

El conjunto de secuencias codificadas producidas por la entrada de k-bits, de salida n-bits y memoria m forman el código convolucional (n,m,k).

R= k/n es la tasa del código. Los bits de redundancia pueden ser sumados a la secuencia de información cuando k<n o R<1. Generalmente k y n son pequeños y la redundancia aumenta incrementando el orden de la memoria.

Debido a que se necesita memoria, el codificador debe ser implementado con un circuito secuencial lógico.

Como una clase importante de los códigos bloque se tiene a los lineales.

Un código bloque es lineal si la aplicación entre mensaje y palabra de código es lineal. Es decir que para cualquier mensaje M_1 y M_2 se cumple:

$$f(M_1+M_2) = f(M_1)+f(M_2)$$

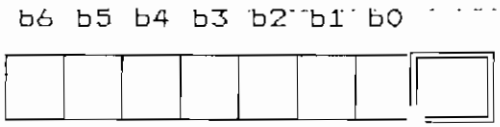
Así, la suma de dos palabras de código es también una palabra de código.

1.2.4 Códigos de Chequeo de Paridad.

Para el control de errores se usan normalmente dos tipos de código de chequeo de paridad: geométricos y cíclicos.

par equivale a emplear aritmética módulo-2, es decir que la suma se obtiene realizando el OR exclusivo de los bits del carácter.

En la fig. 1.6 se presenta un esquema de detección vertical, y un ejemplo de su aplicación.



CARACTER BIT DE PARIDAD

PARIDAD PAR: 0 1 0 1 1 0 1 0

FIG.1.6 Detección Vertical

1.2.4.2 Detección Horizontal (LRC Longitudinal Check).- En este método se utiliza un carácter adicional (de paridad), al final del bloque de información. De esta manera se chequea todos los bits dispersos a través del mensaje, en forma parecida a lo que hace la detección vertical, siendo un mejor instrumento para detectar errores ráfaga que pueden afectar a bits cercanos.

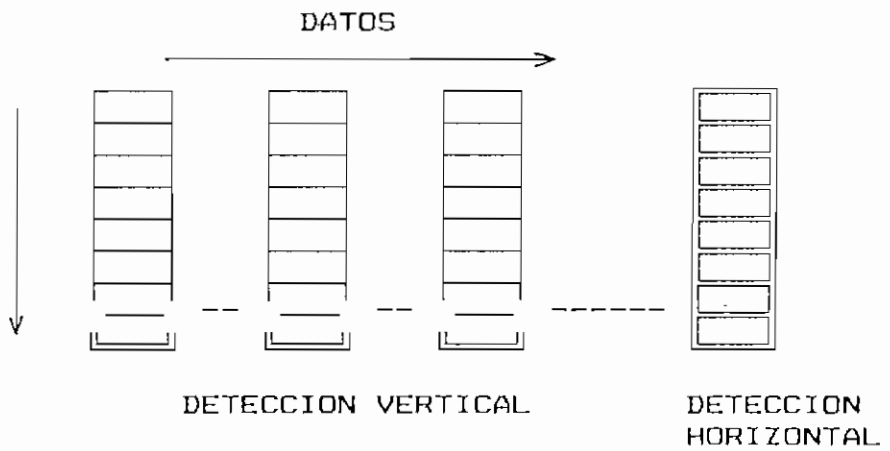
La fig. 1.7 presenta un ejemplo de este tipo de detección.

b_3	0	1	0	1	1	1
b_2	1	0	0	0	1	0 Carácter de
b_1	0	0	0	1	0	1 Paridad
b_0	1	1	0	0	0	0

Fig. 1.7 Detección Horizontal (Paridad Par).

Frecuentemente se utiliza la detección horizontal combinada con la vertical (VRC/LRC), aumentando la capacidad de detección de errores, lo que permite posteriormente aplicar una simple forma de corrección del bit.

En la fig. 1.8 se presenta el esquema de este método y un ejemplo.



b_4	0	1	0	0	1	0	0	1	1
b_3	1	1	0	0	1	1	0	0	0
b_2	1	0	1	1	1	0	1	1	0
b_1	0	1	1	0	0	0	1	1	1 x
b_0	0	0	1	1	1	0	0	1	0
	0	0	1	0	0	1	0	0	
		x							

Fig.1.8 Detección Vertical y Horizontal (P. Par)

Aunque el esquema de paridad horizontal y vertical combinadas es mucho más efectiva que un esquema de paridad simple, no es perfecto porque no puede detectar configuraciones rectangulares de errores.

La fig. 1.9 muestra un posible caso, aunque poco probable en que los errores podrían no ser detectados

utilizando este método. Si los bits resaltados son invertidos por causa de alguna perturbación, los bits de paridad resultan válidos y los errores no son detectados.

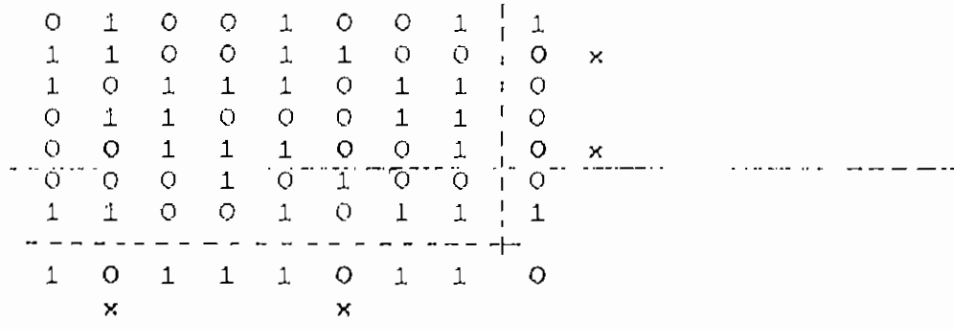


Fig.1.9 Errores no detectados por combinación horizontal-vertical (P. Par).

Códigos Cíclicos (CRC Cyclic Redundancy Check).- Presentan mejores propiedades detectoras que los códigos anteriores y se basan en la división módulo-2 de polinomios.

Los códigos de redundancia cíclica se obtienen por combinación de un código de chequeo de paridad con un código cíclico. Así, los CRCs pueden detectar todos los errores de número impar de bits y todos los modelos de error que detectan los códigos cíclicos.

Un polinomio generador $g_c(X)$ con la siguiente forma describe el código compuesto:

$$g_c(X) = g(X)\ (X+1)$$

donde $g(X)$ es el polinomio generador del código cíclico y $X+1$ representa el chequeo de paridad.

El polinomio del código ciclico es de orden r y coeficientes 0 y 1.

Algunos polinomios g(X) pueden ser utilizados como divisores, sin embargo, tanto el transmisor como el receptor deben tener el mismo.

Las versiones más utilizadas para cálculo del CRC de 16 y 32 bits con el polinomio generador correspondiente se presentan en la siguiente tabla:

Tabla 1.1 Polinomios Generadores

CRC-Ethernet	$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$
CRC-16	$X^{16} + X^{15} + X^{2} + 1$
CCITT (SDLC)	$X^{16} + X^{12} + X^{5} + 1$
CRC-12	$X^{12} + X^{11} + X^{3} + X^{2} + X + 1$
CRC-16 REVERSO	$X^{16} + X^{14} + X + 1$
SDLC REVERSO	$X^{16} + X^{11} + X^{4} + 1$
LRCC-16	$X^{16} + 1$
LRCC-8	$X^{8} + 1$

La trama a transmitir se interoreta como los coeficentes de un polinomio y se divide por g(X); el residuo de la división es la redundancia que se añade en la trama y está constituida por r bits.

En la recepción se divide nuevamente la trama por g(X) y el residuo obtenido se compara con la redundancia añadida; si no coinciden, la trama es necesariamente errónea.

La fig.1.10 presenta en una forma esquemática este método de control de errores.

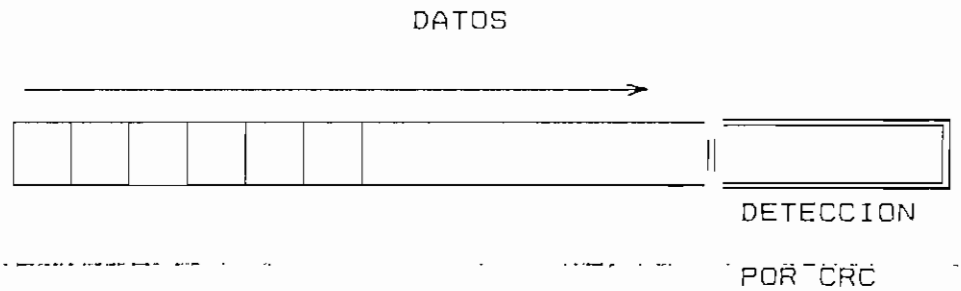


Fig. 1.10 Detección de Errores
a Nivel de Mensaje

1.2.5 Eficiencia de los Códigos de Detección.

Las siguientes eficiencias son típicas de los métodos de detección de errores analizados:

VRC: Con la adición de un bit de chequeo de paridad por carácter detectará aproximadamente 9 de 10 errores.

VRC/LRC: Si en suma a la paridad por carácter se tiene paridad de bloque, 99 de 100 errores serán detectados.

CRC: El CRC-16 y CRC-CCITT tienen las siguientes capacidades de detección de errores:

- Cualquier número impar de errores.
- Todos los errores dobles.
- Todos los errores ráfaga de longitud 15 o menos.
- 99.994 % de errores ráfaga de longitud 16.
- 99.997 % de errores ráfaga de longitud mayor que 16.

1.2.6 Redundancia de los Códigos de Detección.

El número de bits que es necesario añadir para realizar el chequeo de errores difiere según el método como se indica a continuación:

VRC: Requiere un bit por carácter o una redundancia de 1/8=12.5 %.

VRC/LRC: Requiere una paridad de carácter por bloque y un bit por carácter. Si N es el número de caracteres por bloque la redundancia es 1/N +12.5% .

CRC: En el CRC-16 o CRC-CCITT se necesitan dos caracteres por bloque, es decir una redundancia de 2/N.

Tabla 1.2 Redundancia en los Códigos de Detección para diferentes Longitudes de Mensaje

Cód. / N	VRC %	VRC/LRC %	CRC %
8	12.5	25	25
16	12.5	18.75	12.5
32	12.5	15.625	6.25

La tabla anterior contiene la redundancia para longitudes de mensaje diferentes; y se puede observar que mientras aumenta N la redundancia para el CRC se hace más significativamente menor con respecto a los otros métodos.

1.2.7 Desarrollo Práctico de los Códigos.

El tiempo necesario para conocer si se han producido o no errores, depende de cuánto procesamiento requiere el método que se ha utilizado para la detección.

La detección vertical es un método simple de detección y puede implementarse en hardware, de modo que el circuito realice el chequeo de paridad y produzca una interrupción en caso de error. Su desarrollo en software es bajo.

La detección horizontal-vertical también se realiza en hardware, siendo un poco más complejo que el anterior y necesita un cierto tiempo de procesamiento.

Para el CRC se tiene un circuito paralelo que realiza el chequeo en hardware, pero además existen diferentes métodos en software para el mismo fin, los que tratan de mejorarse continuamente.

1.3 Métodos de Corrección de Errores por Retransmisión.

Una vez que los errores han sido detectados utilizando cualquiera de los códigos indicados anteriormente, el siguiente paso es obtener la información correcta.

Los métodos para realizar la corrección se clasifican en tres categorías (Fig. 1.11):

1. Sin control automático de errores.- Se descarta el dato erróneo.

2. Detección automática de errores.- Sin corrección automática.

3. Corrección automática de errores.- Con detección automática.

1.3.1 Sin control Automático de Errores.

En algunos casos es posible pasar por alto los errores que ocurren en la transmisión. Así por ejemplo en un sistema telegráfico punto a punto, si una o dos palabras de un telegrama son alteradas es posible todavía entender el mensaje, ya que el error no es crítico.

En otros casos si el equipo no ha sido diseñado con capacidad para detectar errores, no es posible tomar acciones cuando éstos se presentan.

1.3.2 Detección Automática de Errores.

Cuando existe la facilidad de detectar errores, lo óptimo es disponer de corrección automática, sin embargo existen casos en los que no se utiliza.

Así se tiene:

- Si un error es detectado, y simplemente se descarta.

Se utiliza a menudo en telemetría, donde instrumentos remotos están enviando información de mediciones cada cierto tiempo. Puesto que los datos cambian lentamente, algunos

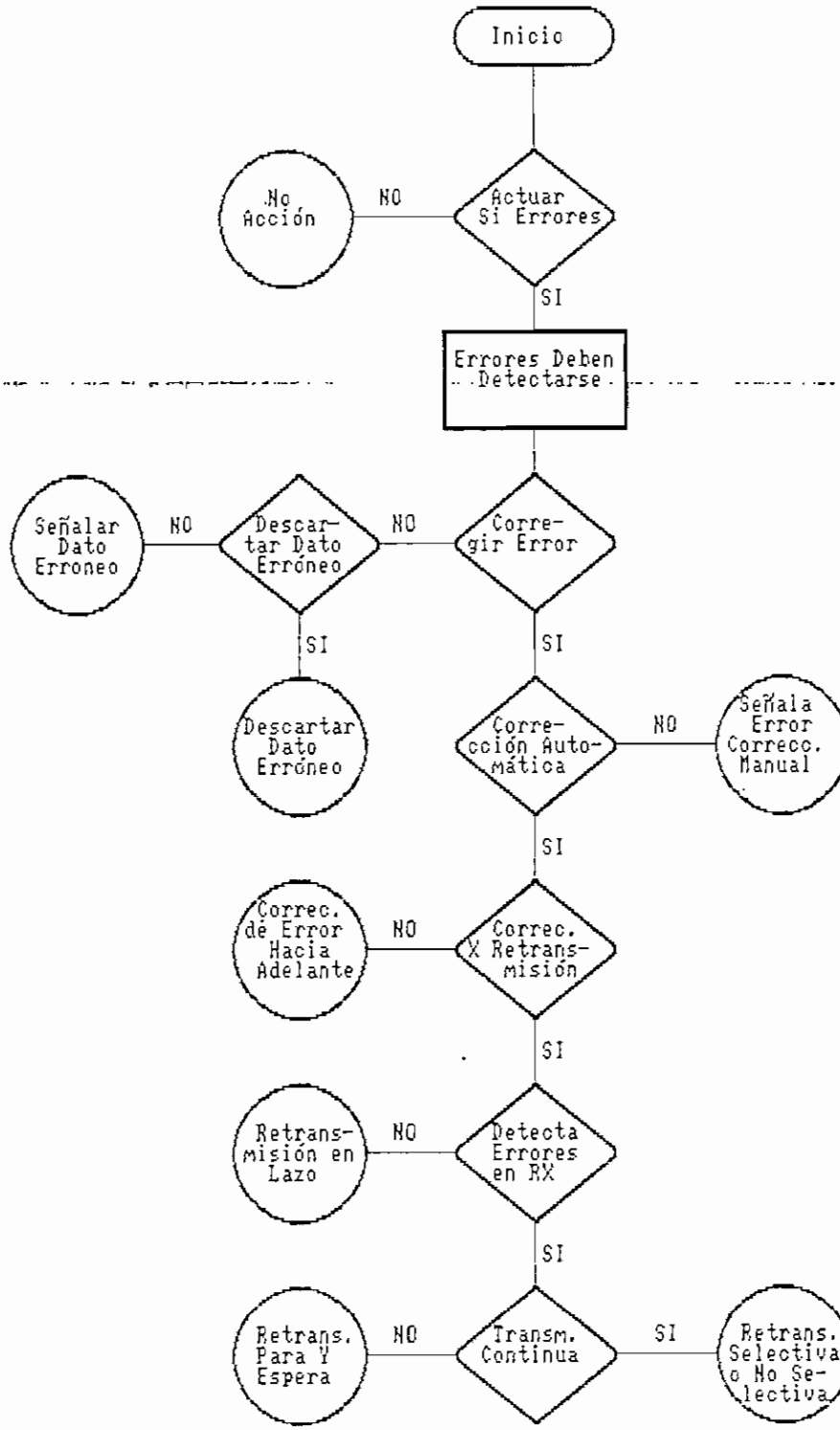


Fig. 1.11 METODOS DE CONTROL DE ERRORES

pueden ser descartados y completados por interpolación luego de recibir la próxima medida libre de error.

- En algunos sistemas simples es posible detectar errores y señalarlos (colocar una bandera), sin que puedan corregirse. Esto es útil para determinar la frecuencia y lugar donde ocurren los errores, dando una idea sobre la integridad del sistema.

- Si la información transmitida es un texto traducible, un operador podría manualmente corregir errores señalados (con bandera). Este procedimiento se complica cuando números o largas corrientes de datos están involucrados, por lo que se utiliza muy poco.

1.3.3 Corrección Automática de Errores.

Además de detectar los errores el sistema es capaz de obtener la información correcta, con alguno de los siguientes métodos:

- Corrección del error hacia adelante.- Es una técnica por la cual la estación receptora reconstruye un mensaje libre de errores de uno contaminado. Esto es posible, transmitiendo información redundante con el mensaje en forma de uno o más duplicados exactos o utilizando alguno de los códigos de caracteres especiales.

La corrección requiere un significativo número de bits de chequeo redundante. En general la habilidad de corregir

una ráfaga de longitud menor o igual a M en un mensaje de N bits necesita que se transmitan $M * \log_2 N$ bits adicionales con cada mensaje, los que no contienen nueva información.

Los códigos de corrección pueden emplearse en la práctica, preferentemente cuando la tasa de error sobre el canal es tan alta que no es conveniente la retransmisión de la información, o en el caso de no disponer de un canal de transmisión de doble sentido. Ninguna de estas situaciones prevalece en sistemas de comunicación de computadores.

- Existen tres métodos de corrección de errores por retransmisión de la información y son:

a) En lazo.

b) Para y espera.

c) Continua.

1.3.3.1 Retransmisión en Lazo.- Este es un modo simple de retransmisión cuando la detección de error se realiza en el transmisor.

La información enviada de la estación A hacia B es nuevamente enviada de B hacia A. Si el mensaje que regresa de B es diferente del original, A invalida la transmisión previa y retransmite el mensaje. No se requiere chequeo de error en el receptor B, puesto que A controla el proceso. Debido a que ambas direcciones del enlace full duplex son

usadas, se impone un 100% de gasto u ocupación para la corrección.

Además un error en el camino de retorno puede ser causa del envío del mensaje nuevamente. La retransmisión en lazo es poco utilizada en computadores.

1.3.3.2 Retransmisión Para y Espera (ARQ/ACK).- El transmisor emite un bloque de información y luego espera un reconocimiento del receptor vía el canal reverso.

Si el bloque es reconocido positivamente (ACK:Acknowledge), es decir sin errores, el transmisor procede a enviar el siguiente bloque; si es reconocido negativamente (NAK:Negative Acknowledge),por la existencia de errores, o el acuse de recibo demora demasiado se retransmite el mismo bloque.

En el transmisor debe guardarse una copia de la información hasta que llegue el asentimiento, liberándose la memoria cuando se recibe un ACK.

Este sistema tiene una aplicación muy común, ya que son simples de implementar y dan satisfactorios resultados en su funcionamiento sobre enlaces donde el tiempo de propagación de la señal es pequeño, como es el caso de circuitos en tierra.

El largo retardo entre reconocimientos reduce la eficiencia del método.

Se puede utilizar para la comunicación un canal full duplex o un canal half duplex.

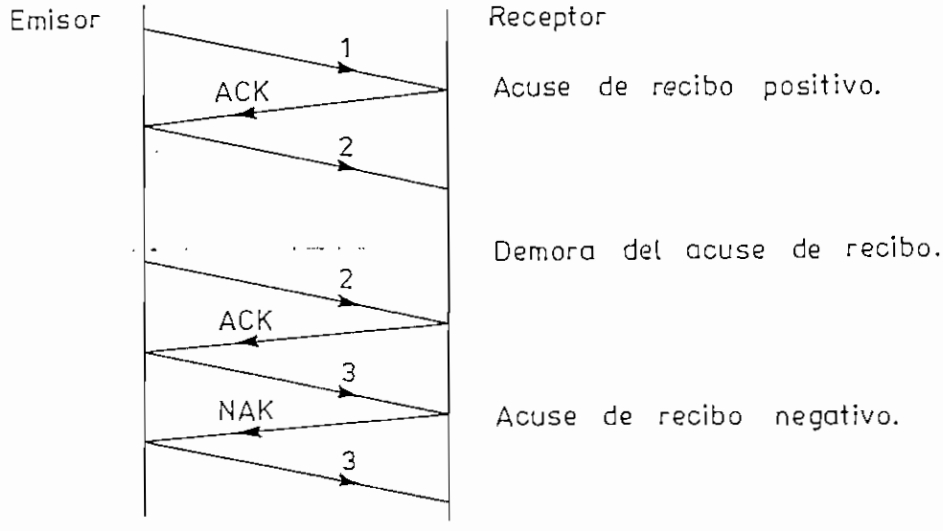


Fig.1.12 Retransmisión Para y Espera

1.3.3.3 Retransmisión Continua (ARQ/NAK).- Una solución más eficiente que la anterior es el envío continuo, que consiste en no detener la transmisión entre bloques contiguos sino enviarlos al receptor numerados. Existen dos formas de retransmisión continua y son: selectiva y no selectiva.

Cuando es no selectiva, si el emisor recibe un reconocimiento negativo (error) del bloque n-ésimo, retransmite dicho bloque y los siguientes.

El método requiere el almacenamiento de bloques de datos y un sistema de numeración para referencia de los reconocimientos.

A continuación se presenta gráficamente esta forma de corrección de errores.

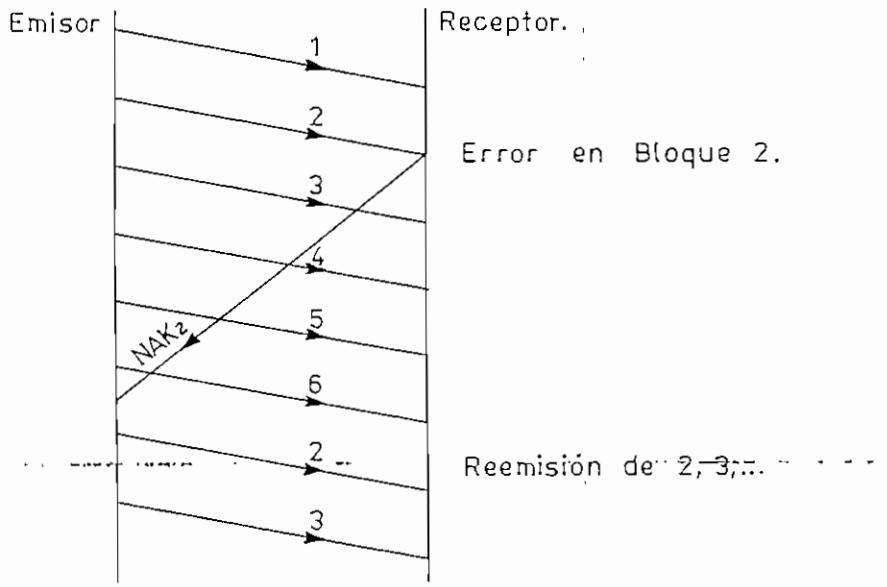


Fig.1.13 Retransmisión Continua No Selectiva

Un mejoramiento con respecto al anterior es el rechazo selectivo, con el cual se retransmite exclusivamente el bloque señalado como erróneo, no los siguientes ya emitidos.

Este procedimiento es más eficiente pero implica controladores más complejos pues los bloques pueden llegar desordenados y es preciso reordenarlos para recomponer el mensaje original. Fig. 1.14.

Para retransmisión continua se requiere un canal full duplex ya que el emisor y receptor pueden transmitir al mismo tiempo, sin embargo, el receptor puede usar una velocidad numérica mucho menor que la del emisor puesto que solamente envía acuses de recibo negativos.

Estos métodos se utilizan en transmisión satelital donde el tiempo de retardo de propagación es alto comparado con el de enlaces en tierra.

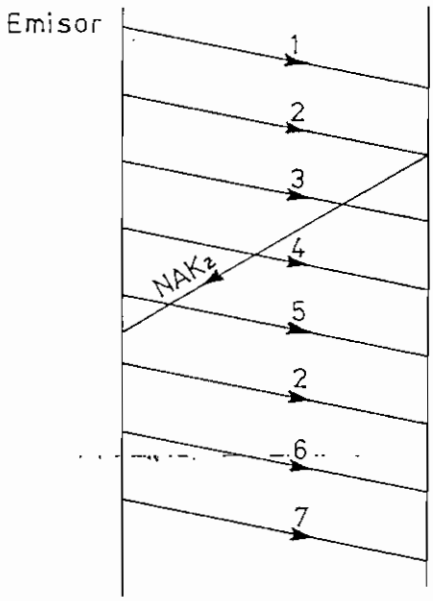


Fig.1.14 Retransmisión Continua Selectiva

Tabla 1.3 Rendimiento según la Retransmisión y el Enlace

TIPO DE TX / ENLACE	PARA Y ESPERA	NO SELECTIVA	SELECTIVA
Veloc. Numérica: 4800 b/s Enlace: 500 Km Long. de Mensaje:1000 bits Tasa de Error: 10 E-5	46%	93%	94%
Veloc. Numérica: 48000 b/s Enlace por satélite Tasa de Error: 10 E-4	Inadecuado	23%	86%

En la tabla anterior se compara el rendimiento de los métodos de corrección por retransmisión en referencia a los enlaces.

1.3.4 Cadencia Eficaz.

Se presenta un análisis simplificado de las estrategias

de retransmisión para y espera y continua. Como medida de la eficiencia se utiliza la cadencia eficaz Ce, que es el cociente entre n número de bits de información de un bloque, y To el tiempo que el circuito permanece ocupado (transmitiendo o esperando) a causa de ese bloque.

En el transmisor se fragmenta la información en bloques de n bits, a los que se añaden m bits correspondientes a las funciones de control.

Las tramas de información y control (bloques con bits de datos o de control de la comunicación), se transmiten con una cadencia de R bits/s,y en el canal de vuelta con una cadencia de Rv bits/s en general distinta de R.

La probabilidad de que un bloque se reciba con error (al menos un bit erróneo) es p, y en medidas efectuadas en circuitos reales se ha observado que es aproximadamente proporcional a la longitud de los bloques ($p=k(n+m)$; $p<<1$ donde k es la tasa de error medida).

El tiempo transcurrido desde que se envia el último bit de información hasta que se recibe el último bit de su correspondiente asentimiento se representa por Tas y está compuesto por: el doble de tiempo de propagación del circuito, el doble de tiempo de giro (turn-around) del modem, si el circuito es semiduplex, el retraso debido al procesado de la trama y generación del asentimiento, y por a/Rv,que es el tiempo de transmisión de los "a" bits de

asentimiento. Se supone despreciable la probabilidad de que una trama de asentimiento se vea afectada por el ruido, dada su generalmente menor longitud.

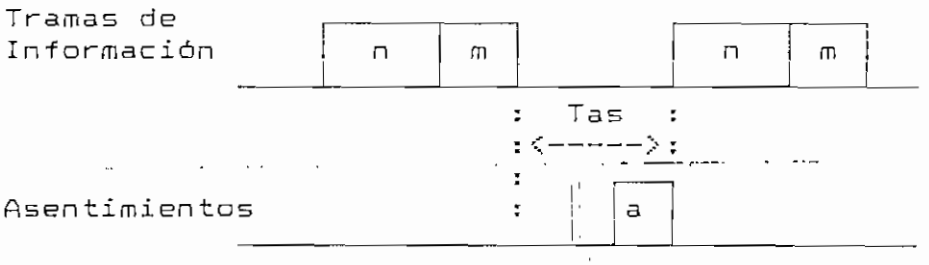


Fig. 1.15 Tas en Para y Espera

El enlace se supone de actividad alternada, es decir, la información fluye en un solo sentido durante un periodo de tiempo, pasado el cual puede invertirse el sentido.

Parada y Espera.- En esta estrategia, el tiempo de ocupación, To, se compone de: el tiempo de transmisión de la trama (n+m)/R, y el tiempo de espera del asentimiento Tas, multiplicado por el número de veces que la trama se ha transmitido (Nt). Este último es una variable aleatoria de valor medio:

$$Nt = 0(1-p)+1\ p(1-p)+2\ p\ p\ (1-p)+\ldots\ldots = \frac{1}{1-p}$$

donde p es la probabilidad de error de un bit.

Por tanto:

$$To = [(n+m)/R + Tas]Nt$$

$$Ce = \frac{n}{To} = (1-p)\ \frac{n}{(n+m)+TasR}\ R \quad bits/s$$

Envío Continuo, retransmisión no selectiva.- En esta estrategia se envían los bloques sin paradas intermedias entre ellos (salvo que no haya más que transmitir), y cuando se recibe el rechazo de un bloque se retransmite el rechazado y los posteriores. El canal debe ser full duolex.

El tiempo de ocupación se compone de: el tiempo de espera del asentimiento Tas, multiplicado por el número medio de retransmisiones Nt-1: más la duración de un bloque (n+m)/R multiplicada por el número medio de transmisiones, Nt.

$$To = Tas(Nt-1) + [(n+m)/R]Nt$$

$$Ce = \frac{n}{To} = (1-o)\ \frac{n}{(n+m)+p\ Tas\ R}\ R \quad bits/s$$

Obsérvese que en este caso la influencia de la espera del asentimiento queda reducida por el factor p que es menor que la unidad.

Envío Continuo, retransmisión selectiva.- El rechazo selectivo implica que al recibir un reconocimiento negativo de un bloque se retransmite exclusivamente el rechazado. Por tanto, el único tiempo ocupado en la línea a causa de un bloque es su duración multiplicada por el número medio de transmisiones. Y la cadencia eficaz es:

$$To = [(n+m)/R]Nt$$

$$Ce = \frac{n}{To} = (1-p)\ \frac{n}{n+m}\ R \quad bits/s$$

A continuación se presenta un ejemplo para calcular la cadencia eficaz con las diferentes estrategias.

Ejemplo: Si R= Rv= 1200 b/s
n= 1000 bits
m= 48 bits
p= 5^{-6} (n+m)
K= 5^{-6} tasa de error.

Retardo de asentimiento de 200 mseg más la duración del asentimiento a= 48 bits (TasR= 288 bits).

En la tabla 1.4 se muestra las cadencias eficaces y rendimientos obtenidos con los diferentes métodos de retransmisión.

Tabla 1.4 Cadencia Eficaz y Rendimiento

	Parada y Espera	Rechazo no Selectivo	Rechazo Selectivo
Ce (bits/s)	893	1138	1139
Ce/R (%)	74	94.8	94.9

Obsérvese el aumento (20%) de rendimiento obtenido con el envío continuo con respecto a la parada y espera, y la casi coincidencia de resultados entre las dos formas de rechazo. Esta cercanía no es general, ya que la retransmisión no selectiva se degrada respecto a la selectiva a medida que el término p.Tas.R aumenta, lo que ocurre si el circuito es muy ruidoso, o bien si el retardo del asentimiento es elevado (p. ejm. un enlace vía satélite).

Dada la proximidad de eficiencia entre ambos tipos de rechazo, en circuitos telefónicos terrestres de bajo ruido, el rechazo selectivo no suele utilizarse, puesto que el escaso aumento en el rendimiento no justifica el aumento de complejidad requerido para reordenar las tramas retransmitidas.

Las tramas utilizadas en las estrategias de envío continuo deben contener un campo de longitud finita, para numerar los bloques. Sea M el número máximo de bloques que permite numerar ese campo. Si el retardo de asentimiento es mayor que la duración máxima de M tramas, se produce una parada por agotarse los números disponibles para nuevos bloques. Por tanto, hay que utilizar un M adecuado al retardo de asentimientos, Tas.

CAPITULO II

FUNDAMENTOS DE LOS CODIGOS CICLICOS.

Los códigos cíclicos forman una subclase importante de los códigos lineales.

Tienen como características favorables que: su implementación (codificador y decodificador) es bastante simple, pues tan solo requieren registros de desplazamiento y conexiones de realimentación; se adaptan a la detección de errores independientes o en paquetes. Además su estructura algebraica hace posible encontrar métodos prácticos para la decodificación.

2.1 Descripción y Propiedades de los Códigos Cíclicos.

2.1.1 Definición y Representación.

Definición: Un código lineal C(n,k) [1] se denomina cíclico, si con cada permutación circular de un vector de código en C se obtiene otro vector contenido en C.

Si los componentes del vector de código $v=(v_0,v_1,..v_{n-1})$ se desplazan cíclicamente un lugar hacia la derecha, se

obtiene otra n_tupla, $v^{(1)}= (v_{n-1} ,v_0 ,...v_{n-2})$.

Del mismo modo, si se realizan i desplazamientos de v a la derecha o lo que equivale a hacerlo n-i lugares a la izquierda resulta:

$$v^{(i)} = (v_{n-i} ,v_{n-i+1} ,.....v_{n-1} ,v_0 ,.....v_{n-i-1})$$

Los componentes de $v =(v_0,v_1,...v_{n-1})$ pueden tratarse como coeficientes de un polinomio de grado menor o igual a n-1:

$$v(X)= v_0 +v_1X +v_2X^2 +....+v_{n-1}X^{n-1}$$

con v_i=0 o 1, que se denomina **polinomio de código.** Entre v(X) y v existe correspondencia uno a uno.

Así, para $v^{(i)}$ se tiene un $v^{(i)}(X)$,

$$v^{(i)}(X)= v_{n-i} +v_{n-i+1}X +...+v_{n-1}X^{i-1} +v_0X^i +v_1X^{i+1}+...$$
$$...+v_{n-i-1}X^{n-1}$$

después de i desplazamientos cíclicos [2].

Relación entre v(X) y $v^{(i)}(X)$.- De acuerdo con [2] $v^{(i)}(X)$ se obtiene multiplicando v(X) por X^i,

$$X^iv(X)= v_0X^i +v_1X^{i+1} +....+v_{n-i-1}X^{n-1}+ v_{n-i}X^n +v_{n-i+1}X^{n+1}....$$
$$...+v_{n-1}X^{n+i-1}$$

"Un código bloque de longitud n y 2^k palabras código se dice que es un código bloque lineal (n,k) si y sólo si, éstas forman un subespacio k dimensional del espacio formado por todas las n_tuplas que pertenecen al campo GF (2)"[1]

Sumando dos veces el polinomio $v_{n-i} + v_{n-i+1}X + \ldots + v_{n-1}X^{i-1}$ resulta:

$$X^i v(X) = v_{n-i} + v_{n-i+1}X + \ldots + v_{n-1}X^{i-1} + v_0 X^i + v_1 X^{i+1} + \ldots$$
$$\ldots + v_{n-i-1}X^{n-1} + v_{n-i}X^n + v_{n-i+1}X^{n+1} \ldots + v_{n-1}X^{n+i-1} +$$
$$+ v_{n-i} + v_{n-i+1}X + \ldots + v_{n-1}X^{i-1}$$

La ecuación anterior puede escribirse de la siguiente forma:

$$X^i v(X) = v_{n-i} + v_{n-i+1}X + \ldots + v_{n-1}X^{i-1} + v_0 X^i + \ldots + v_{n-i-1}X^{n-1} +$$
$$+ v_{n-i}(X^n+1) + v_{n-i+1}X\,(X^n+1) + \ldots + v_{n-1}X^{i-1}\,(X^n+1)$$

$$X^i v(X) = q(X)\,(X^n+1) + v^{(i)}(X) \qquad (2.1)$$

donde $q(X) = v_{n-i} + v_{n-i+1}X + \ldots + v_{n-1}X^{i-1}$.

De (2.1) se observa que el polinomio de código $v^{(i)}(X)$ es el residuo que resulta al dividir $X^i v(X)$ por X^n+1.

2.1.2 Propiedades de los Códigos Cíclicos.

Las propiedades algebraicas de los códigos cíclicos se presentan en los siguientes teoremas:

Nota: Las operaciones de polinomios se realizan en aritmética módulo-2. El detalle de estas operaciones y algunas propiedades a utilizarse en adelante se encuentran en el Apéndice A.

"En K (conjunto de polinomios binarios de grado $\leq n-1$),

$X.(b_0 + b_1X + \ldots + b_{n-1}X^{n-1}) = b_{n-1} + b_0X + b_1X^2 + \ldots + b_{n-2}X^{n-1}$

es decir que la multiplicación por X corresponde a un desplazamiento cíclico. Sujeto a la relación $X^n=1$" [2]

Teorema 2.1.2.1 El polinomio diferente de cero de grado mínimo en un código cíclico es único.

Demostración: Sea $g(X)= g_0 +g_1X +... +g_{r-1}X^{r-1} +X^r$ un polinomio de grado mínimo. Suponiendo que:

$$g'(X)= g_0' +g_1'X +... +g_{r-1}'X^{r-1} +X^r$$

también sea de grado mínimo,

$$g(X)+g'(X)= (g_0+g_0') +(g_1+g_1')X +... +(g_{r-1}+g_{r-1}')X^{r-1}$$

es de grado r-1, menor que el grado de g(X), lo que no es posible.Por tanto, g(X)+g'(X)=0 de lo que g(X)=g'(X). L.Q.D.

Teorema 2.1.2.2 Sea $g(X)= g_0 +g_1X +...+g_{r-1}X^{r-1} +X^r$ un polinomio diferente de cero de grado mínimo en un código cíclico C(n,k). Entonces el término constante g_0 debe ser igual a 1.

Demostración: Suponiendo que $g_0 =0$, entonces

$$g(X)= g_1X +g_2X^2 +...+g_{r-1}X^{r-1} +X^r$$

si se desplaza cíclicamente n-1 lugares a la derecha (o un lugar a la izquierda), resulta el polinomio de código

$$g_1 +g_2X + ...+g_{r-1}X^{r-2} +X^{r-1}$$

que es de grado menor que g(X) que es el de grado mínimo, siendo una contradicción, $g_0 \neq 0$. L.Q.D.

Del teorema anterior se establece que el polinomio de código de grado mínimo es de la forma:

(2.2) $$g(X)= 1 +g_1X +g_2X^2 +...+g_{r-1}X^{r-1} +X^r$$

Desplazamientos ciclicos de g(X).- Los polinomios $Xg(X)$, $X^2g(X),\ldots,X^{n-r-1}g(X)$, que tienen grados r+1, r+2,..... ....,n-1 respectivamente, se expresan como:

$Xg(X)=g^{(1)}(X)$, $X^2g(X)=g^{(2)}(X),\ldots,$ $X^{n-r-1}g(X)=g^{(n-r-1)}(X)$

son desplazamientos cíclicos de g(X) y por tanto polinomios de código en C.

Una combinación lineal de g(X), $Xg(X),\ldots,X^{n-r-1}g(X)$ es de la forma:

$$v(X)= u_0g(X) +u_1Xg(X)+\ldots+u_{n-r-1}X^{n-r-1}g(X)$$

$$v(X)= (u_0+u_1X+\ldots+u_{n-r-1}X^{n-r-1})g(X), \tag{2.3}$$

donde u_i=0 o 1, también es un polinomio de código.

Teorema 2.1.2.3 Siendo $g(X)= 1 +g_1X+\ldots+g_{r-1}X^{r-1} +X^r$ un polinomio de código diferente de cero y de grado mínimo en un código cíclico C(n,k). Un polinomio binario de grado menor o igual a "n-1" es un polinomio de código si y sólo si es múltiplo de g(X).

Demostración: Si v(X) es un polinomio binario de grado menor o igual a n-1, suponiendo que es múltiplo de g(X), entonces:

$$v(X)= (a_0 +a_1X +\ldots+a_{n-r-1}X^{n-r-1})g(X)$$
$$= a_0g(X) +a_1Xg(X) +\ldots+a_{n-r-1}X^{n-r-1}g(X)$$

es una combinación lineal de g(X), $Xg(X),\ldots,X^{n-r-1}g(X)$, y por tanto polinomio de código en C. Esto prueba la primera parte del teorema.

Dividiendo v(X) (polinomio de código en C), por g(X) [3] se obtiene:

$$v(X)= a(X)g(X)+b(X)$$

donde b(X) es cero o de menor grado que g(X).

Dado que la adición y sustracción en módulo-2 son iguales,

$$b(X)= v(X)+a(X)g(X)$$

De la demostración anterior, a(X)g(X) y v(X) son polinomios de código y por tanto b(X) también lo es. Si b(X)=0, sería de menor grado que g(X), lo que contradice la hipótesis. Así, b(X)=0 y v(X) es múltiplo de g(X). L.Q.D.

Si v(X)=a(X)g(X), los coeficientes de a(X) pueden tener el valor 0 o 1. El número de polinomios binarios de grado menor o igual a n-1 múltiplos de g(X) es 2^{n-r} ,en el teorema (2.1.2.3) se demostró que éstos son polinomios de código.

Para una longitud k se pueden tener 2^k posibles mensajes diferentes, un código cíclico C (n,k) contiene los 2^k polinomios de código correspondientes, de modo que $2^{n-r}=2^k$,y r=n-k.

El polinomio de código diferente de cero con grado mínimo en un código cíclico C (n,k) tiene la siguiente

---

"Siendo f(D) y g(D) polinomios sobre GF (q) y con g(D) de grado por lo menos 1. Entonces existen dos únicos polinomios h(D) y r(D) sobre GF (q) tales que:

$$f(D)=g(D)h(D)+r(D)$$

donde el grado de r(D) es menor que el de g(D)" [3]

forma:

(2.4) $$g(X)= 1 +g_1X +g_2X +...+g_{n-k-1}X^{n-k-1} +X^{n-k}$$

Teorema 2.1.2.4 En un código ciclico C (n,k) existe un único polinomio de código de grado n-k.

$$g(X)= 1 +g_1X +g_2X +...+g_{n-k-1}X^{n-k-1} +X^{n-k}$$

Este teorema es una conclusión de los anteriores.

Por el teorema (2.1.2.3), v(X) puede expresarse de siguiente forma:

(2.5) $$v(X)= u(X)g(X)$$

$$v(X)= (u_0 +u_1X +...+u_{k-1}X^{k-1})\ g(X)$$

Si los coeficientes de u(X), u_0 ,u_1 ,...u_{k-1} son los k dígitos de la información; v(X) es el correspondiente polinomio de código. La codificación puede realizarse multiplicando el mensaje u(X) por g(X).

Por lo tanto, un código ciclico es especificado completamente por el polinomio de grado mínimo g(X) dado en (2.4) llamado polinomio generador del código, cuyo grado es igual al número de dígitos de chequeo de paridad [4].

Teorema 2.1.2.5 El polinomio generador g(X) de un código ciclico es un factor de X^n+1.

"El codificador, por cada mensaje de k dígitos generará un nuevo bloque de información v de n elementos que cumplirán con la condición de n>k. De esta manera se ha incrementado n-k dígitos, denominándose dígitos verificadores de paridad o de control" [4]

Demostración: Multiplicando g(X) por X^k se tiene un polinomio $X^k g(X)$ de grado n, el mismo que si se divide por X^n+1 da como resultado:

(2.6) $$X^k g(X) = (X^n+1)+g^{(k)}(X)$$

donde $g^{(k)}(X)$ es el residuo.

De (2.1), $g^{(k)}(X)$ se obtiene desplazando g(X) k veces a la derecha, y por consiguiente es múltiplo de g(X); es decir $g^{(k)}(X)=a(X)g(X)$. Reemplazando en (2.6) se obtiene:

(2.7) $$X^n+1 = \{X^k+a(X)\}\ g(X)$$

Por tanto, g(X) es un factor de X^n+1. L.Q.D.

Teorema 2.1.2.6 **Si g(X) es un polinomio de grado n-k y es un factor de X^n+1, entonces g(X) genera un código cíclico C (n,k).**

Demostración: Considerando los k polinomios g(X),Xg(X),.... ....$X^{k-1}g(X)$ de grado menor o igual a n-1 múltiplos de g(X), y su combinación lineal se tiene:

$$v(X)= a_0 g(X) + a_1 X g(X) + \ldots + a_{k-1} X^{k-1}\ g(X)$$
$$=(a_0 + a_1 X + \ldots + a_{k-1} X^{k-1})\ g(X)$$

que también es de grado menor o igual a n-1 y múltiplo de g(X).

Hay un total de 2^k de estos polinomios que forman un código lineal (n,k).

Ejemplo (2.1).- Para n=7,k=4 el polinomio X^7+1 puede factorarse como sigue:

$$X^7+1= (1+X)(1+X+X^3)(1+X^2+X^3)$$

a) Considerando el polinomio generador $g(X)=1+X+X^3$ y el mensaje a codificar u=(1100) que corresponde a $u(X)=1+X$, se tiene:

$$v(X)= (1+X)(1+X+X^3)$$
$$= 1 +X^2 +X^3 +X^4.$$

El vector de código resultante: v=(1011100).

b) Si u=(0110), $u(X)=X+X^2$

$$v(X)= (X+X^2)(1+X+X^3)$$
$$= X +X^3 +X^4 +X^5$$

El vector de código correspondiente: v=(0101110).

Con este procedimiento se obtienen las 16 palabras código, pero debe notarse que no siempre los últimos 4 dígitos de v son iguales a los de información.

2.2.1 Código Cíclico Sistemático.

Dado el polinomio generador g(X) de un código cíclico (n,k), el código obtenido puede tener forma sistemática, es decir que k dígitos del vector de código corresponden a la información inalterada, y n-k bits son de chequeo de paridad.

Donde hay k dígitos de información y n-k dígitos de chequeo de paridad que son el residuo de dividir $X^{n-k}\,u(X)$ por $g(X)$.

La codificación de un código cíclico en forma sistemática consiste en los siguientes pasos:

1. Premultiplicar el mensaje $u(X)$ por X^{n-k}.
2. Obtener el residuo $b(X)$ -dígitos de chequeo de paridad- de dividir $X^{n-k}\,u(X)$ por el polinomio generador $g(X)$.
3. Combinar $b(X)$ y $X^{n-k}\,u(X)$, obteniendo el polinomio de código $b(X)+X^{n-k}\,u(X)$.

Ejemplo (2.2).- Para $n=7, k=4$ y el polinomio generador $g(X)=1+X+X^3$.

Si el mensaje a codificar es: $u=(1010)$, $u(X)=1+X^2$.

La división de $X^3\,u(X)=X^3+X^5$ por $g(X)=1+X+X^3$ se realiza de la siguiente manera:

$$\begin{array}{r|l} & X^2 \;:\; \text{(cuociente)} \\ \hline X^3+X+1 & X^5 \quad +X^3 \\ & X^5 \quad +X^3 \quad +X^2 \\ \hline & X^2 \quad \text{(residuo)} \end{array}$$

por tanto $b(X)=X^2$, el polinomio de código (2.10) es:

$$\begin{aligned} v(X) &= b(X)+X^3\,u(X) \\ &= X^2+X^3+X^5 \end{aligned}$$

el vector de código resultante es: $v=(0011010)$, donde los cuatro últimos dígitos corresponden a los del mensaje.

La tabla de los vectores código y polinomios de código para el código cíclico (7,4) en forma sistemática, generado por el polinomio $g(X)=1+X+X^3$ se presentan en la tabla 2.1.

Tabla 2.1 Código Cíclico Sistemático (7,4)

$g(X)= 1+X+X^3$

Mensaje	Vect.de Código	Polinomio de Código
0000	0000000	$0.g(X)= 0$
1000	1101000	$1g(X)= 1+X+X^3$
0100	0110100	$xg(X)= X+X^2+X^4$
1100	1011100	$(1+X)g(X)= 1+X^2+X^3+X^4$
0010	1110010	$(1+X^2)g(X)= 1+X+X^2+X^5$
1010	0011010	$X^2g(X)= X^2+X^3+X^5$
0110	1000110	$(1+X+X^2)g(X)= 1+X^4+X^5$
1110	0101110	$(X+X^2)g(X)= X+X^3+X^4+X^5$
0001	1010001	$(1+X+X^3)g(X)= 1+X^2+X^6$
1001	0111001	$(X+X^3)g(X)= X+X^2+X^3+X^6$
0101	1100101	$(1+X^3)g(X)= 1+X+X^4+X^6$
1101	0001101	$X^3g(X)= X^3+X^4+X^6$
0011	0100011	$(X+X^2+X^3)g(X)= X+X^5+X^6$
1011	1001011	$(1+X+X^2+X^3)gX)= 1+X^3+X^5+X^6$
0111	0010111	$(X^2+X^3)g(X)= X^2+X^4+X^5+X^6$
1111	1111111	$(1+X^2+X^5)g(X)=$ $= 1+X+X^2+X^3+X^4+X^5+X^6$

2.2.2 Matrices Generadora y de Chequeo de Paridad en Códigos Cíclicos.

Algunos conceptos sobre los códigos bloque en general y su representación en forma de matrices se encuentra en el Apéndice B.

Considerando un código cíclico (n,k) con un polinomio generador $g(X)= g_0 +g_1X +...+g_{n-k}X^{n-k}$, si las n_tuplas

correspondientes a los k polinomios de código g(X), Xg(X), ...X^{k-1}g(X) en C se usan como filas de una matriz kxn, se obtiene la matriz generadora del código ciclico.

(2.11)

$$G=\begin{bmatrix} g_0 & g_1 & g_2 & g_3 & \cdots\cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & g_2 & \cdots\cdots & & g_{n-k} & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \cdots\cdots & & g_{n-k} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & & & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & g_0 & g_1 & \cdots\cdots & & & g_{n-k} \end{bmatrix}$$

$g_0 = g_{n-k} = 1$

$$v = u \cdot G$$

En general con G se obtiene un código que no tiene forma sistemática.

Dado que el polinomio generador g(X) es un factor de X^n+1, se tiene:

(2.12) $$X^n+1 = g(X)h(X)$$

donde el polinomio h(X) tiene grado k y es de la forma:

$$h(X) = h_0 + h_1X + \ldots + h_kX^k$$

con $h_0 = h_k = 1$.

Para $v=(v_0, v_1, \ldots, v_{n-1})$ vector de código en C, v(X)=a(X) g(X). Multiplicando por h(X) resulta:

$$v(X)h(X) = a(X)g(X)h(X)$$

$$= a(X)(X^n+1) = a(X) + X^n a(X)$$

con $a(X) = a_0 + a_1X + \ldots + a_{n-r-1}X^{n-r-1}$; r=n-k

Puesto que a(X) es de grado menor o igual a k-1, las potencias X^k , X^{k+1} ,....., X^{n-1} no aparecen en la suma

a(X) + X^na(X) y los coeficientes correspondientes en el producto v(X)h(X) son cero.

De lo que se obtienen las siguientes n-k igualdades:

$$(2.13) \qquad \sum_{i=0}^{k} h_i \, v_{n-1-j} = 0 \qquad \text{para } 1 \le j \le n-k$$

Si se define el recíproco de h(X) como sigue:

$$(2.14) \qquad X^k \, h(X^{-1}) = h_k + h_{k-1}X + h_{k-2}X^2 + \ldots + h_0 X^k$$

éste también es un factor de X^n+1 . El polinomio $X^k \, h(X^{-1})$ genera un código cíclico (n,n-k) con la siguiente matriz generadora (n-k)xn

$$(2.15) \qquad H = \begin{bmatrix} h_k & h_{k-1} & h_{k-2} & & & h_0 & 0 & \ldots & 0 \\ 0 & h_k & h_{k-1} & h_{k-2} & & & h_0 & 0 & \ldots 0 \\ 0 & 0 & h_k & h_{k-1} & h_{k-2} & & & h_0 & \ldots 0 \\ & \vdots & & \vdots & & & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & h_k & h_{k-1} & \ldots & & h_0 \end{bmatrix}$$

Utilizando las n-k igualdades dadas por (2.13) en el producto $v \cdot H^T$ se puede comprobar que el resultado es cero, por lo que se concluye que cada vector v en C es ortogonal a cada una de las filas de H [e.d. $v \cdot H^T = 0$]. Por lo que H es una matriz de chequeo de paridad del código C, siendo h(X) el polinomio de paridad.

Asi, el código cíclico es especificado por el polinomio de paridad.

El siguiente teorema indica una importante propiedad de la matriz de chequeo de paridad.

Teorema 2.2.2.1. Siendo C un código ciclico (n,k) generado por el polinomio g(X). El código dual de C [5] es también un código ciclico y es generado por el polinomio $X^k\, h(X^{-1})$, donde $h(X)=(X^n+1)/g(X)$.

2.2.2.1 Matriz Generadora en forma Sistemática.- Esta matriz puede obtenerse con el siguiente procedimiento:

Dividiendo X^{n-k+i} por g(X) para i=0,1,...,k-1

(2.16) $$X^{n-k+i}= a_i(X)g(X)+b_i(X)$$

donde $b_i(X)$ es el residuo y tiene la siguiente forma:

$$b_i(X)= b_{i0} +b_{i1}X +...+b_{i,n-k-1}X^{n-k-1}$$

Puesto que $b_i(X)+X^{n-k+i}$ para i=0,1,...,k-1 son múltiplos de g(X), también son polinomios de código y pueden arreglarse como una matriz kxn.

(2.17)

$$G=\begin{bmatrix} b_{00} & b_{01} & b_{02} & \cdots & b_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ b_{10} & b_{11} & b_{12} & \cdots & b_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ b_{20} & b_{21} & b_{22} & \cdots & b_{2,n-k-1} & 0 & 0 & 1 & \cdots & 0 \\ & & & \vdots & & & & \vdots & & \\ b_{k-1,0} & b_{k-1,1} & b_{k-1,2} & \cdots & b_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

La correspondiente matriz de chequeo de paridad es:

(2.18)

$$H=\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & b_{00} & b_{10} & b_{20} & \cdots & b_{k-1,0} \\ 1 & 0 & 0 & \cdots & 0 & b_{01} & b_{11} & b_{21} & \cdots & b_{k-1,1} \\ 0 & 1 & 0 & \cdots & 0 & b_{02} & b_{12} & b_{22} & \cdots & b_{k-1,2} \\ & & & \vdots & & & & \vdots & & \\ 0 & 0 & 0 & \cdots & 1 & b_{0,n-k-1} & b_{1,n-k-1} & b_{2,n-k-1} & \cdots & b_{k-1,n-k-1} \end{bmatrix}$$

Ejemplo (2.3).- Para el código (7,4) con $g(X)=1+X+X^3$ se tiene la matriz generadora:

$$G= \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Siguiendo el procedimiento indicado anteriormente para obtener la matriz del código sistemático. dividiendo X^3, X^4, X^5, X^6 por $g(X)$ resulta:

$$X^3 = g(X) + (1+X)$$

$$X^4 = Xg(X) + (X+X^2)$$

$$X^5 = (X^2+1)g(X) + (1+X+X^2)$$

$$X^6 = (X^3+X+1)g(X) + (1+X^2)$$

Arreglando las ecuaciones de arriba se obtienen los polinomios de código:

$$v_0(X) = 1 + X + X^3$$

$$v_1(X) = X + X^2 + X^4$$

$$v_2(X) = 1 + X + X^2 + X^5$$

$$v_3(X) = 1 + X^2 + X^6$$

La matriz en forma sistemática (arreglo de acuerdo a 2.17) es:

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

"Las 2^{n-k} combinaciones lineales de las filas de la matriz H forman un código lineal $(n,n-k)$ representado por Cd cuyos elementos son todos diferentes a los que pertenecen al código C, es decir que para todo v que pertenece a C y para todo w que pertenece a Cd se tiene que:

$$v \cdot w = 0$$

a Cd se le conoce como código dual de C" [5]

Partiendo de G se puede encontrar la matriz de chequeo de paridad H. Conforme a (2.18):

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

la misma que genera el código dual C_d (7.3).

Para el código C (7,4) con el mensaje u =(1011). realizando el producto u·G se obtiene el siguiente vector de código con estructura sistemática:

$$v = (1001011)$$

Para el código dual C_d (7.3) con u' =(111), realizando el producto u'·H el resultado es:

$$w = (1110010)$$

$$v \cdot w = 1 + 0 + 0 + 0 + 0 + 1 + 0 = 0$$

2.2.3 Circuitos de Codificación para Códigos Cíclicos.

Los pasos señalados para obtener un código cíclico en forma sistemática (2.2.1) pueden cumplirse utilizando un circuito de división que consiste en un desplazamiento lineal de n-k etapas con conexiones de realimentación basadas en los coeficientes del polinomio generador $g(X)= 1+g_1X+g_2X^2 +...+g_{n-k-1}X^{n-k-1}+X^{n-k}$.

Al circuito ingresa el mensaje de bit en bit por lo que la división se realiza serialmente, y corresponde a la

fig. 2.1.

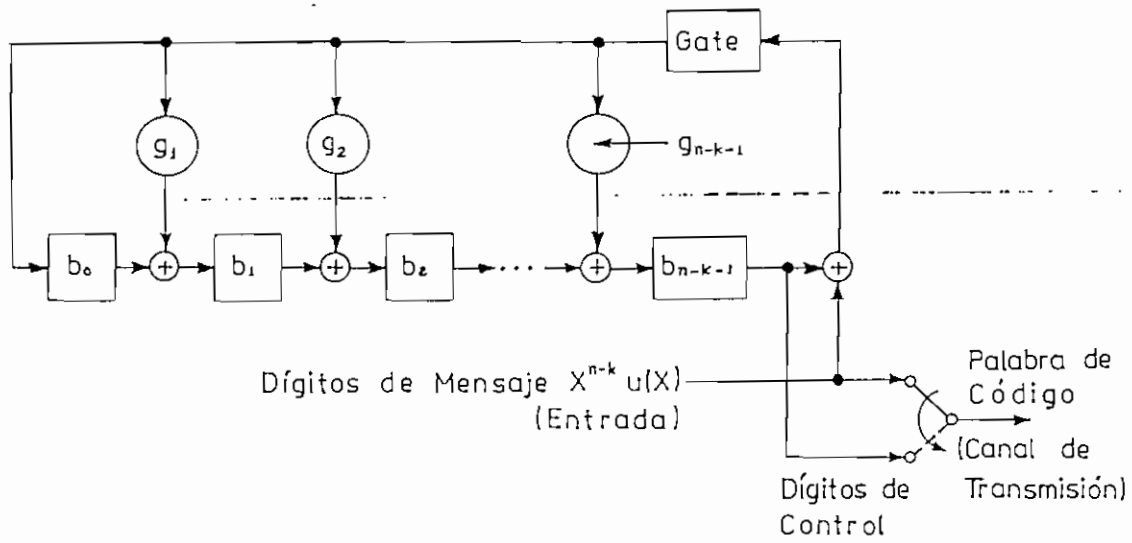


Fig.2.1 Codificador de un Código Cíclico (n,k)
basado en el polinomio generador

Su funcionamiento es el siguiente:

1. Cuando el switch está en ON los k dígitos de información son desplazados en el circuito y simultáneamente entran al canal de comunicación, lo que equivale a premultiplicar u(X) por X^{n-k}.

Una vez que el mensaje completo ingresa, los n-k dígitos contenidos en los registros son el residuo de la división y corresponden a los dígitos de chequeo de paridad.

2. Se interrumpe la conexión de realimentación poniendo el switch en OFF.

3. Se desplazan los dígitos de chequeo de paridad y se

envían al canal, para completar la palabra codificada.

Ejemplo (2.5).- Al código cíclico (7.4) generado por $g(X)=1+X+X^3$, le corresponde el circuito codificador de la fig. 2.2.

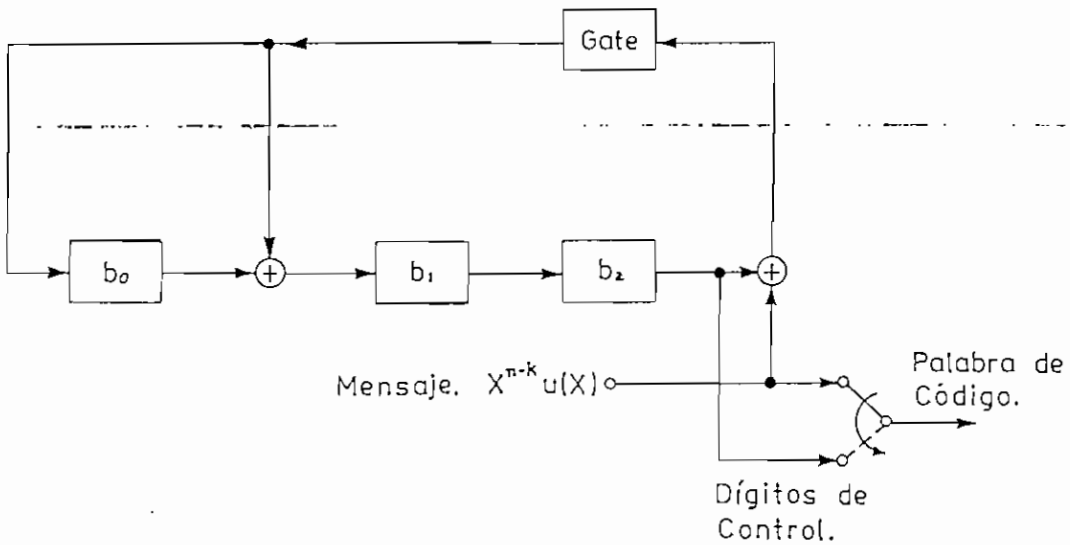


Fig.2.2 Codificador del código cíclico (7,4)

Para el mensaje u=(1110), los desplazamientos en los registros se presentan en la tabla 2.2.

Tabla 2.2

Entrada	Contenido de los Registros
0	0 0 0 Estado Inicial
1	0 0 0 Primer Desplazamiento
1	1 1 0 Segundo Desplazamiento
1	0 1 0 Bits de paridad

El vector que se transmite es v=(0101110).

La codificación también puede realizarse utilizando el polinomio de paridad $h(X)= h_0 +h_1X +...+h_kX^k$.

El vector $v=(v_0,v_1,\ldots,v_{n-1})$ debe satisfacer las n-k igualdades de (2.13); puesto que $h_k=1$ es posible escribir:

$$v_{n-k-j} = \sum_{i=0}^{k-1} h_i\, v_{n-i-j} \qquad 1\leq j\leq n-k \tag{2.19}$$

Los n-k dígitos de chequeo de paridad (v_0 ,v_1 ,v_2 ,.....,v_{n-k-1}) se obtienen aplicando la regla anterior y el circuito que la satisface es el presentado en la fig. 2.3.

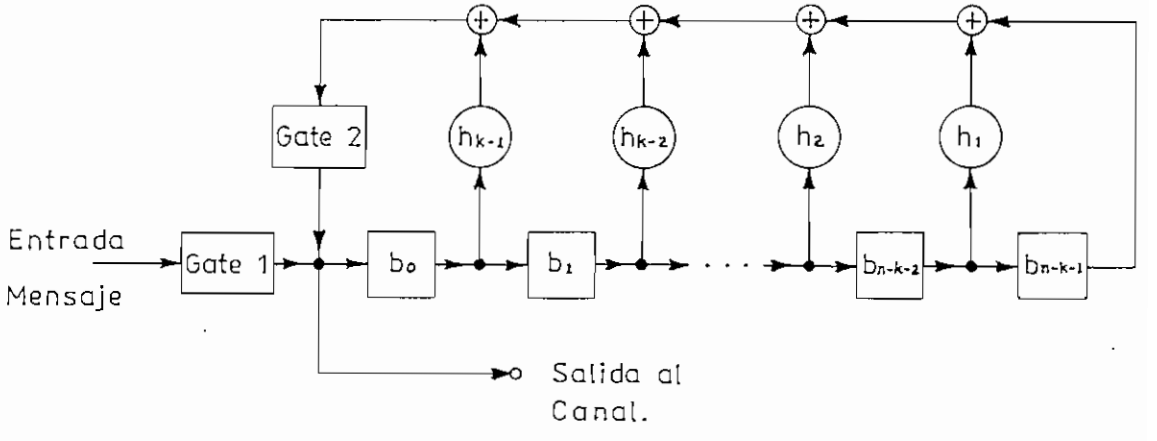


Fig.2.3 Codificador de un Código Cíclico (n,k) basado en el polinomio de paridad.

Las conexiones de realimentación corresponden a los coeficientes del polinomio de paridad h(x),($h_0=h_k=1$).

Este circuito usa k etapas de registros de desplazamiento.

2.3 Síndrome y Detección de Errores.

Si se transmite un vector v y se recibe $r=(r_0,r_1,.r_{n-1})$; debido a que el canal de comunicaciones tiene ruido estos dos vectores pueden no ser iguales.

El primer paso para decodificar un código lineal es calcular el síndrome, que es $s=r.H^T$, donde H es la matriz de chequeo de paridad. Si el síndrome es cero quiere decir que r es un vector de código y el decodificador lo acepta como el transmitido, pero si el síndrome es diferente de cero indica que r no es un vector de código y un error ha sido detectado.

Para un código lineal sistemático, el síndrome es simplemente el vector suma de los dígitos de paridad recibidos y los recalculados a partir de los bits de información que llegan al receptor.

Si el vector recibido se trata como un polinomio de grado menor o igual a n-1 y se divide por g(X) se tiene:

$$r(X)= r_0 +r_1X +r_2X^2 +....+r_{n-1}X^{n-1}$$

$$r(X)= a(X)g(X)+s(X) \qquad (2.20)$$

donde el residuo s(X) es un polinomio de grado igual o menor a n-k-1. Los n-k coeficientes de s(X) forman el síndrome.

Un circuito de división igual al empleado para

codificación con n-k etapas de desplazamiento puede emplearse para calcular el síndrome, con la diferencia que el polinomio recibido r(X) es desplazado en el circuito desde el lado izquierdo.

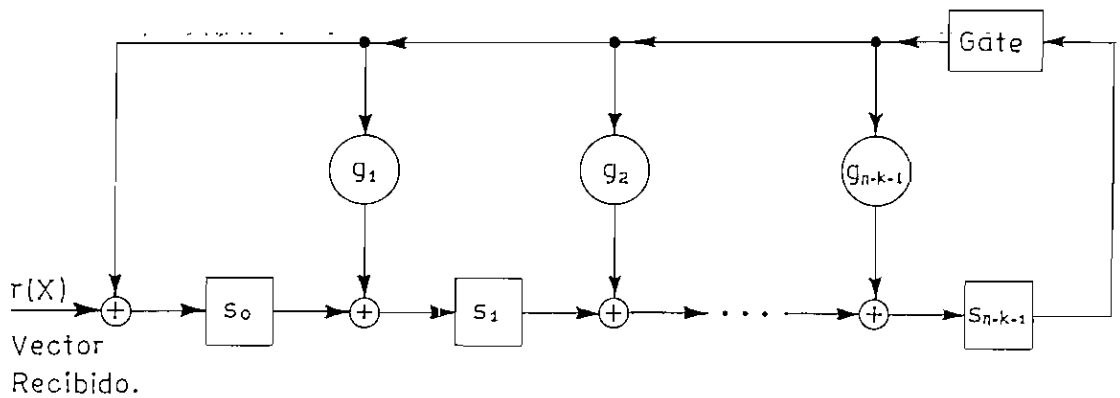


Fig.2.4 Circuito para calcular el Síndrome.
Entrada de r(X) por el lado izquierdo.

Teorema 2.3.1 Sea s(X) el síndrome de un polinomio recibido $r(X)= r_0 + r_1X + \ldots + r_{n-1}X^{n-1}$. Entonces, el residuo $s^{(1)}(X)$ resultante de dividir Xs(X) por el polinomio generador g(X) es el síndrome de $r^{(1)}(X)$, que es un desplazamiento ciclico de r(X).

Demostración: De acuerdo a (2.1) r(X) y $r^{(1)}(X)$ satisfacen la siguiente relación:

$$Xr(X)= r_{n-1}(X^n+1) + r^{(1)}(X)$$

que puede arreglarse como:

$$r^{(1)}(X)= r_{n-1}(X^n+1) + Xr(X)$$

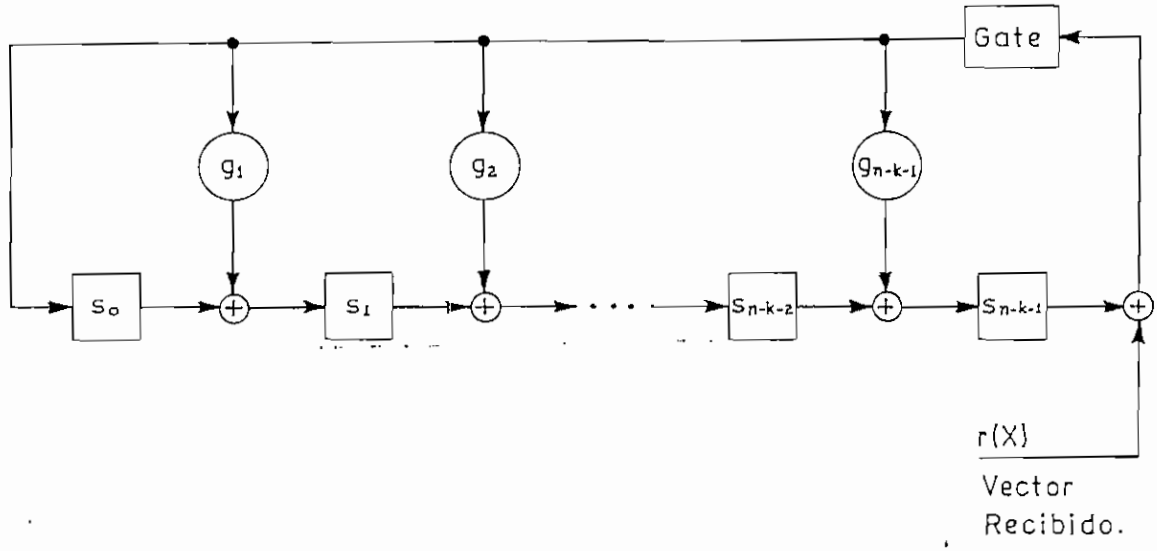


Fig.2.5.Circuito para calcular el Sindrome
Entrada de r(X) por la derecha

Ejemplo (2.6).- Cálculo del síndrome a partir del circuito de la fig. 2.4. para C (7,4), con r=(0111110).

Tabla 2.3

Desplazamiento	Entrada	Registro	
		0 0 0	Estado Inicial
1	0	0 0 0	
2	1	1 0 0	
3	1	1 1 0	
4	1	1 1 1	
5	1	0 0 1	
6	1	0 1 0	
7	0	0 0 1	Sindrome s
8	-	1 1 0	Sindrome $s^{(1)}$
9	-	0 1 1	Sindrome $s^{(2)}$

La tabla 2.3. indica el contenido de los registros para

los diferentes desplazamientos.

$s^{(1)}$ es el síndrome correspondiente a r desplazado una vez. es decir $r^{(1)}$ =(0011111).

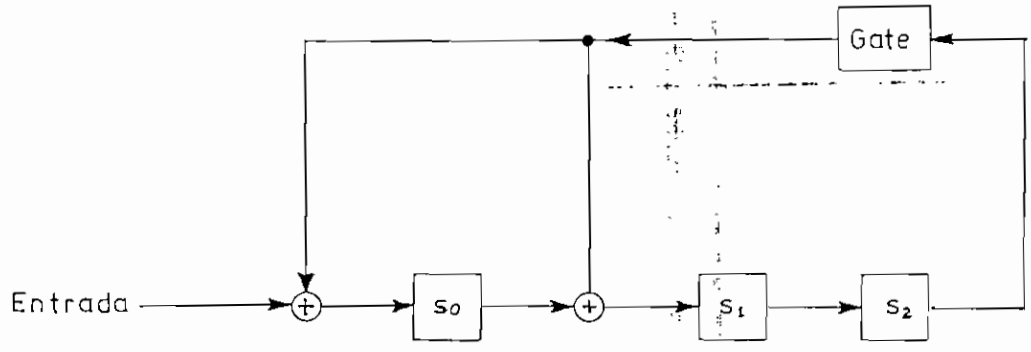


Fig.2.6 Circuito para el Síndrome de C (7,4)

2.3.1 Detección del Error en Códigos Cíclicos.

Si v(X) es la palabra código transmitida y $e(X)=e_0 + e_1X+.....+e_{n-1}X^{n-1}$ el modelo de error [6], el polinomio recibido es:

(2.21) $$r(X) = v(x)+e(X)$$

Puesto que v(X) es múltiplo de g(X) se establece la

"El vector suma e =r+v
$=(e_0, e_1, \ldots, e_{n-1})$
es una n_tupla donde $e_i=1$ para $r_i \neq v_i$ y $e_i=0$ para $r_i = v_i$. Esta n_tupla es llamada vector de error (modelo de error)."[6]

siguiente relación entre el error y el síndrome:

$$e(X) = [a(X)+b(X)]g(X) + s(X)$$

donde $v(X)=b(X)g(X)$.

El síndrome es el residuo de dividir el error por el polinomio generador.

El error e(X) es desconocido para el decodificador, pero puede ser estimado a partir del síndrome, que a su vez se calcula por medio del vector recibido.

Una revisión de los modelos de error se encuentra en el Arreglo Standard. Apéndice B.

El síndrome es cero si el modelo de error e(X) es cero. Si e(x) es idéntico a un vector de código, e(X) es no detectable [7].

Para detectar el error se coloca como entradas a una compuerta OR los dígitos provenientes del circuito del síndrome. si s(X) no es cero, la salida es 1 y la presencia de errores se ha detectado.

Los códigos cíclicos son muy efectivos para detectar errores de tipo aleatorio o ráfaga.

Otra forma de detectar el error consiste en dividir la

"Cualquier código cíclico de distancia de Hamming mínima H garantiza la detección de H-1 errores" [7]

palabra de código que llega al receptor por el polinomio generador, el residuo no es cero sólo cuando ésta no es válida. Así, el circuito es un simple divisor de la misma configuración que el usado en la codificación.

En un sistema de transmisión en dos sentidos el mismo circuito puede emplearse para la codificación y decodificación, lo que resulta más económico. Durante la decodificación, las salidas de las etapas del registro de desplazamiento que guardan el residuo de la división se llevan a un comparador binario que genera una señal "error detectado" si el resultado final es diferente de cero, la que puede utilizarse como "pedido de repetición" al transmisor.

La fig. 2.7 ilustra este decodificador.

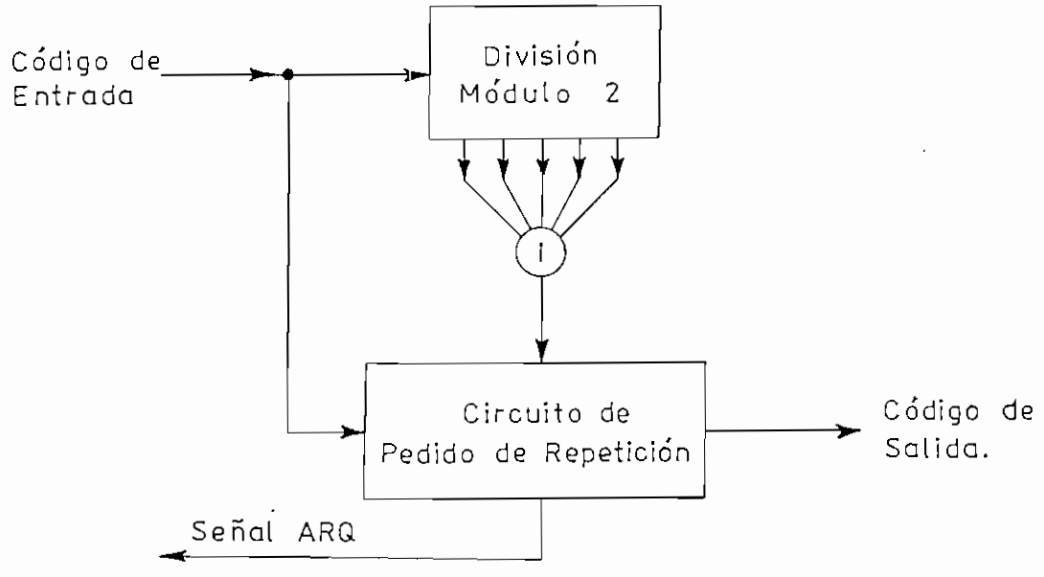


Fig. 2.7 Detector de errores

Una manera de simplificar la imolementación consiste en utilizar los bits de información de la palabra de código recibida para regenerar los bits de control de errores. Estos son comparados con los digitos de chequeo provenientes del codificador, si son idénticos se asume que no han ocurrido errores durante la transmisión.

2.4 Decodificación de Códigos Ciclicos.

La estructura de los códigos ciclicos permiten que un vector recibido r(X) se decodifique en forma serial.

Para la decodificación de los códigos cíclicos deben seguirse tres pasos que son: cálculo del síndrome, asociación del sindrome a un modelo de error y la corrección del error.

Como se vio anteriormente, para obtener el síndrome se utiliza un circuito de división. La asociación de éste a un modelo de error puede especificarse completamente con una tabla de decodificación [8]. Y la corrección del error consiste en sumar el modelo de error al vector recibido.

Luego del cálculo del síndrome, se chequea si éste corresponde a un modelo de error corregible [9], con un

---

"El conjunto de las secuencias de errores que aparecen en la tabla decodificadora, es precisamente el conjunto de errores que pueden ser corregidos"[8]

error en la posición de mayor orden X^{n-1} (e.d. $e_{n-1}=1$). Si esto no ocurre el polinomio recibido (almacenado en un buffer) y el registro del sindrome se desplazan ciclicamente una vez simultáneamente, asi se obtiene $r^{(1)}(X) = r_{n-1} + r_0X + \ldots + r_{n-2}X^{n-1}$ y el sindrome es $s^{(1)}(X)$.

Ahora, el segundo digito r_{n-2} de $r(X)$ es el primer digito de $r^{(1)}(X)$. Se verifica igual que para el bit anterior.

En el momento que el sindrome $s(X)$ corresponda a un modelo de error con $e_{n-1}=1$, el primer digito recibido r_{n-1} es erróneo y debe ser corregido. Lo que consiste en sumar $r_{n-1} + e_{n-1}$, quedando el polinomio recibido:

$$r_1(X) = r_0 + r_1X + \ldots + r_{n-2}X^{n-2} + (r_{n-1}+e_{n-1})X^{n-1}$$

El efecto del digito erróneo e_{n-1} sobre el sindrome es removido; ésto puede realizarse sumando el sindrome de $e'(X)= X^{n-1}$ a $s(X)$, resultado de lo cual se obtiene el sindrome de $r_1(X)$.

Si se desplaza $r_1(X)$ una vez se llega a:

$$r_1^{(1)}(X) = (r_{n-1} + e_{n-1}) + r_0X + \ldots + r_{n-2}X^{n-1}$$

"La decodificación es correcta si y sólo si el modelo de error causado por el canal es un coset leader. Por esta razón, los 2^{n-k} coset leaders (incluyendo el vector cero), son llamados modelos de error corregibles" [9]

El síndrome $s_1^{(1)}(X)$ es el residuo de dividir $X[s(X)+X^{n-1}]$ por $g(X)$, resultando:

(2.22) $$s_1^{(1)}(X) = s^{(1)}(X)+1$$

Así, si se suma 1 a la izquierda del registro del síndrome se obtiene $s_1^{(1)}(X)$.

Luego de ésto, se procede a decodificar el dígito recibido r_{n-2}, en forma idéntica a r_{n-1}.

En el decodificador luego de n desplazamientos, si e(X) es un tipo de error corregible el contenido del síndrome vuelve a cero, si ésto no ocurre es que se ha detectado un tipo de error no corregible.

Un decodificador general para un código cíclico (n,k) se presenta en la fig.2.8, es conocido como DECODER MEGGITT.

El funcionamiento del circuito se señala a continuación:

1. El síndrome se forma por el desplazamiento del vector recibido en el registro del síndrome, al mismo tiempo r(X) se almacena en el buffer.

2. El síndrome es leído en el detector y se examina el correspondiente modelo de error. El detector es un circuito combinacional tal que su salida es 1 si el síndrome corresponde a un error en la posición X^{n-1}, en este caso el dígito recibido en la etapa del extremo derecho es asumido

como erróneo y debe ser corregido.

3. El primer símbolo recibido es leído fuera del buffer y el registro del síndrome se desplaza una vez.
Cuando este dígito se detecta con error se corrige a la salida del detector y se realimenta al registro del síndrome para modificar su contenido. Este es un nuevo síndrome. y corresponde al vector recibido desplazado una vez a la derecha.

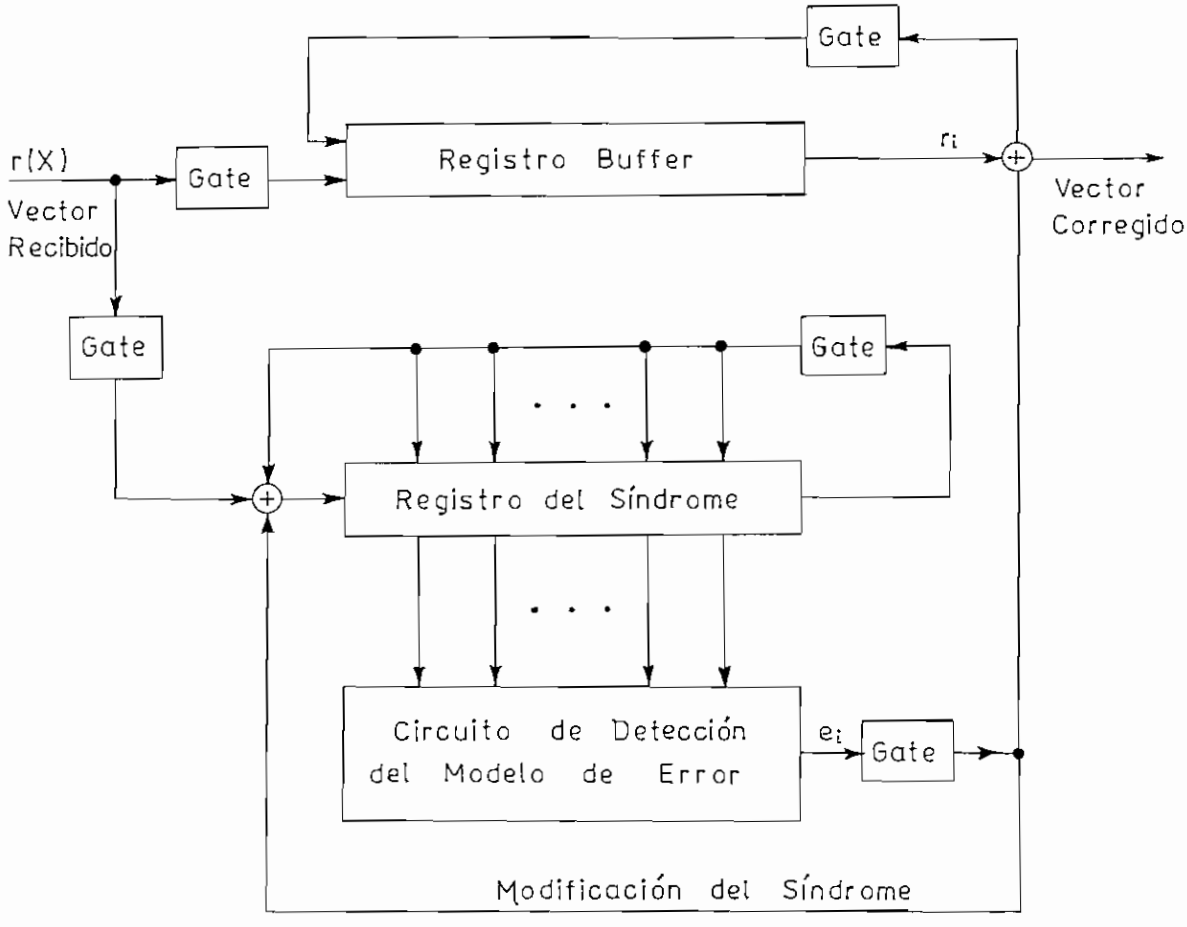


Fig.2.8 Decodificador para C (n,k)
DECODER MEGGITT

4. El nuevo síndrome del paso 3 es usado para detectar si el segundo símbolo es correcto. El decoder repite los pasos 2 y 3. Este dígito se corrige en caso de ser necesario en forma idéntica al primero.

5. Se decodifica el vector recibido por símbolo de la manera indicada hasta cuando el vector es leído completamente fuera del buffer.

A continuación se ilustra este procedimiento.

Ejemplo (2.7).- Para el código (7,4) la distancia mínima es 3 [10], y por tanto puede corregir errores simples [11] sobre un bloque de 7 dígitos.

Tabla 2.4 Tabla de Decodificación C (7,4)

e(X)	s(X)	(s_0, s_1, s_2)
$e(X)=X^6$	$s(X)=1+X^2$	(1 0 1)
$e(X)=X^5$	$s(X)=1+X+X^2$	(1 1 1)
$e(X)=X^4$	$s(X)=X+X^2$	(0 1 1)
$e(X)=X^3$	$s(X)=1+X$	(1 1 0)
$e(X)=X^2$	$s(X)=X^2$	(0 0 1)
$e(X)=X^1$	$s(X)=X$	(0 1 0)
$e(X)=X^0$	$s(X)=1$	(1 0 0)

Hay 7 modelos de error simple y junto con el vector cero forman todos los coset leaders de la tabla de decodificación.

"Si la distancia mínima de un código bloque C es dmin. cualesquiera dos vectores distintos de C difieren en al menos dmin lugares."[10]

El sindrome corresoondiente a cada vector es igual al residuo de dividir e(X) oor g(X).

La tabla 2.4 es la tabla de decodificación.

El circuito decodificador para el código (7.4) se presenta a continuación.

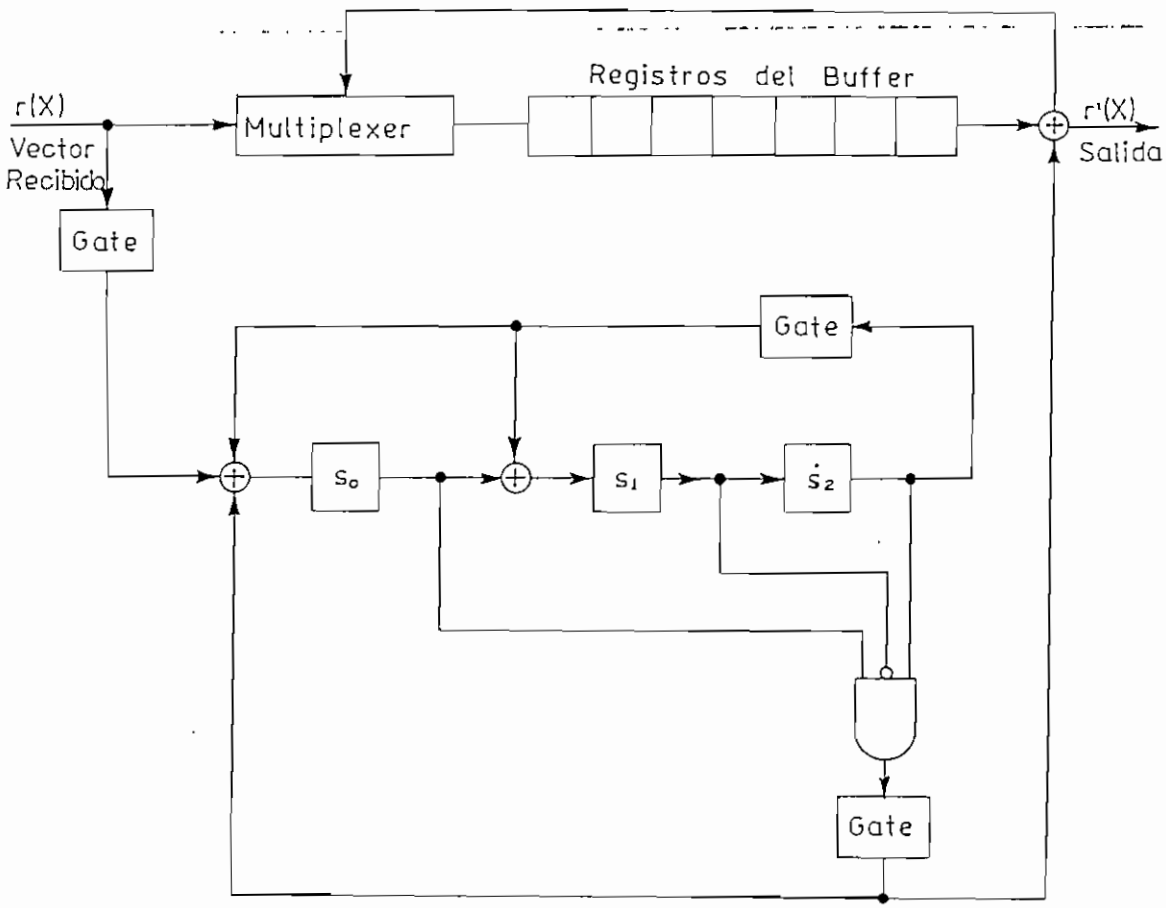


Fig.2.9 Circuito de Decodificación C (7,4)

"Un código bloque con distancia mínima dmin garantiza corregir todos los modelos de error de t=[(dmin-1)/2] o menos errores.
Con [] el más alto entero no mayor oue (dmin/2)" [11]

e_6 (X) $=X^6$ (error en la posición de mayor orden X^{n-1}) es el error simple en la posición X^6 con síndrome (1 0 1), cuando éste se detecta indica que hay un error y debe corregirse. Si el error está en X^i se requieren 6-i desplazamientos antes de obtener el síndrome correspondiente (1 0 1).

El único síndrome que debe detectarse es (1 0 1) y se utiliza una AND.

Si el vector transmitido es v=(0101110), y el recibido r=(1101110).

$$v(X)= X +X^3 +X^4 +X^5$$

$$r(X)= 1 +X +X^3 +X^4 +X^5$$

Se tiene un error simple en X^0 y para este vector recibido el registro del síndrome contiene inicialmente (1 0 0).

El proceso de decodificación para el ejemplo se presenta en la fig. 2.10.

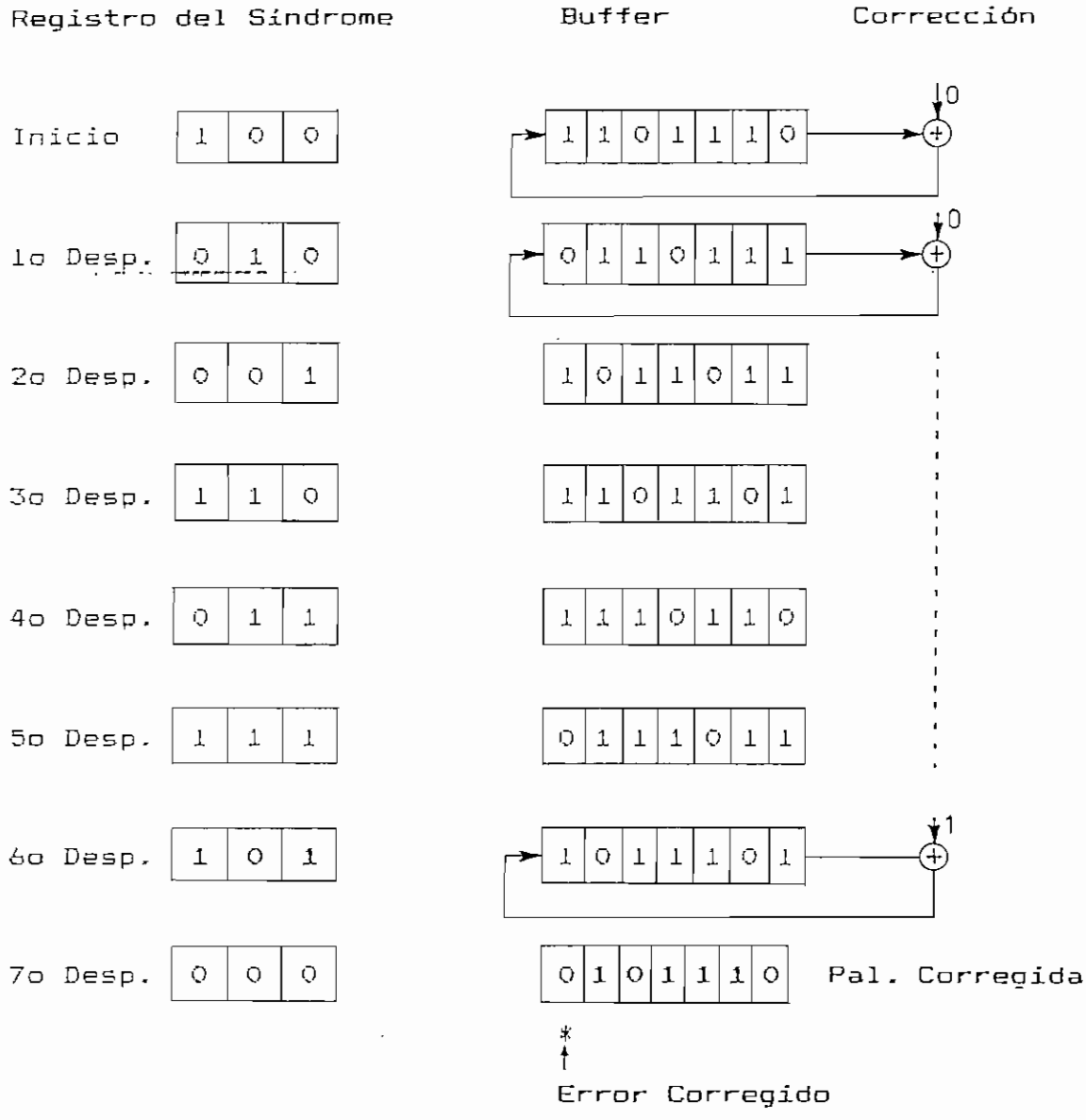


Fig.2.10 Proceso de Corrección de un Error

REFERENCIAS

1. Shu Lin y Costello, Error Control Coding:Fundamentals and Aplications, Prentice-Hall, New Jersey, 1983, Pg. 52.

2. Robert, Mc.Eliece, The Theory of Information and Coding, Pg. 148.

3. Gallager, Robert, Information Theory and Reliable Communications, John Wiley, New York, 1968, Pg. 216.

4. Egas, Carlos, Codificadores y Decodificadores para Códigos de Bloque Lineales, EPN, 1987, Pg. 26.

5. Egas, Carlos, Codificadores y Decodificadores para Códigos de Bloque Lineales, EPN, 1987, Pg. 31.

6. Shu Lin y Costello, Error Control Coding:Fundamentals and Aplications, Prentice-Hall, New Jersey, 1983, Pg. 58.

7. Coates, R., Modern Communication Systems, Macmillan, London, Pg. 233.

8. Gallager, Robert, Information Theory and Reliable Communications, John Wiley, New York, 1968, Pg. 203.

9. Shu Lin y Costello, Error Control Coding:Fundamentals and Aplications, Prentice-Hall, New Jersey, 1983, Pg. 70.

10. Shu Lin y Costello, Error Control Coding:Fundamentals and Aplications, Prentice-Hall, New Jersey, 1983, Pg. 65.

11. Shu Lin y Costello, Error Control Coding:Fundamentals and Aplications, Prentice-Hall, New Jersey, 1983, Pg. 67.

tiene como fundamento teórico las propiedades de los códigos cíclicos.

Si el mensaje se considera como un polinomio u(X), éste primeramente es multiplicado por X^r y luego dividido por el polinomio generador g(X) de grado r, de lo que resulta un cociente q(X) y un residuo r(X)/g(X). La aritmética se realiza en módulo-2.

Esto se indica en la siguiente ecuación:

$$\frac{X^r\, u(X)}{g(X)} = q(X) \oplus \frac{r(X)}{g(X)}$$

La adición y sustracción en módulo-2 son iguales, de tal manera que:

$$X^r\, u(X) \oplus r(X) = q(X)\, g(X)$$

donde r(X) es de grado menor a r.

El algoritmo del CRC calcula r(X) y lo añade al mensaje a enviarse. Puesto que $X^r\, u(X) \oplus r(X)$ es igual a q(X)g(X) el total de bytes transmitidos (información y control), son igualmente divisibles por g(X) si y sólo si los bits no han sido alterados.

En el receptor el mensaje original y r(X) se divide por el polinomio generador g(X). Si el resultado es cero se asume que no se han producido errores o que éstos no son detectables, en caso contrario se considera que hay un error.

longitud, los que se consultan durante el proceso, y

- Calculando los resultados intermedios del CRC para un byte dentro del algoritmo.

Los algoritmos señalados anteriormente se diferencian en la extensión de memoria ocupada y en la velocidad de detección de errores, que es lo que se verificará en la parte práctica de la tesis.

3.2 Algoritmo de Cálculo del CRC tomando el Mensaje Bit a Bit.

Puesto que la aritmética del CRC se hace en módulo-2, el cálculo es fácilmente realizable en hardware con registros de desplazamiento y compuertas OR-exclusivo como se muestra en la fig. 3.1 para el polinomio generador del CCITT. Cada flip-flop contiene un bit del registro del CRC: con este circuito se opera sobre un bit de mensaje cada vez.

El circuito satisface el procedimiento para obtener un código cíclico sistemático (2.2.1), que consiste en multiplicar el mensaje - para el polinomio del CCITT - por X^{16} y enviarlo al canal, luego realiza una división por $X^{16}+X^{12}+X^{5}+1$ resultado del cual se obtiene los dos bytes correspondientes al CRC, que se transmiten seguidamente a los bits de información.

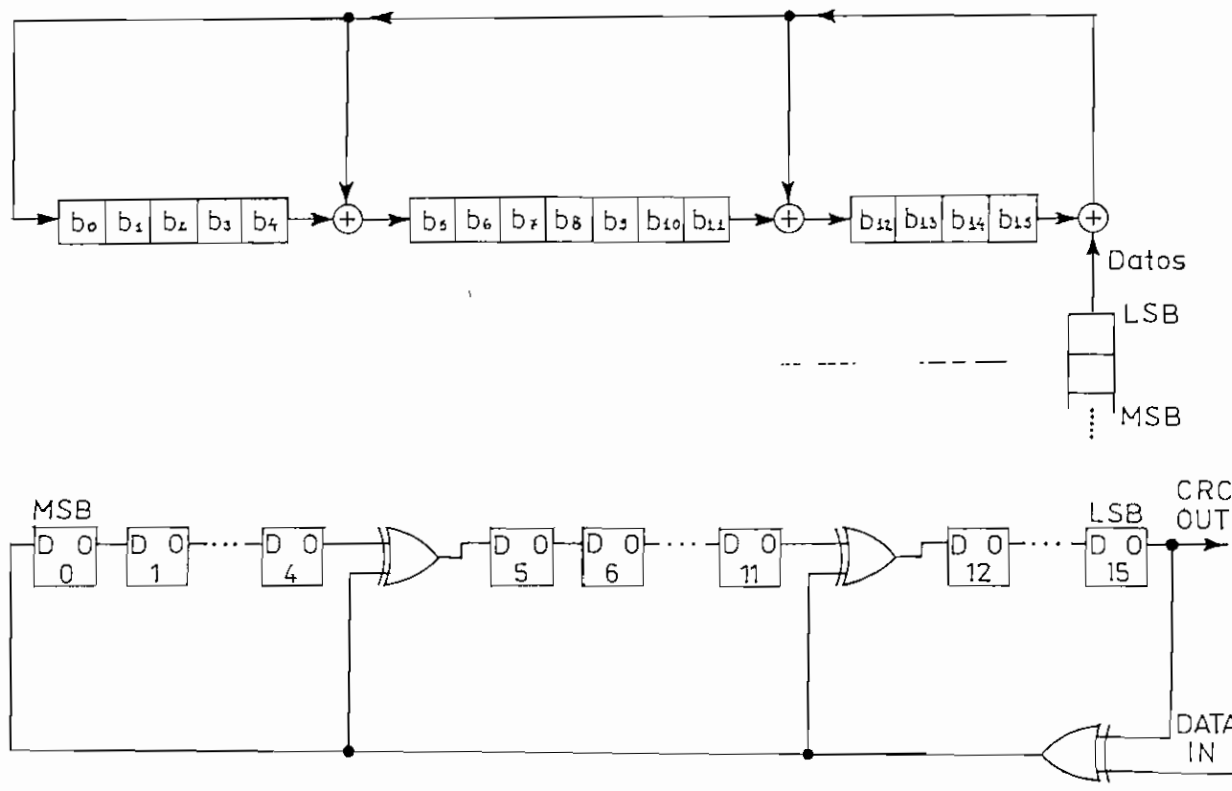


Fig. 3.1 Hardware para el Cálculo del CRC-CCITT

Para que el CRC conseguido de esta manera sea igual al que resulta al aplicar la División Euclidiana (de polinomios en módulo-2), el mensaje debe ingresar en sentido inverso, es decir primero el bit menos significativo.

3.2.1 Obtención del Algoritmo.

El algoritmo a obtener se basa en el polinomio del CCITT cuyo circuito en hardware es el de la fig.3.1. Las potencias determinan las ramas de realimentación y por tanto

Tabla 3.1

SH	IN	X^0	X^1	X^2	X^3	X^4	X^5	X^6	X^7	X^8	X^9	X^{10}	X^{11}	X^{12}	X^{13}	X^{14}	X^{15}
		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
0		c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
1	u_7	c_{15} u_7	c_0	c_1	c_2	c_3	c_4 c_{15} u_7	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11} c_{15} u_7	c_{12}	c_{13}	c_{14}
2	u_6	c_{14} u_6	c_{15} u_7	c_0	c_1	c_2	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5	c_6	c_7	c_8	c_9	c_{10} c_{14} u_6	c_{11} c_{15} u_7	c_{12}	c_{13}
3	u_5	c_{13} u_5	c_{14} u_6	c_{15} u_7	c_0	c_1	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5	c_6	c_7	c_8	c_9 c_{13} u_5	c_{10} c_{14} u_6	c_{11} c_{15} u_7	c_{12}

Algoritmo.-

1. Hacer un OR-exclusivo del bit de mensaje con el bit menos significativo del CRC.

2. Desplazar los registros del CRC un lugar a la derecha.

3. Colocar el resultado de la suma del punto 1 en cada posición de OR-exclusivo dado por las potencias del polinomio, es decir en X^0, X^3, X^{12}.

4. Repetir los pasos de 1 a 3 para cada bit de entrada.

Después de ingresar todos los bits del mensaje los registros del CRC contienen los dos bytes de control de errores.

Con este algoritmo el número de sumas en módulo-2 y desplazamientos crece conforme aumenta la longitud del

mensaje a transmitir.

3.2.2 Determinación del Valor Inicial para el Cálculo del CRC.

La Recomendación V.41 incluye el codificador y decodificador para el cálculo del CRC y la verificación de la presencia de errores. Estos son circuitos en hardware con registros de desplazamiento cuyo estado inicial es cero. La realización en software con la misma finalidad no se especifica; para su desarrollo debe determinarse el valor del CRC inicial y la forma en que se han de tratar los bits de información.

La DEC (Digital Equipment Corporation), en su VAX Architecture Handbook sugiere como valor inicial para el CRC-CCITT de polinomio generador $X^{16}+X^{12}+X^{5}+1$, -1 (FFFFh) complementando el resultado final del CRC; pero sin una justificación dentro de la citada referencia, por lo que es necesario utilizar los fundamentos teóricos de los códigos cíclicos para establecer si es conveniente.

El proceso de codificación, como se vio en el capítulo anterior y se describe en la Recomendación V.41, consiste en la división del mensaje por el polinomio generador para obtener el residuo que es precisamente el CRC.

Cualquier circuito en hardware o algoritmo para software debe dar igual resultado que al aplicar el método

de la división Euclidiana (división de polinomios en módulo-2).

A continuación se presenta un ejemplo de codificación aplicando el procedimiento de la división Euclidiana y el algoritmo propuesto en el item 3.2.1.

Ejemplo (3.1).-

Para $g(X) = X^{16} + X^{12} + X^5 + 1$ y el mensaje u=(00000001), $u(X)=X^7$.

Siguiendo los pasos para codificación sistemática (2.2.1) se tiene:

$$X^{16}\ (X^7)=X^{23}$$

$$\begin{array}{l|l} X^{23} & : X^{16}+X^{12}+X^5+1 \\ X^{23}\ +X^{19}\ +X^{12}\ +X^7 & :\text{----------} \\ \text{----------} & \\ X^{19}\ +X^{12}\ +X^7 & X^7+X^3 \\ X^{19}\ +X^{15}\ +X^8\ +X^3 & \\ \text{----------} & \\ X^{15}\ +X^{12}\ +X^8+X^7\ +X^3 & \end{array}$$

$b(X)= X^{15} + X^{12} + X^8 + X^7 + X^3$ b=(0001000110001001)= 1189h

$v(X)= X^{23} + X^{16} + X^{12} + X^8 + X^7 + X^3$ v=(0001000110001001 00000001)
CRC Mensaje

$v(X)=b(X) + X^{n-k}u(X)$ es polinomio de código y múltiplo de g(X) (Teorema 2.1.2.3); de acuerdo a ésto el procedimiento de verificación de errores de la Recomendación V.41, señala que en el receptor al dividir r(X) (mensaje recibido), por g(X) si no hay errores el resultado será cero (e.d. r(X)=v(X)).

Realizando la división del mensaje codificado por g(X).

se obtiene:

$$
\begin{array}{llllllll}
X^{23} & & +X^{16} & +X^{12} & +X^{8} & +X^{7} & +X^{3} & : \quad X^{16}+X^{12}+X^{5}+1 \\
X^{23} & +X^{17} & & +X^{12} & & +X^{7} & & :\text{----------}
\end{array}
$$

$$
\begin{array}{llllll}
\text{----------} & & & & & \\
 & X^{17} & +X^{16} & +X^{8} & +X^{3} & X^{7}+X^{3} \\
 & X^{17} & +X^{16} & +X^{8} & +X^{3} & \\
\text{----------} & & & & & \\
 & & & & 0 &
\end{array}
$$

En la última operación no hay residuo lo que indica que si se recibe este polinomio de código, es válido por ser múltiplo de g(X). Por tanto, el CRC añadido (1189h).es el correcto para el mensaje del ejemplo.

El valor del CRC inicial no está especificado; pudiendo ser 0 o -1, lo que se determinará calculando el CRC con el algoritmo para los dos casos, y comparando con el resultado obtenido en el procedimiento anterior.

a) CRC inicial= -1 (FFFFh)

Mensaje: u=(00000001)

El primer bit en ingresar es el menos significativo.

11111	1111111	1111	Dato	Suma $u_i + c_{LSB}$
01111	1111111	1111	1	0
10111	0111111	0111	0	1
11011	0011111	0011	0	1
11101	0001111	0001	0	1
11110	0000111	0000	0	1
01111	0000011	1000	0	0
00111	1000001	1100	0	0
00011	1100000	1110	0	0

CRC = 1E0Eh

b) CRC inicial=0

00000	0000000	0000	Dato	Suma $u_i + c_{LSB}$
10000	1000000	1000	1	1
01000	0100000	0100	0	0
00100	0010000	0010	0	0
00010	0001000	0001	0	0
10001	1000100	1000	0	1
01000	1100010	0100	0	0
00100	0110001	0010	0	0
00010	0011000	1001	0	0

CRC = 1189h

Según los resultados, se determina que el valor inicial del CRC es 0.

Se dispone de una referencia para todos los algoritmos utilizando la librería incluida en el compilador del lenguaje C, desarrollada por:

David Nienhiser 20-JUN.-1988

Copyright(C) 1988 Greenleaf Software Inc.

en la misma. se calcula el CRC para el mensaje que se almacena en un buffer cuyo tamaño se introduce como parámetro, además de indicar el valor inicial y los bytes de datos.

El programa se incluye en el Apéndice C.

Evaluando con éste, para V inicial= 0 y el Mensaje =1, se obtuvo como resultado CRC= 1189h.

3.3 Algoritmo Orientado a Byte utilizando la Tabla de los CRC de un Byte.

3.3.1 Cálculo del CRC a nivel de Bytes.

Operando a nivel de bytes se trata de lograr mayor

rapidez en el tratamiento del mensaje y la detección de posibles errores. En este caso el cálculo del CRC se lo realiza para un byte de ocho bits, por lo que se necesita un algoritmo que produzca igual resultado que el obtenido después de ocho desplazamientos en el método bit a bit.

Para derivar el algoritmo se utiliza la tabla 3.2: en la que se muestra el contenido de los registros del CRC para cada uno de los ocho pasos. La notación no se ha alterado.

En la tabla 3.2. se observa que el resultado de los registros del CRC después de ingresar un byte es una función (OR-exclusivo), de diferentes combinaciones de los bits del dato de entrada y los contenidos iniciales de dichos registros.

La propiedad conmutativa para la suma en módulo-2 señala:

$$A \oplus B = B \oplus A$$

con su aplicación se llega a la tabla 3.3.

Tabla 3.3

Contenidos del CRC Después de Ocho Desplazamientos

SH	IN	X^0	X^1	X^2	X^3	X^4	X^5	X^6	X^7	X^8	X^9	X^{10}	X^{11}	X^{12}	X^{13}	X^{14}	X^{15}
		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
8	u_0	c_8 u_0 c_{12} u_4	c_9 u_1 c_{13} u_5	c_{10} u_2 c_{14} u_6	c_{11} u_3 c_{15} u_7	c_{12} u_4	c_{13} u_5 c_8 u_0 c_{12} u_4	c_{14} u_6 c_9 u_1 c_{13} u_5	c_{15} u_7 c_{10} u_2 c_{14} u_6	c_0 c_{11} u_3 c_{15} u_7	c_1 c_{12} u_4	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7 c_8 u_0 c_{12} u_4	c_5 c_9 u_1 c_{13} u_5	c_6 c_{10} u_2 c_{14} u_6	c_7 c_{11} u_3 c_{15} u_7

Tabla 3.2

SH	IN	X^0	X^1	X^2	X^3	X^4	X^5	X^6	X^7	X^8	X^9	X^{10}	X^{11}	X^{12}	X^{13}	X^{14}	X^{15}
		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
0		c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
1	u_7	c_{15} u_7	c_0	c_1	c_2	c_3	c_4 c_{15} u_7	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11} c_{15} u_7	c_{12}	c_{13}	c_{14}
2	u_6	c_{14} u_6	c_{15} u_7	c_0	c_1	c_2	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5	c_6	c_7	c_8	c_9	c_{10} c_{14} u_6	c_{11} c_{15} u_7	c_{12}	c_{13}
3	u_5	c_{13} u_5	c_{14} u_6	c_{15} u_7	c_0	c_1	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5	c_6	c_7	c_8	c_9 c_{13} u_5	c_{10} c_{14} u_6	c_{11} c_{15} u_7	c_{12}
4	u_4	c_{12} u_4	c_{13} u_5	c_{14} u_6	c_{15} u_7	c_0	c_1 c_{12} u_4	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5	c_6	c_7	c_8 c_{12} u_4	c_9 c_{13} u_5	c_{10} c_{14} u_6	c_{11} c_{15} u_7
5	u_3	c_{11} c_{15} u_7 u_3	c_{12} u_4	c_{13} u_5	c_{14} u_6	c_{15} u_7	c_0 c_{11} c_{15} u_7 u_3	c_1 c_{12} u_4	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5	c_6	c_7 c_{11} c_{15} u_7 u_3	c_8 c_{12} u_4	c_9 c_{13} u_5	c_{10} c_{14} u_6
6	u_2	c_{10} c_{14} u_6 u_2	c_{11} c_{15} u_7 u_3	c_{12} u_4	c_{13} u_5	c_{14} u_6	c_{15} u_7 c_{10} c_{14} u_6 u_2	c_0 c_{11} c_{15} u_7 u_3	c_1 c_{12} u_4	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5	c_6 c_{10} c_{14} u_6 u_2	c_7 c_{11} c_{15} u_7 u_3	c_8 c_{12} u_4	c_9 c_{13} u_5
7	u_1	c_9 c_{13} u_5 u_1	c_{10} c_{14} u_6 u_2	c_{11} c_{15} u_7 u_3	c_{12} u_4	c_{13} u_5	c_{14} u_6 c_9 c_{13} u_5 u_1	c_{15} u_7 c_{10} c_{14} u_6 u_2	c_0 c_{11} c_{15} u_7 u_3	c_1 c_{12} u_4	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7	c_5 c_9 c_{13} u_5 u_1	c_6 c_{10} c_{14} u_6 u_2	c_7 c_{11} c_{15} u_7 u_3	c_8 c_{12} u_4
8	u_0	c_8 c_{12} u_4 u_0	c_9 c_{13} u_5 u_1	c_{10} c_{14} u_6 u_2	c_{11} c_{15} u_7 u_3	c_{12} u_4	c_{13} u_5 c_8 c_{12} u_4 u_0	c_{14} u_6 c_9 c_{13} u_5 u_1	c_{15} u_7 c_{10} c_{14} u_6 u_2	c_0 c_{11} c_{15} u_7 u_3	c_1 c_{12} u_4	c_2 c_{13} u_5	c_3 c_{14} u_6	c_4 c_{15} u_7 c_8 c_{12} u_4 u_0	c_5 c_9 c_{13} u_5 u_1	c_6 c_{10} c_{14} u_6 u_2	c_7 c_{11} c_{15} u_7 u_3

Definiendo un vector X_i compuesto por los X_i que resultan del OR-exclusivo del i-ésimo bit del byte de dato de entrada con el correspondiente bit del byte menos significativo del registro CRC, se tiene:

$$X = (X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7)$$

$$X_0 = c_8 \oplus u_0$$

$$X_1 = c_9 \oplus u_1$$

$$X_2 = c_{10} \oplus u_2$$

$$X_3 = c_{11} \oplus u_3$$

$$X_4 = c_{12} \oplus u_4$$

$$X_5 = c_{13} \oplus u_5$$

$$X_6 = c_{14} \oplus u_6$$

$$X_7 = c_{15} \oplus u_7$$

Para un byte del mensaje, X es el resultado del OR-exclusivo del byte de bajo orden del CRC con éste, por lo que se obtiene la simplificación de la tabla 3.4.

Tabla 3.4

SH	IN																
		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
8	u_0	X_0 X_4	X_1 X_5	X_2 X_6	X_3 X_7	X_4	X_5 X_0 X_4	X_6 X_1 X_5	X_7 X_2 X_6	c_0 X_3 X_7	c_1 X_4	c_2 X_5	c_3 X_6	c_4 X_7 X_0 X_4	c_5 X_1 X_5	c_6 X_2 X_6	c_7 X_3 X_7

3.3.2 Obtención del Algoritmo.

En la tabla 3.4 se observa:

- El byte de alto orden del CRC depende solamente de las combinaciones de OR-exclusivo del byte de bajo orden del CRC inicial y el byte de entrada.

- El byte de bajo orden del CRC depende de las funciones del byte bajo del CRC con el byte de datos y además de los ocho bits más altos del CRC inicial.

Esto lleva a concluir que es posible desplazar el byte alto del CRC hacia la posición del byte bajo del mismo y al hacer el OR-exclusivo con una palabra de 16 bits que se obtiene por combinación de los X_i, resulta el nuevo valor del CRC.

La tabla 3.5. indica el contenido final de los registros del CRC después de ocho desplazamientos.

Tabla 3.5 Contenido Final del CRC

SH	IN															
		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
		0	0	0	0	0	0	0	0	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
8	u_0	X_0 X_4	X_1 X_5	X_2 X_6	X_3 X_7	X_4	X_5 X_0 X_4	X_6 X_1 X_5	X_7 X_2 X_6	X_3 X_7	X_4	X_5	X_6	X_7 X_4 X_0	X_5 X_1	X_6 X_2	X_7 X_3

Algoritmo.-

1. Hacer el OR-exclusivo del byte del mensaje con el byte de bajo orden del CRC previo, obteniendo el vector X.

2. Desplazar el registro del CRC ocho bits a la derecha.

3. Calcular la combinación de los X_i que dan el valor de la palabra de 16 bits definido bajo la linea en la tabla 3.5.

4. OR-exclusivo del registro CRC (2) con el valor calculado en el paso 3.

5. Repetir los pasos 1 a 4 para cada byte del mensaje.

Debido a que el valor a obtenerse en el paso 3 depende solamente de X_0 a X_7, y hay únicamente 256 combinaciones posibles de éstos, el resultado puede ser tabulado con el vector X como índice. Asi, se llega al siguiente algoritmo:

1. Hacer el OR-exclusivo del byte de entrada con el byte de bajo orden del CRC previo, obteniendo el vector X.

2. Desplazar el registro del CRC ocho bits a la derecha.

3. OR-exclusivo del registro del CRC (2) con el contenido de la tabla, utilizando X como índice.

4. Repetir los pasos 1 a 3 para cada byte del mensaje.

Para el polinomio del CRC-CCITT se presenta en la tabla 3.6

el CRC de los posibles mensajes de un byte de longitud.

Para el procedimiento de cálculo bit a bit el mensaje debe introducirse en el orden inverso, es decir comenzando con el bit menos significativo; en el método último los bytes deben tomarse en la misma forma que se desean enviar.

Con la finalidad de ilustrar los dos algoritmos se presenta como ejemplo el cálculo del CRC para un mensaje de 3 bytes de longitud.

Ejemplo (3.2).-

Mensaje: 0,1,3

Mensaje en hexadecimal: 00h,01h,03h

a) Algoritmo de cálculo bit a bit.

Datos: 00000000,00000001,00000011

00000	0000000	0000	Dato	Suma u_i+C_{LSB}
00000	0000000	0000	0	0
00000	0000000	0000	0	0
00000	0000000	0000	0	0
00000	0000000	0000	0	0
00000	0000000	0000	0	0
00000	0000000	0000	0	0
00000	0000000	0000	0	0
00000	0000000	0000	0	0

CRC_1=0000h

10000	1000000	1000	1	1
01000	0100000	0100	0	0
00100	0010000	0010	0	0
00010	0001000	0001	0	0
10001	1000100	1000	0	1
01000	1100010	0100	0	0
00100	0110001	0010	0	0
00010	0011000	1001	0	0

CRC_2=1189h

00001	0001100	0100	1	0
10000	0000110	1010	1	1
01000	0000011	0101	0	0
10100	1000001	0010	0	1
01010	0100000	1001	0	0
10101	1010000	1100	0	1
01010	1101000	0110	0	0
00101	0110100	0011	0	0

CRC_3=2B43h

b) Algoritmo de cálculo por bytes.
Datos: 00000000,00000001,00000011

CRC Inicial= 0000h

```
0000  0000  0000  0000   0000h
XOR         0000  0000     00h
-----------------------------
0000  0000  0000  0000   00h=0d Indice
```

Tab (0+1)= 0000h

```
0000  0000  0000  0000   0000h
XOR         0000  0000     00h
-----------------------------
0000  0000  0000  0000   CRC1 = 0000h
```

```
0000  0000  0000  0000   0000h
XOR         0000  0001     01h
-----------------------------
0000  0000  0000  0001   01h=1d Indice
```

Tab (1+1)= 1189h

```
0001  0001  1000  1001   1189h
XOR         0000  0000     00h
-----------------------------
0001  0001  1000  1001   CRC2 = 1189h
```

```
0001  0001  1000  1001    1189h
XOR         0000  0011      03h
-----------------------------
0001  0001  1000  1010   8Ah=138d Indice
```

Tab (138+1)= 2B52h

```
0010  1011  0101  0010   2B52h
XOR         0001  0001     11h
-----------------------------
0010  1011  0100  0011    CRC3 = 2B43h
```

Los valores del CRC para un byte se obtuvieron en la tabla 3.6.

Tabla 3.6

TABLA DEL CRC PARA MENSAJES DE UN BYTE
POLINOMIO GENERADOR: $X^{16} + X^{12} + X^{3} + 1$

X	0	1	2	3	4	5	6	7
0	0000h	1189h	2312h	329Bh	4624h	57ADh	6536h	74BFh
8	8C48h	9DC1h	AF5Ah	BED3h	CA6Ch	DBE5h	E97Eh	F8F7h
16	1081h	0108h	3393h	221Ah	56A5h	472Ch	75B7h	643Eh
24	9CC9h	8D40h	BFDBh	AE52h	DAEDh	CB64h	F9FFh	E876h
32	2102h	308Bh	0210h	1399h	6726h	76AFh	4434h	55BDh
40	AD4Ah	BCC3h	8E58h	9FD1h	EB6Eh	FAE7h	C87Ch	D9F5h
48	3183h	200Ah	1291h	0318h	77A7h	662Eh	54B5h	453Ch
56	BDCBh	AC42h	9ED9h	8F50h	FBEFh	EA66h	D8FDh	C974h
64	4204h	538Dh	6116h	709Fh	0420h	15A9h	2732h	36BBh
72	CE4Ch	DFC5h	ED5Eh	FCD7h	8868h	99E1h	AB7Ah	BAF3h
80	5285h	430Ch	7197h	601Eh	14A1h	0528h	37B3h	263Ah
88	DECDh	CF44h	FDDFh	EC56h	98E9h	8960h	BBFBh	AA72h
96	6306h	728Fh	4014h	519Dh	2522h	34ABh	0630h	17B9h
104	EF4Eh	FEC7h	CC5Ch	DDD5h	A96Ah	B8E3h	8A78h	9BF1h
112	7387h	620Eh	5095h	411Ch	35A3h	242Ah	16B1h	0738h
120	FFCFh	EE46h	DCDDh	CD54h	B9EBh	A862h	9AF9h	8B70h
128	8408h	9581h	A71Ah	B693h	C22Ch	D3A5h	E13Eh	F0B7h
136	0840h	19C9h	2B52h	3ADBh	4E64h	5FEDh	6D76h	7CFFh
144	9489h	8500h	B79Bh	A612h	D2ADh	C324h	F1BFh	E036h
152	18C1h	094Bh	3BD3h	2A5Ah	5EE5h	4F6Ch	7DF7h	6C7Eh
160	A50Ah	B483h	8618h	9791h	E32Eh	F2A7h	C03Ch	D1B5h
168	2942h	38CBh	0A50h	1BD9h	6F66h	7EEFh	4C74h	5DFDh
176	B58Bh	A402h	9699h	8710h	F3AFh	E226h	D0BDh	C134h
184	39C3h	284Ah	1AD1h	0B58h	7FE7h	6E6Eh	5CF5h	4D7Ch
192	C60Ch	D785h	E51Eh	F497h	8028h	91A1h	A33Ah	B2B3h
200	4A44h	5BCDh	6956h	78DFh	0C60h	1DE9h	2F72h	3EFBh
208	D68Dh	C704h	F59Fh	E416h	90A9h	8120h	B3BBh	A232h
216	5AC5h	4B4Ch	79D7h	685Eh	1CE1h	0D68h	3FF3h	2E7Ah
224	E70Eh	F687h	C41Ch	D595h	A12Ah	B0A3h	8238h	93B1h
232	6B46h	7ACFh	4854h	59DDh	2D62h	3CEBh	0E70h	1FF9h
240	F78Fh	E606h	D49Dh	C514h	B1ABh	A022h	92B9h	8330h
248	7BC7h	6A4Eh	58D5h	495Ch	3DE3h	2C6Ah	1EF1h	0F78h

3.4 Algoritmo Orientado a Byte calculando el CRC de un Byte en el Proceso.

En el algoritmo anterior de cálculo del CRC con búsqueda en una tabla (CRC de los 256 valores posibles de X), se requiere tenerla a disposición permanentemente. Sin embargo, en la tabla 3.5 se observa que la palabra de 16 bits es una combinación fija de los X_i que depende únicamente del polinomio generador, pudiendo calcularse a partir del vector X dentro del procedimiento.

Haciendo la modificación correspondiente resulta el siguiente algoritmo para el cálculo a nivel de bytes:

1. Hacer el OR-exclusivo del byte de entrada con el byte bajo del CRC previo, obteniendo el vector X ($X_0 - X_7$).
2. Cálculo del CRC de un byte por combinación de los X_i del paso 1.
3. Desplazamiento del registro del CRC ocho bits a la derecha.
4. OR-exclusivo de los resultados de 2 y 3.
5. Repetir los pàsos 1 a 4 para cada byte del mensaje.

Este algoritmo en adelante se va a distinguir con el nombre de cálculo del CRC por bytes "on the fly".

En la parte práctica de la tesis se desarrollan los algoritmos estudiados en este capítulo, para compararlos.

CAPITULO IV

IMPLEMENTACION PRACTICA DE LOS ALGORITMOS EN TRANSMISION DE DATOS

4.1 Diagrama de Flujo de los Algoritmos de Cálculo del CRC.

Para obtener los dos bytes del CRC corresoondientes a un mensaje se aplican los siguientes algoritmos de cálculo:

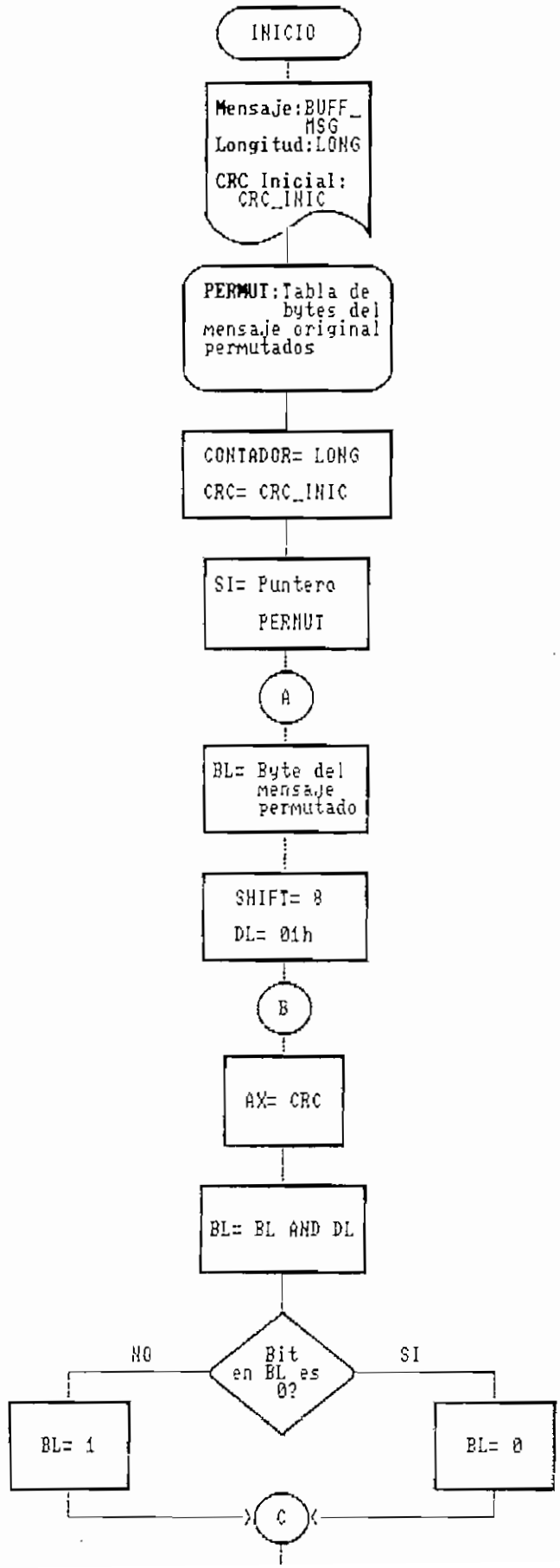
1) Bit a bit.

2) Por byte con búsqueda del CRC de un byte en la tabla que contiene todos los CRC para longitud uno.

3) Por byte con cálculo del CRC para longitud uno en el proceso ("on the fly").

El diagrama de flujo de cada uno de estos métodos se presenta a continuación.

FIG. 4.1. DIAGRAMA DE FLUJO DEL CALCULO DEL CRC BIT A BIT



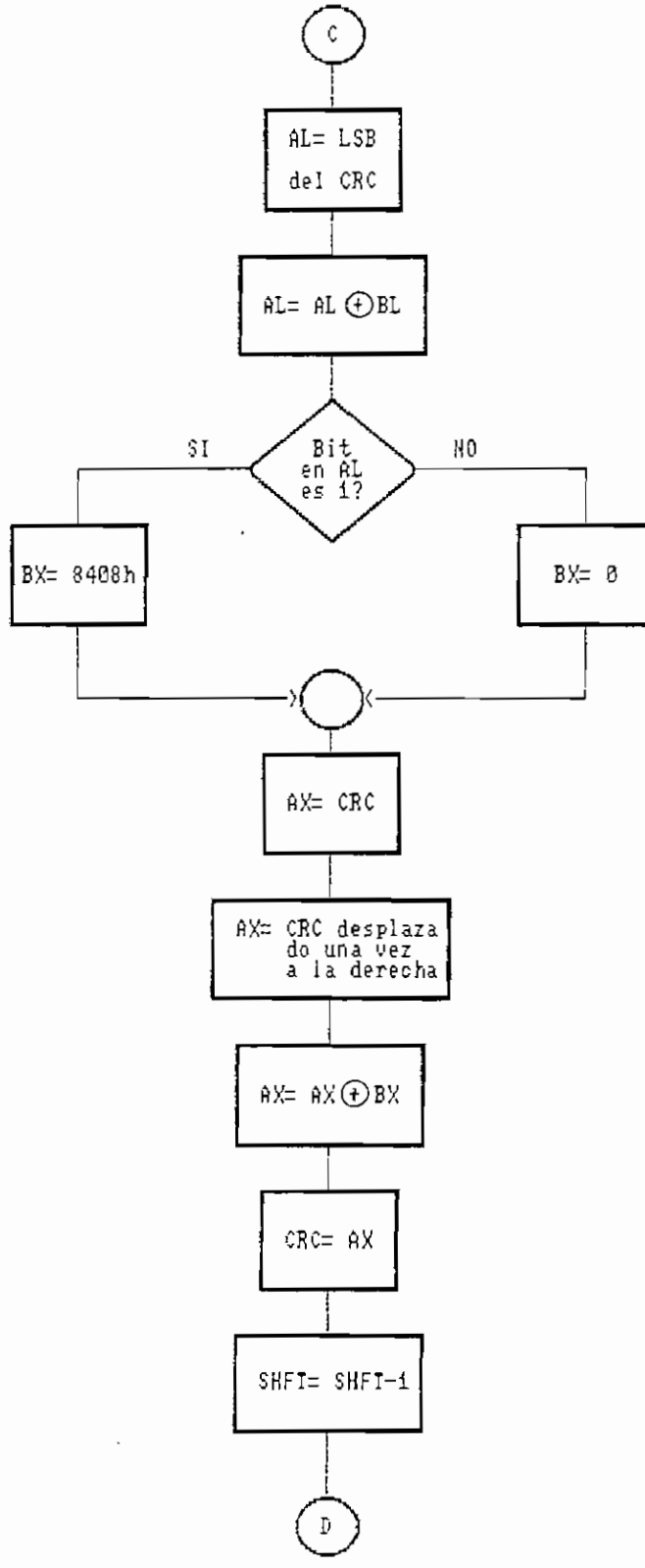


FIG. 4.1 CONTINUACION

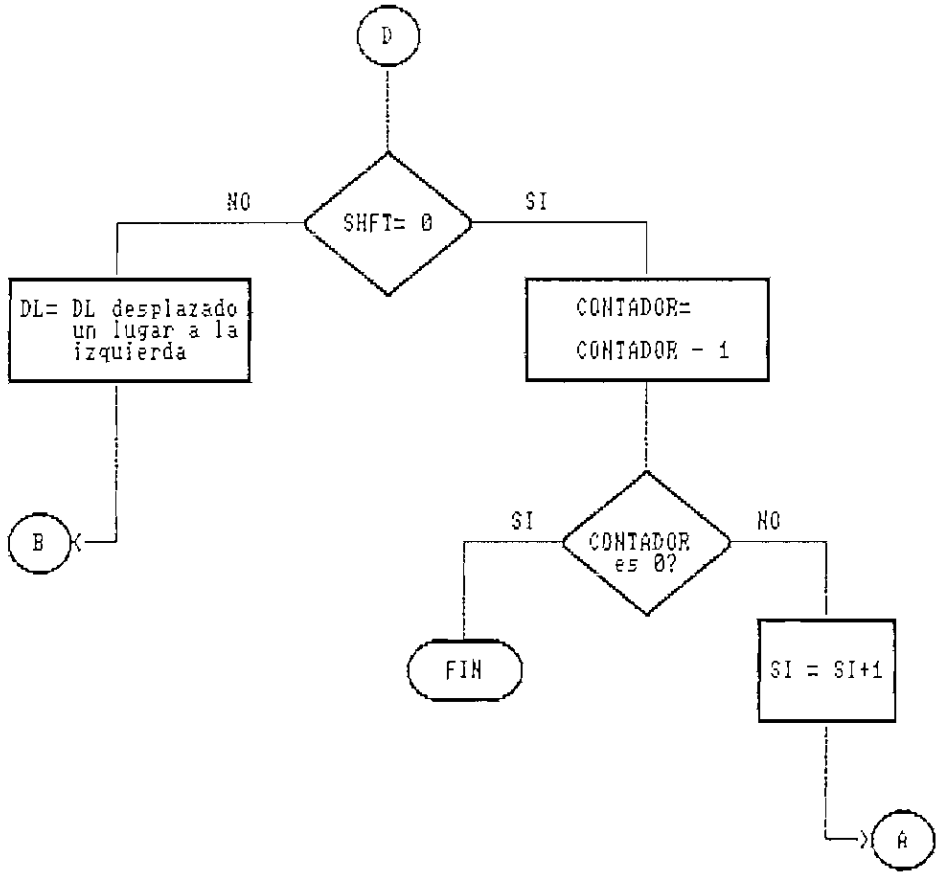
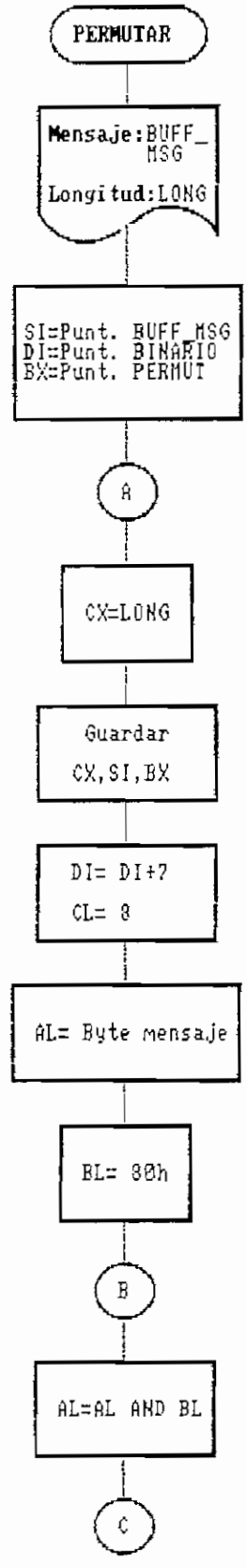


FIG. 4.1 CONTINUACION

FIG. 4.2. DIAGRAMA DE FLUJO DE LA PERMUTACION DEL MENSAJE



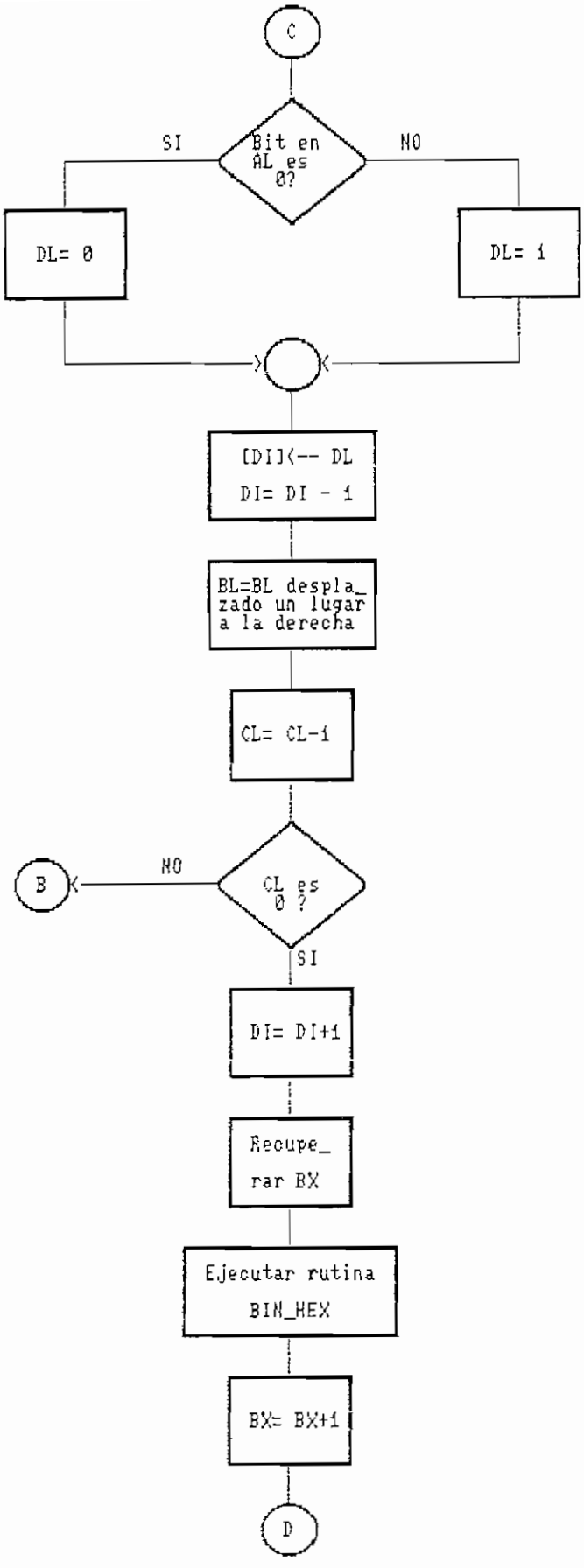


FIG. 4.2 CONTINUACION

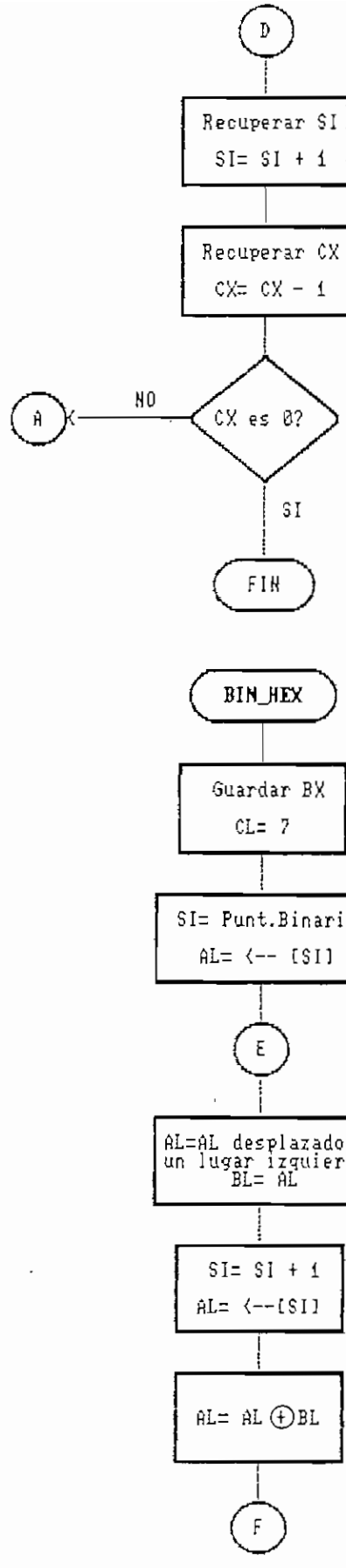


FIG. 4.2 CONTINUACION

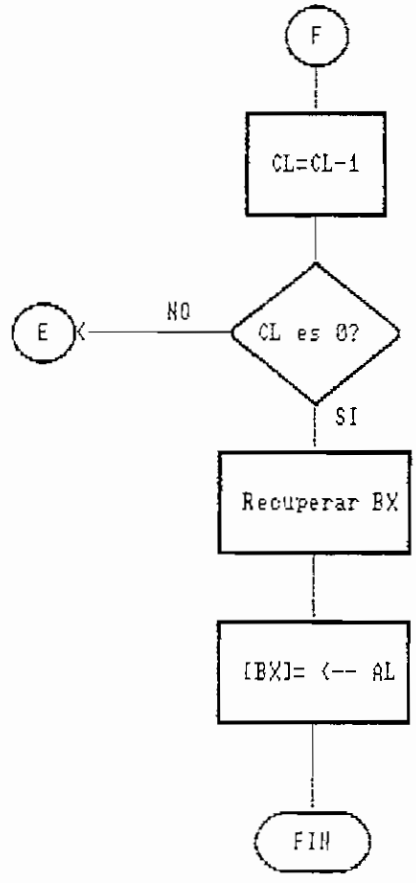
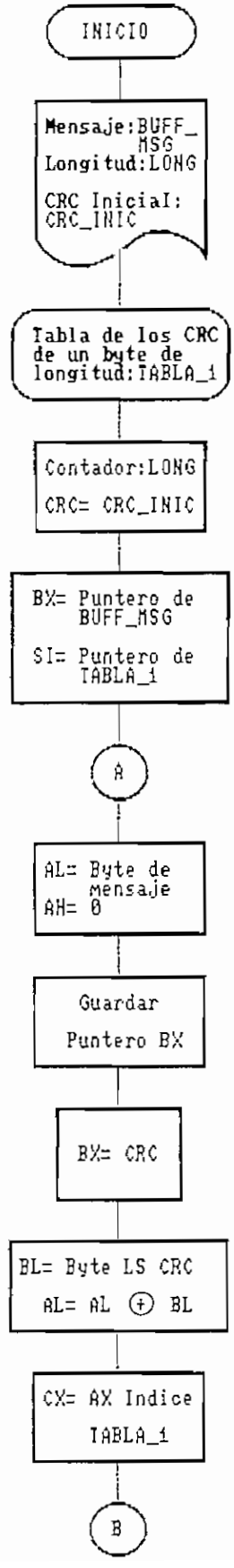


FIG. 4.2 CONTINUACION

FIG. 4.3. DIAGRAMA DE FLUJO DEL CALCULO DEL CRC CON BUSQUEDA EN UNA TABLA



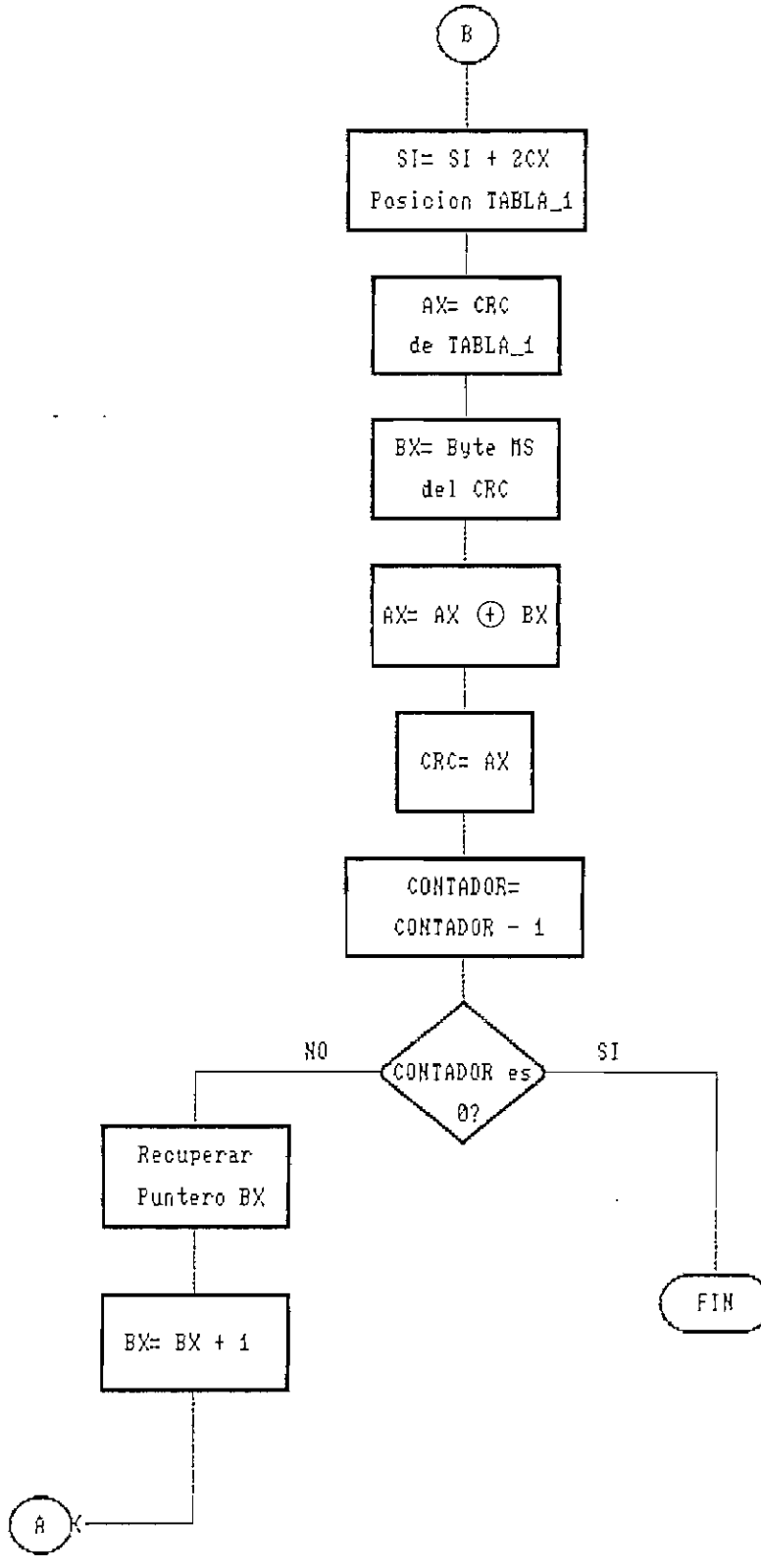


FIG. 4.3 CONTINUACION

FIG. 4.4. DIAGRAMA DE FLUJO DEL CALCULO DE LA TABLA DE CRC DE UN BYTE

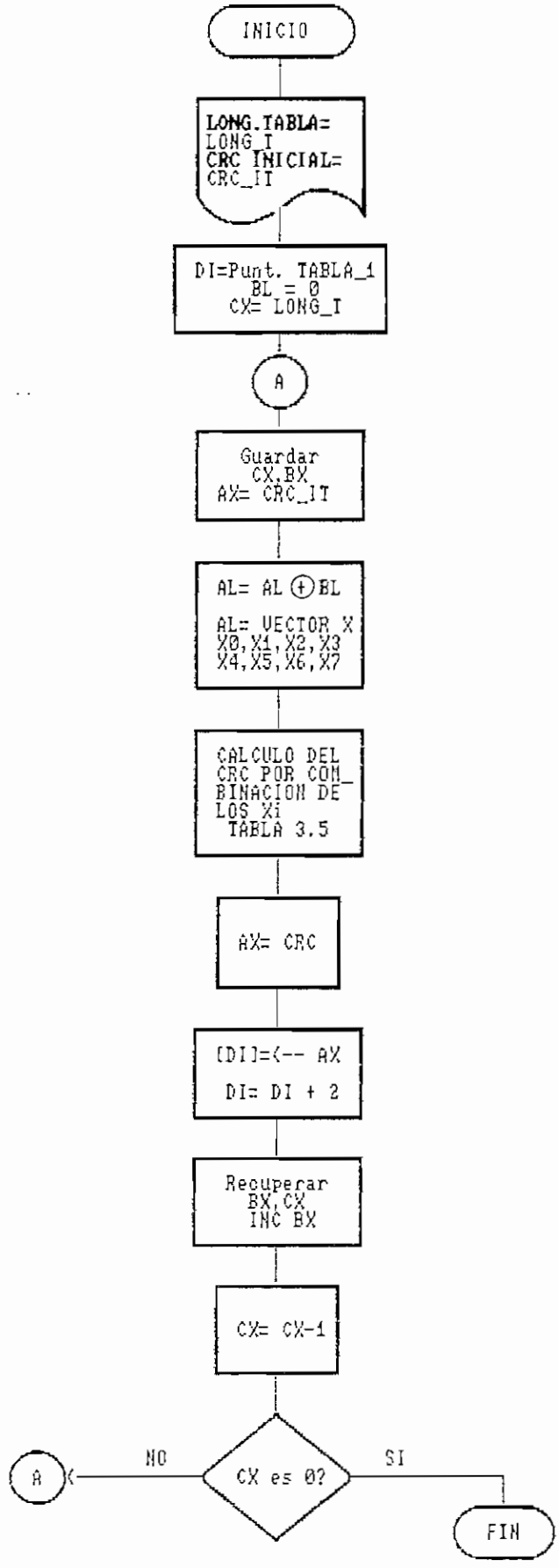
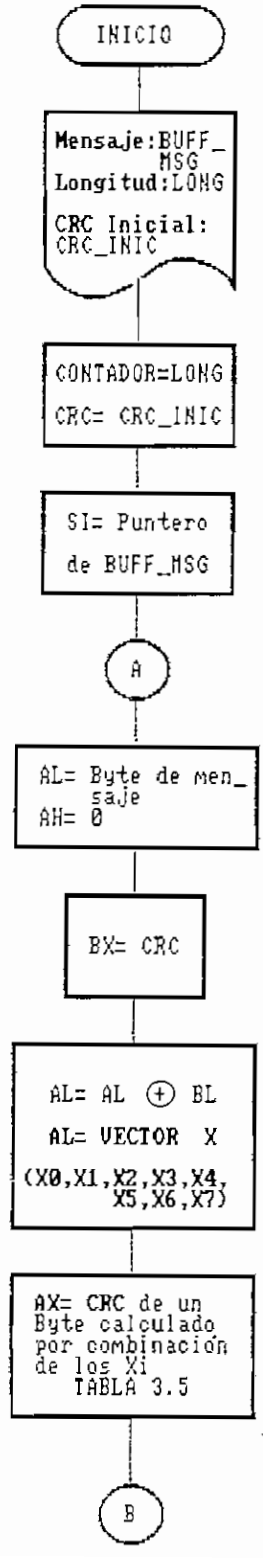


FIG. 4.5. DIAGRAMA DE FLUJO DEL CALCULO DEL CRC "ON THE FLY"



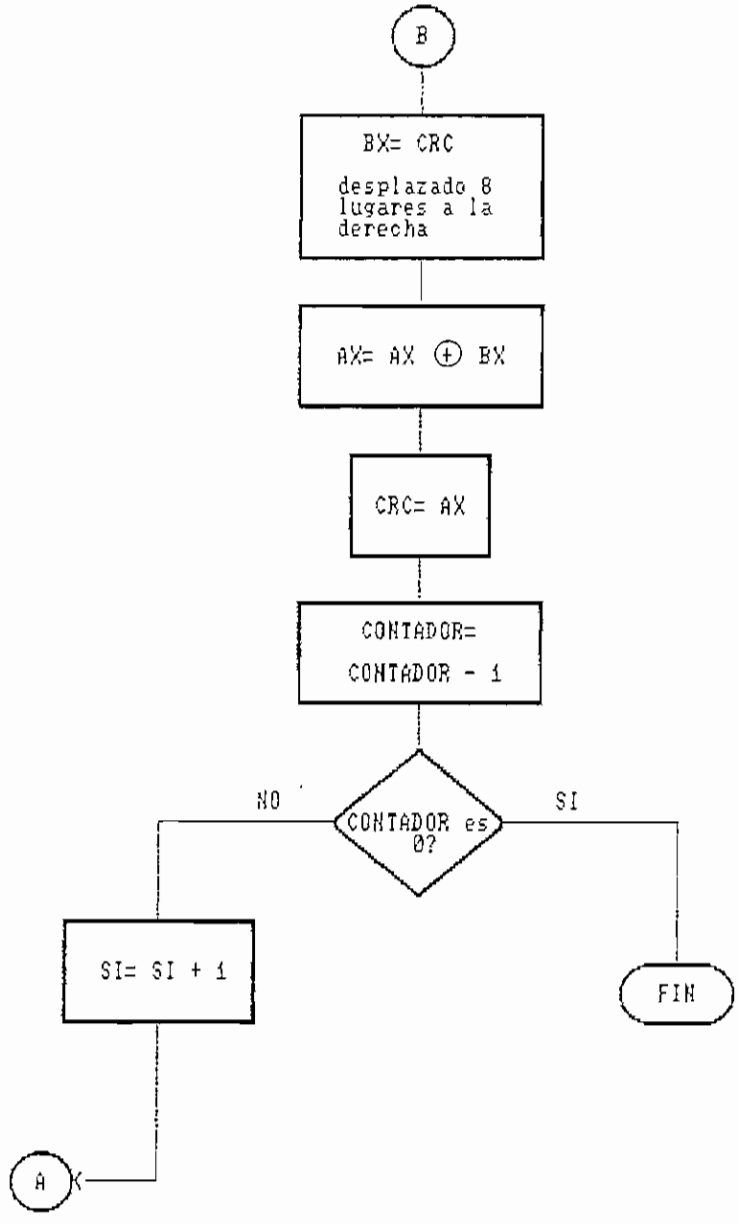
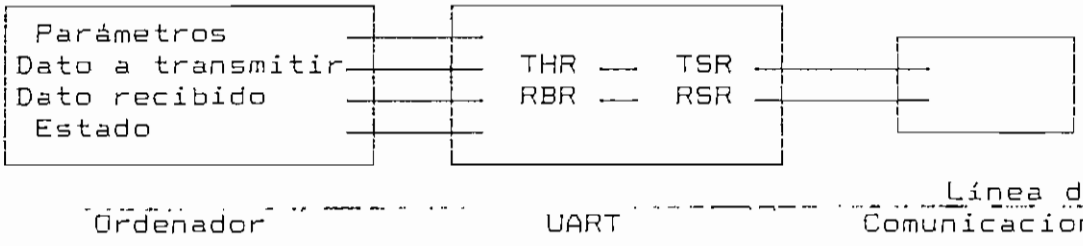


FIG. 4.5 CONTINUACION

buffer register)

El esquema de funcionamiento del UART es el siguiente:



THR (Transmitter Holding Register), registro temporal de entrada
TSR (Transmitter Shift Register), registro de salida
RBR (Receiver Buffer Register), registro de entrada
RSR (Receiver Shift Register), registro temporal de entrada

Los datos salientes se almacenan en el registro de desplazamiento de transmisión (THR). El bit menos significativo de este registro está unido a la línea serie de salida y el registro desplaza hacia la derecha, un bit cada vez según la razón en baudios. Los bits de start y stop se añaden según se precisen.

Los datos que entran son desplazados en el registro de de recepción (RSR), hasta que se acumulan los bits del carácter que luego es llevado al registro buffer de recepción (RDR).

El UART puede programarse en lenguajes de bajo y alto nivel como por ejemplo. en lenguaje ensamblador, en C, o en BASIC, con la finalidad de realizar la comunicación entre

terminales.

4.2.2.1 El Interfaz Serie RS-232C.- Es un interfaz eléctrico estándar establecido por Electronic Industries Association (EIA), para la interconexión de modems, impresoras y computadoras.

El RS-232C define un camino de señal de 25 conductores que conforma 18 circuitos con retorno a través de tierra. Define también los voltajes - los rangos de 0 y 1 lógicos - usados en los circuitos.

El IBM PC utiliza como máximo 9 conductores, los que se presentan en la tabla 4.1.

A menudo se utilizan sólo 3 conductores para la conexión simple entre dos computadores, llamada de modem nulo, éstos son el de transmisión TxD (2), de RxD (3) y el de tierra (7).

En la parte práctica de la tesis se utiliza esta forma de conexión para la transmisión entre los dos computadores.

Aunque el estándar RS-232C especifica el número de los pines, no especifica el conector. Este normalmente se referencia como un conector DB-25 y está disponible en los géneros macho y hembra, el DB-25P y el DB-25S respectivamente.

Tabla 4.1

Definición de los Pines del RS-232C

Pines	Señal	Descripción
2	TxD	Salida de datos del PC
3	RxD	Entrada de datos al PC
4	RTS	Request to Send. El PC quiere emitir datos
5	CTS	Clear to Send. Al PC, cuando el aparato está listo para recibir datos
6	DSR	Data Set Ready. Al PC cuando el modem está encendido y conectado
7	GND	Señal de tierra
8	CD	Carrier Detect. Al PC cuando el modem detecta portadora
20	DTR	Data Terminal Ready. Del PC siempre que la comunicación está activa
22	RI	Ring Indicator. Al PC cuando el modem recibe un timbre

4.3 Diagrama de Flujo de la Transmisión de Datos.

4.3.1 Descripción del Programa.

Se ha desarrollado un programa que permita observar los diferentes pasos a seguir para la transmisión de datos con control de errores aplicando el CRC.

La secuencia de trabajo, controlada a través del programa principal se va señalando por medio de las opciones resaltadas en el menú, para seguirla solamente debe presionarse la tecla de la letra subrayada.

El programa puede ser utilizado en dos modos:

transmisión y recepción. La transmisión de mensajes entre los dos computadores es simplex, de manera que uno de ellos debe iniciarse para transmitir y el otro para recibir.

TRANSMISION.- Primeramente debe especificarse el nombre del archivo en el cual se va a grabar el mensaje, puede señalarse también el directorio que lo contiene o en el que se va a crear si es nuevo; al nombre se añade la extensión .msg.

Los mensajes a enviar son de texto, máximo de 8 Kbytes de longitud, y para escribirlos se utiliza el Norton Editor que debe acompañar al programa CRC.EXE; cuando se ha terminado de editar se utiliza la función del editor "grabar y salir" para continuar con el procedimiento.

Se lee el archivo de mensaje, si su longitud es menor a 1024 bytes se considera como un solo bloque, pero si es mayor se divide en bloques de 1024 bytes. El CRC se calcula para cada bloque.

Por medio de una rutina se obtiene el tiempo requerido por los diferentes métodos para el cálculo del CRC del bloque número uno del mensaje, ésto nos permite comparar su velocidad.

En este nivel se dispone de dos opciones que son transmisión del mensaje e introducción de errores.

Si se desea introducir errores, ésto es, hacer una simulación de los que podrían producirse en la línea, el programa crea un archivo con el nombre anterior y la extensión -e.msg. y se ingresa nuevamente al editor para modificar el mensaje original en cualquier bloque con la función "sobreescribir". Para continuar se debe emplear la función "grabar y salir".

Luego de introducir errores o en el nivel anterior a éste se tiene la opción de transmisión, para lo que se han definido los siguientes parámetros:

Velocidad: 9600 bps
Paridad: Ninguna
Número de bits de datos: 8
Número de bits de parada: 1

Si no varía el mensaje se dispone únicamente del archivo nombre.msg, cuando se escoge introducir errores se tiene además el archivo nombre-e.msg, que son los últimos archivos listos para transmitirse en cada caso.

Al escoger la transmisión se procede a enviar un bloque de información del último archivo .msg con el respectivo CRC y se espera el acuse de recibo del computador remoto, si es positivo envía el siguiente bloque con su CRC. Pero si es negativo se retransmite el bloque con el mismo número esta vez del archivo de mensaje sin errores (nombre.msg), se espera el acuse de recibo y si es positivo se transmite el bloque siguiente. Después de recibir el acuse positivo del

último bloque de datos se transmite el carácter de fin de transmisión.

De esta manera se aplica el método de retransmisión para y espera.

RECEPCION.- En este modo se pide ingresar el directorio y nombre del archivo en el que se va a grabar el mensaje a recibirse. la extensión que se colocará en este caso es .rcp.

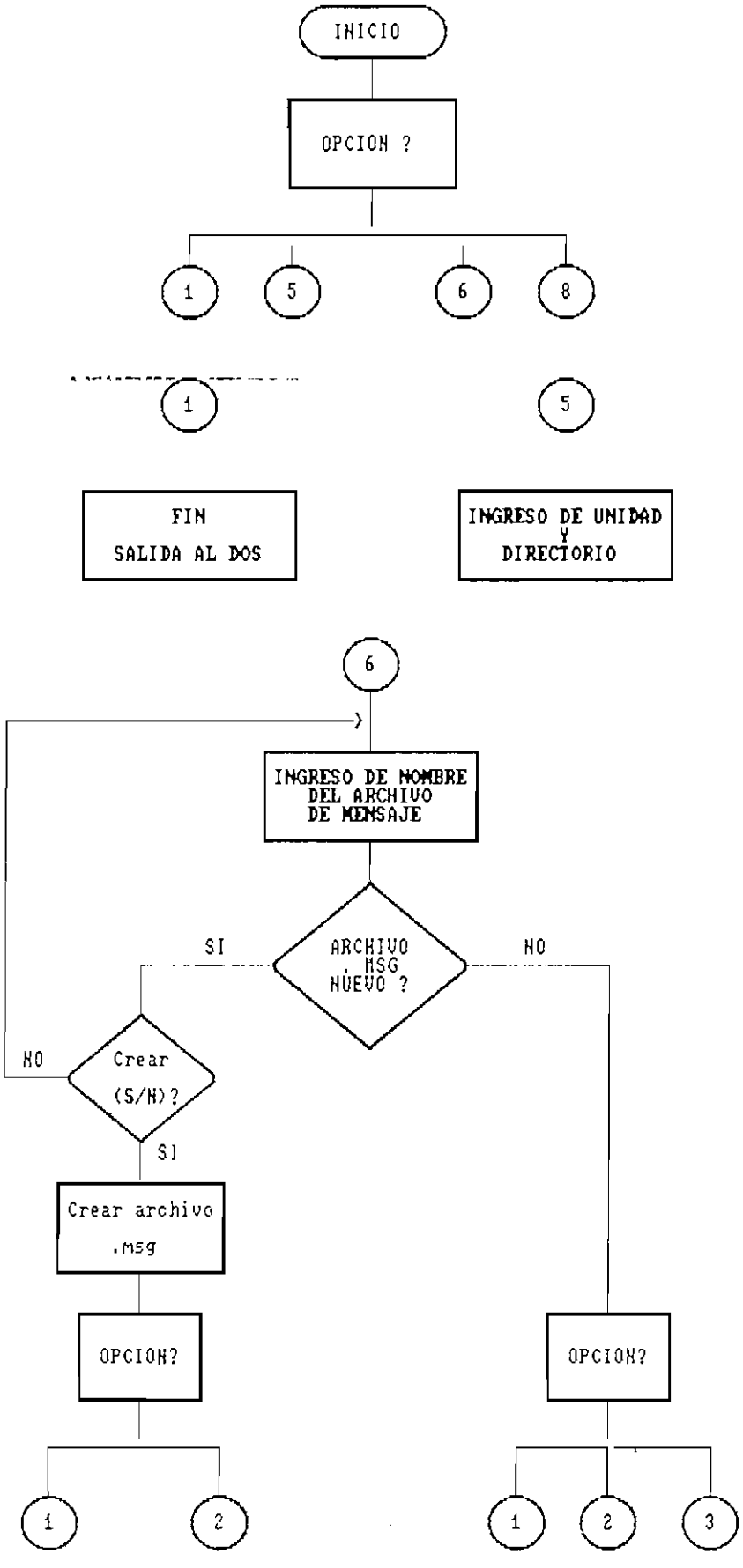
Seguidamente se pone en espera de información. Cuando llegan los datos se calcula el síndrome del bloque de mensaje y el CRC recibidos.

La detección de errores se realiza por chequeo del valor del síndrome, si es cero, el bloque recibido está correcto, si no es cero existen errores.

Cuando el bloque es correcto se graba en el archivo nombre.rcp y se envía al computador emisor el acuse de recibo positivo, si es erróneo se envía el acuse de recibo negativo, luego de lo que se pone nuevamente en espera de información.

Cuando se recibe el carácter que indica el fin de la transmisión se presenta en la pantalla el mensaje completo que ha llegado al receptor, el mismo que también queda

FIG. 4.6. DIAGRAMA DE FLUJO DE LA
TRANSMISION DEL MENSAJE



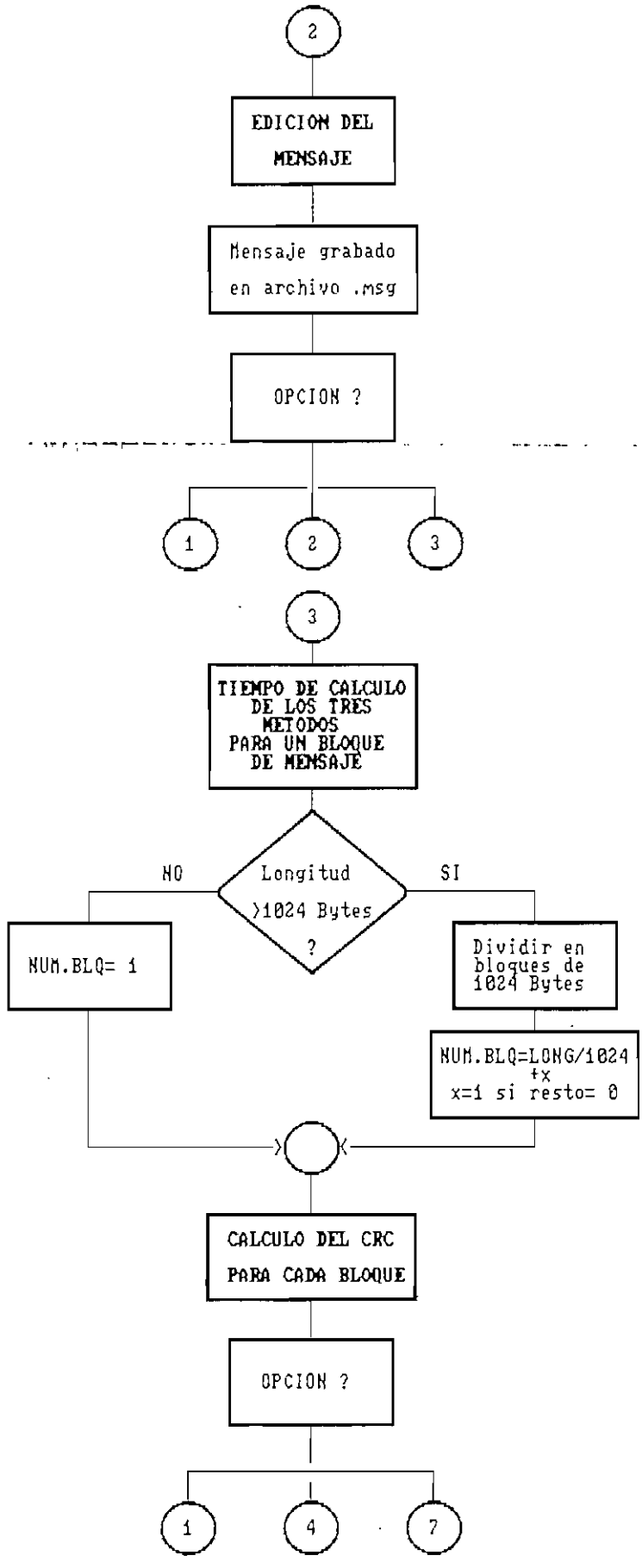


FIG. 4.6 CONTINUACION

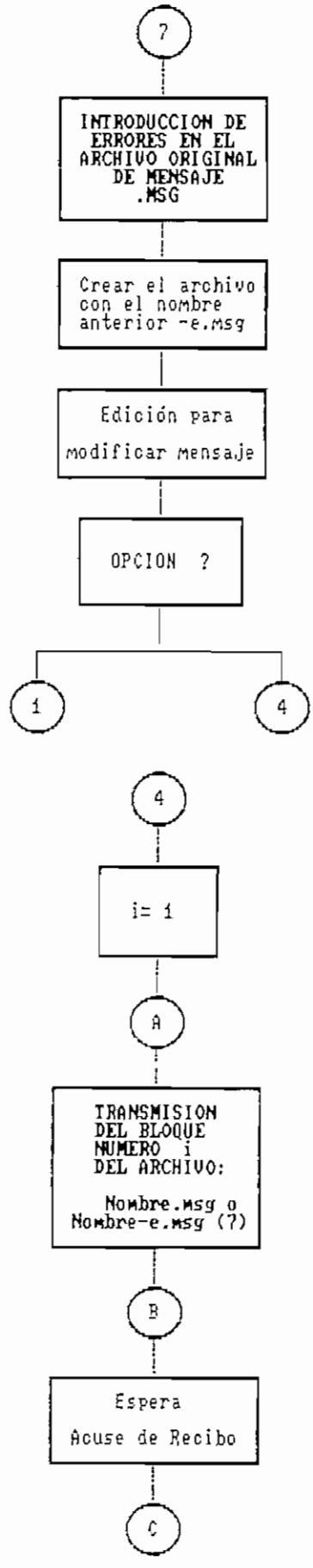
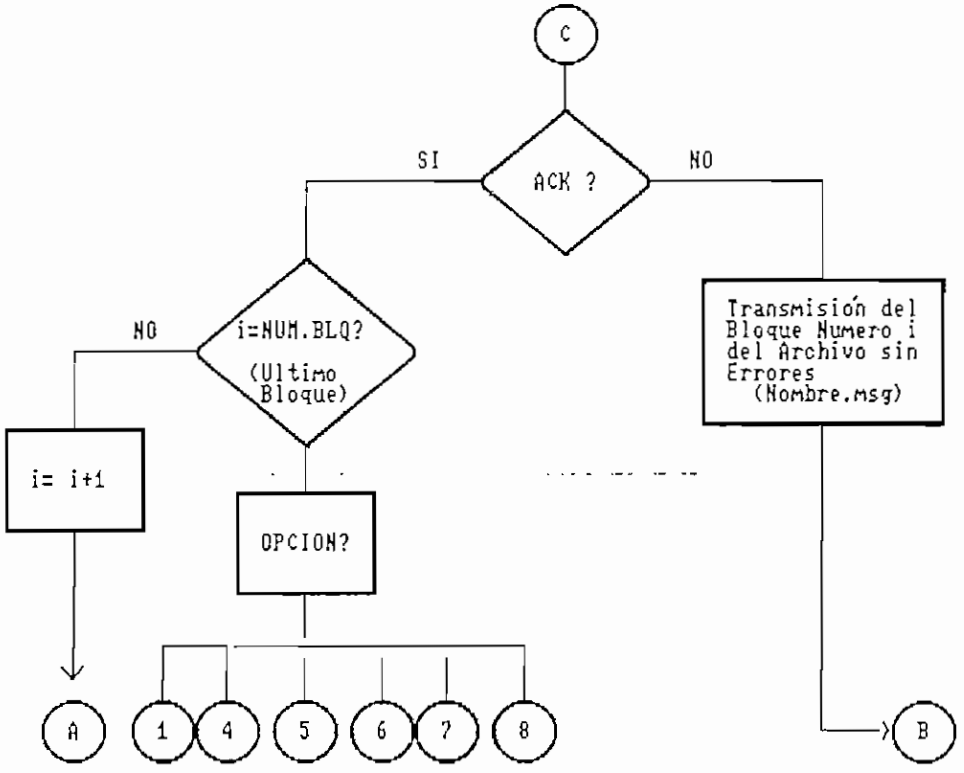


FIG. 4.6 CONTINUACION



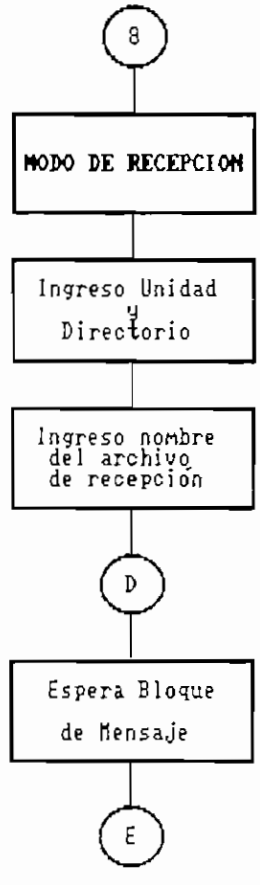


FIG. 4.6 CONTINUACION

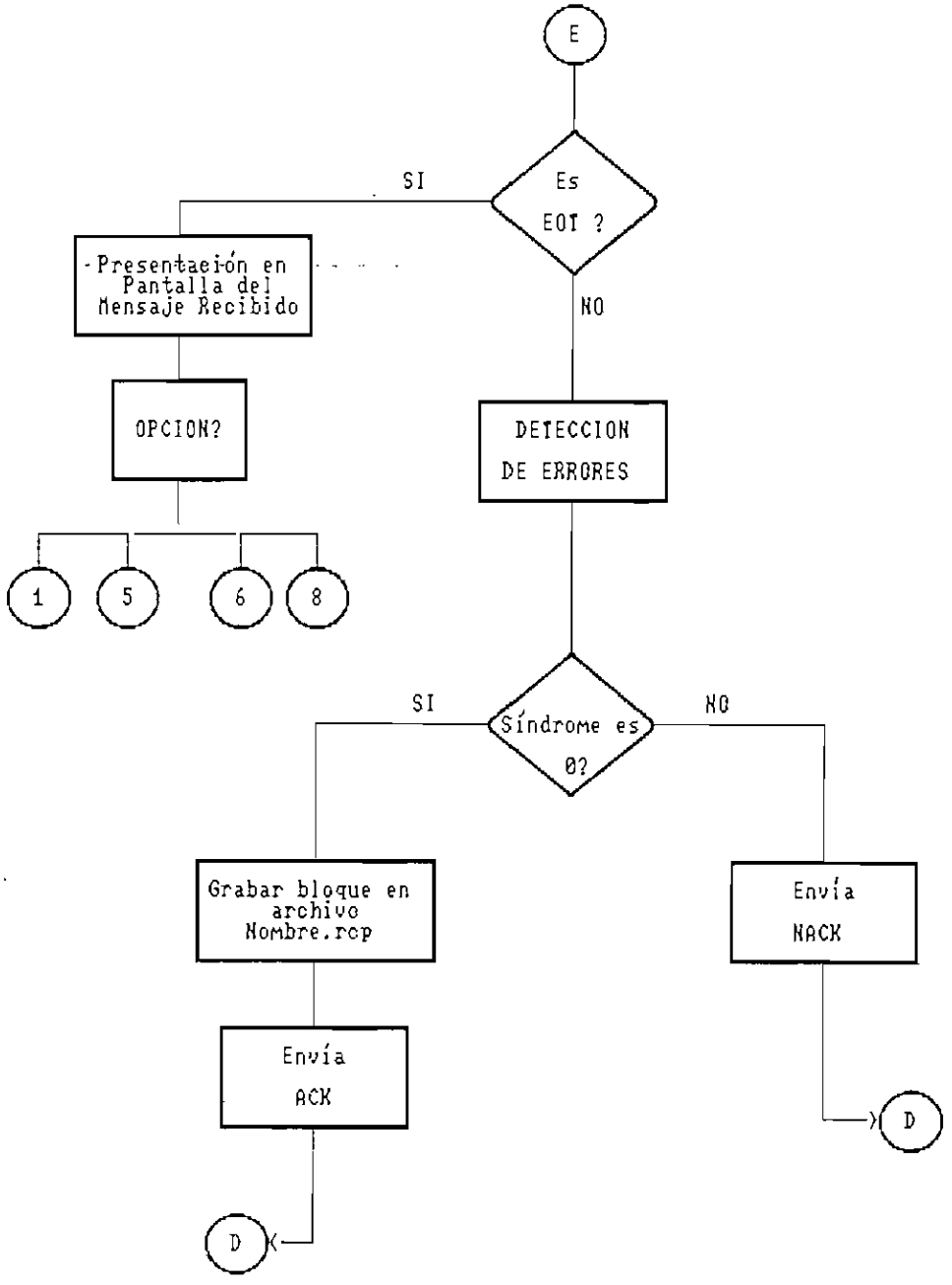


FIG. 4.6 CONTINUACION

CAPITULO V

RESULTADOS CONCLUSIONES Y RECOMENDACIONES

5.1 Ejemplos de Utilización del Programa.

A continuación se presentan dos ejemplos de transmisión de mensajes con el programa CRC.EXE.

Ejemplo (5.1).- Transmisión de un mensaje corto del microcomputador 1 al microcomputador 2; el primero es el emisor.

Microcomputador 1.-

Al ingresar al programa aparece la pantalla con el menú, en el cual se escoge la opción "Cambio de Directorio" para dar la unidad y nombre del directorio:

A ←┘

EMISION ←┘

luego en "Cambio de Archivo", se escribe, para el ejemplo:

MENS_1 ←┘

éste es un archivo nuevo en ese directorio, por lo que respondiendo afirmativamente es creado.

En seguida vamos a la edición con el Norton Editor que

nos coloca en el archivo nombrado anteriormente; A:\EMISION\MENS_1.MSG. El mensaje es:

Prueba de los algortimos de calculo del CRC-CCITT

El próximo paso es el cálculo del CRC y la presentación del tiempo para cada método. En un microcomputador PACKARD BELL XT (1986) con el microprocesador NEC V20 se obtuvieron los siguientes resultados:

CRC con los tres algoritmos: CB1Ah

Tiempo con el método "Bit a Bit": 23,33 mseg

Tiempo con el método "Byte con Tabla": 3,33 mseg

Tiempo con el método "Byte on the Fly: 4,44 mseg

Longitud del bloque: 49 bytes

Luego escogiendo la opción "Transmisión del mensaje" se envía el texto con el CRC y espera hasta recibir el asentimiento. Como en este caso no hay errores llega el ACK y el transmisor termina con el envio del EOT, retornando a un menú que permite salir al DOS o continuar escogiendo una de las opciones.

En la siguiente prueba, accediendo a "Introducción de errores", volvimos al editor para modificar el mensaje anterior utilizando la función sobreescribir que es F6 INS. Se cambió uno de los caracteres y luego se grabó con F3 E.

El nuevo archivo es: A:\EMISION\MENS_1-E.MSG .

En este caso se recibió un NACK y se retransmitió el

mensaje original (sin errores), terminando luego de recibir el ACK respectivo, con el EOT y retornando a un menú del que se sale al DOS.

Microcomputador 2.-

En el menú inicial, al ingresar al CRC se escoge la opción "Modo de recepción" y se ingresa los datos de unidad, directorio y nombre del archivo de recepción:

A ←┘

PRINC ←┘

MENS_1 ←┘

Después de crear el archivo se pone en espera del mensaje.

En el primer caso, cuando se transmitió el mensaje original el síndrome resultante fue cero y se pasó a archivarlo, luego se envió el ACK. Cuando se introdujeron errores el síndrome fue diferente de cero y no se grabaron los datos, sino que se solicitó retransmisión enviando el NACK.

Cuando se recibió el mensaje retransmitido se reconoció como correcto y se grabó. Luego se conoce el nombre del archivo de recepción, el mismo que se presenta en la pantalla para su lectura.

El archivo de recepción es: A:\PRINC\MENS_1.RCP

Ejemplo (5.2).- El mensaje a enviar estuvo previamente editado con el nombre: C:\SIDES\CAP_1.MSG, tiene una

longitud mayor a 2 Kbytes.

El procedimiento seguido fue el mismo del ejemplo (5.1), con la diferencia que el mensaje ya estuvo escrito y como no se deseaba alterarlo, se pasó directamente al cálculo del CRC.

Los resultados para el CRC y el tiempo requerido para el primer bloque son:

CRC con los tres algoritmos: B7E4h

Tiempo con el método "Bit a Bit": 477,77 mseg

Tiempo con el método "Byte con Tabla": 61,66 mseg

Tiempo con el método "Byte on the Fly: 98,33 mseg

Longitud del bloque: 1024 bytes

El mensaje se transmite en tres bloques, dos de los cuales son de 1024 bytes; con el CRC respectivo. Luego de transmitir un bloque se espera el ACK para enviar el siguiente, así hasta el fin del mensaje.

También se probó modificando el texto original (errores) en el bloque 1 y 3 para observar que solamente éstos se retransmiten.

En la recepción también se trabaja con cada bloque para la detección de errores, se graban los bloques cuyo síndrome es cero; cuando llega el carácter de fin de transmisión EOT se observa en la pantalla el mensaje total recibido, por

páginas de 1000 bytes cada una.

5.2 Tiempo de Cálculo de los Algoritmos en función de la Longitud del Mensaje.

La rutina que permite observar el tiempo requerido por un algoritmo para obtener el CRC del bloque de mensaje, se basa en un contador del número de veces que el timer del computador invoca a la interrupción 1ch durante el cálculo.

Este contador se pone en cero antes de empezar el procedimiento.

Se toma como referencia que el temporizador emite aproximadamente 18 interrupciones tipo "tic" -que invocan a la interrupción 1ch- por segundo.

Debido a que el tiempo resulta muy pequeño inclusive para el método más lento, y es difícil de observar, se realiza un cierto número de repeticiones del mismo cálculo, sin que se altere el resultado del CRC; lo que nos permite tener un cierto valor en el contador de interrupciones, con el cual se puede trabajar con mayor facilidad.

A partir de este número de interrupciones se hace la transformación a un formato adecuado. Pudiendo finalmente obtener el tiempo necesario para el cálculo del CRC una sola vez.

Los resultados del tiempo tienen una precisión de:

1/ 18 * número de repeticiones del cálculo (seg).

Hay variación de un computador a otro en razón de su

velocidad. Sin embargo, en un determinado computador permite hacer una comparación de la rapidez de procesamiento con los diferentes algoritmos.

Utilizando el programa LNG_TM1.EXE incluido en el Apéndice E, para varias longitudes de mensaje comprendidas entre 1 y 1024 bytes y con un contador de repeticiones de 100 se obtuvieron los tiempos en segundos para los tres algoritmos. Luego se transformó en milisegundos y se dividió por 100 para tener el valor del tiempo para un solo cálculo. Los resultados se presentan en la tabla 5.1.

Tabla 5.1

Tiempo de Cálculo vs Long. del Mensaje

Mensajes de 1 a 1024 bytes

Número de cálculos= 1 Tiempo: milisegundos

Número de Bytes	"Bit a bit" Método 1	"Con tabla" Método 2	"On the fly" Método 3
1	0.55	0.00	0.00
10	4.44	0.55	1.11
50	23.33	2.77	5.00
100	46.66	6.11	9.44
200	93.33	12.22	19.44
300	140.00	18.33	28.88
400	186.66	24.44	38.33
500	233.88	30.00	47.77
600	280.55	36.11	57.77
700	326.66	42.22	67.22
800	373.33	48.33	76.66
900	420.00	54.44	86.11
1000	467.22	60.00	96.11
1024	478.33	61.66	98.33

Con el mismo programa se evaluó para mensajes más

largos, con el contador de repeticiones en 1, tal como se presenta en la tabla 5.2.

Tabla 5.2

Tiempo de Cálculo vs Long. del Mensaje

Mensajes de 1024 a 64000 bytes

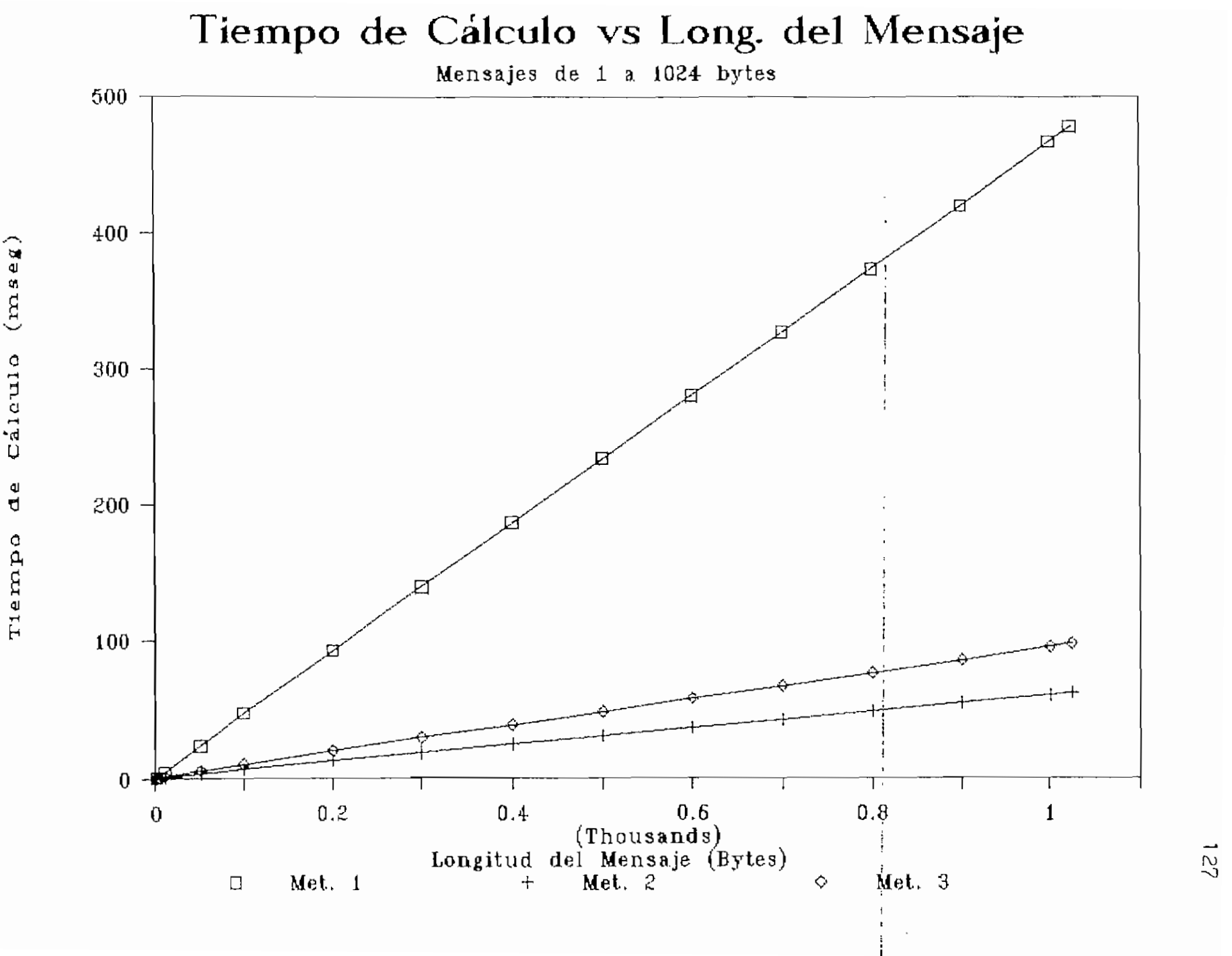
Número de cálculos= 1 Tiempo: segundos

Número de Bytes	"Bit a bit" Método 1	"Con tabla" Método 2	"On the fly" Método 3
1024	0.500	0.055	0.111
5000	2.333	0.277	0.500
10000	4.666	0.611	1.000
15000	7.055	0.888	1.444
20000	9.388	1.166	1.944
25000	11.666	1.500	2.388
30000	14.055	1.833	2.833
35000	16.388	2.111	3.388
40000	18.722	2.388	3.833
45000	21.055	2.722	4.277
50000	23.388	3.000	4.833
55000	25.722	3.333	5.277
60000	28.055	3.611	5.777
64000	29.944	3.833	6.166

Los datos de las tablas anteriores se graficaron, y se tienen los siguientes resultados:

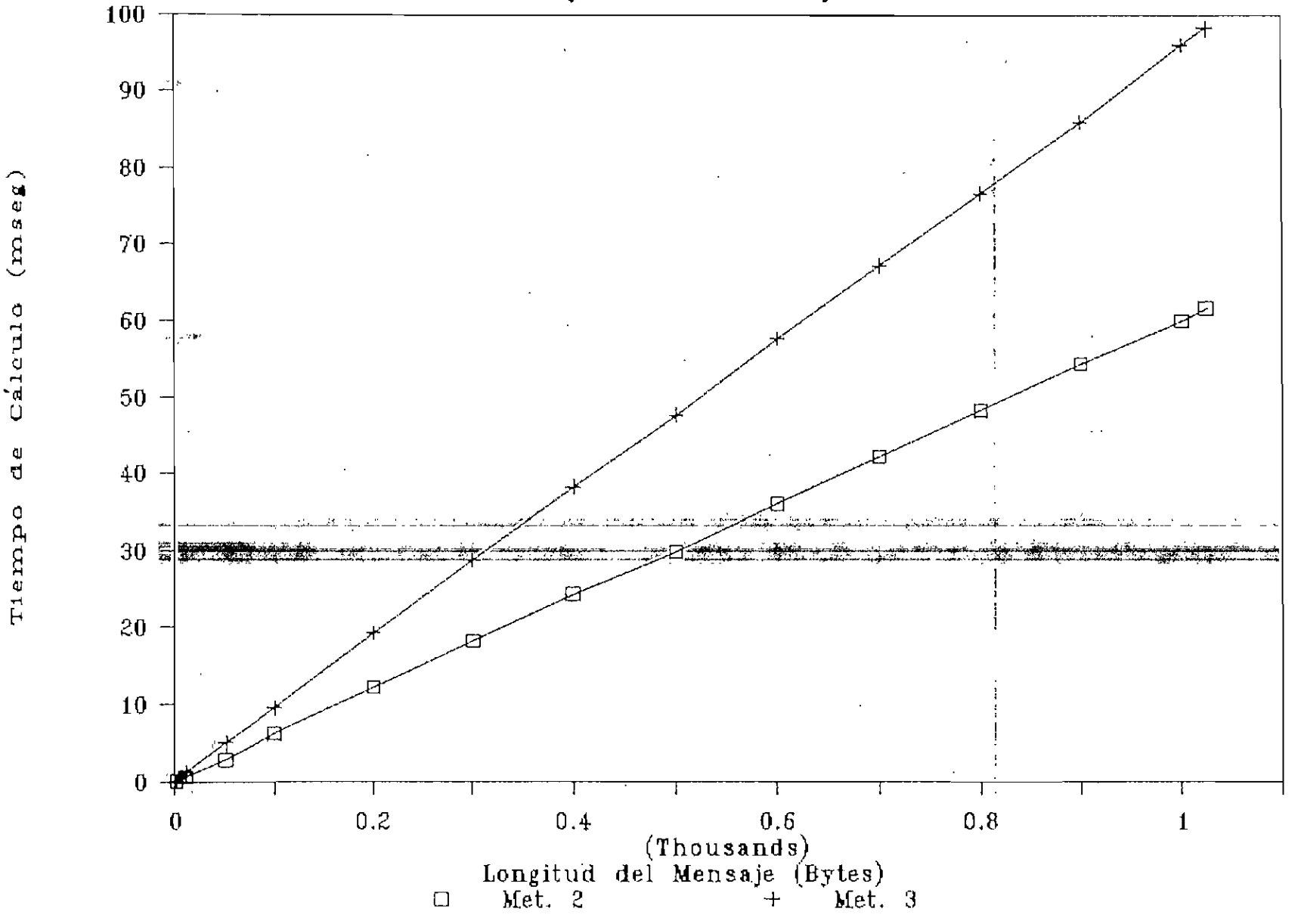
- El tiempo necesario para el cálculo del CRC con los tres algoritmos tiene una relación lineal con la longitud del mensaje, siendo el método de cálculo bit a bit el que crece más rápidamente cuando aumenta el número de bits.

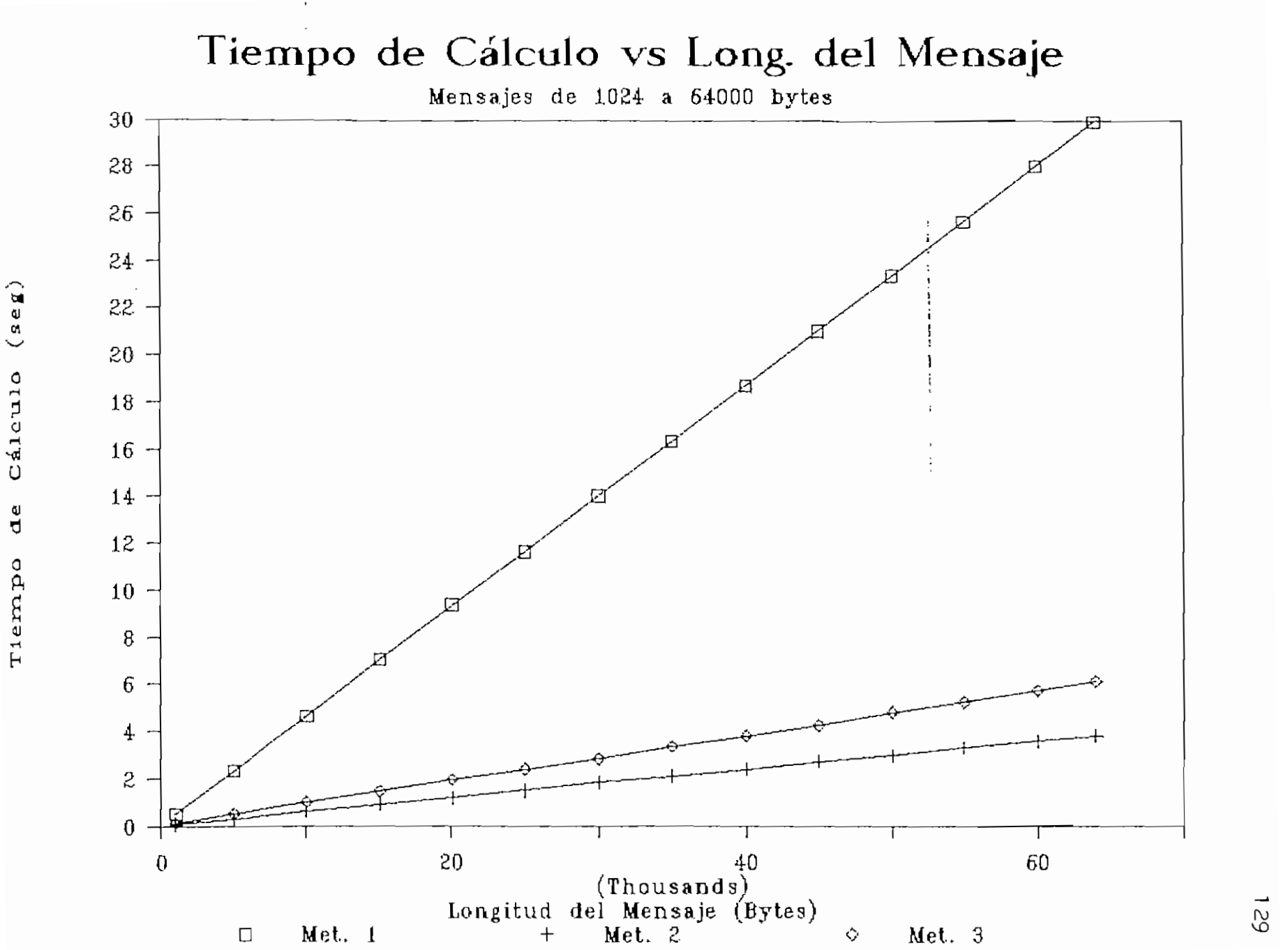
- Con el método bit a bit se necesita un tiempo 4.86 veces mayor que con el cálculo "on the fly" y 7.7 veces más

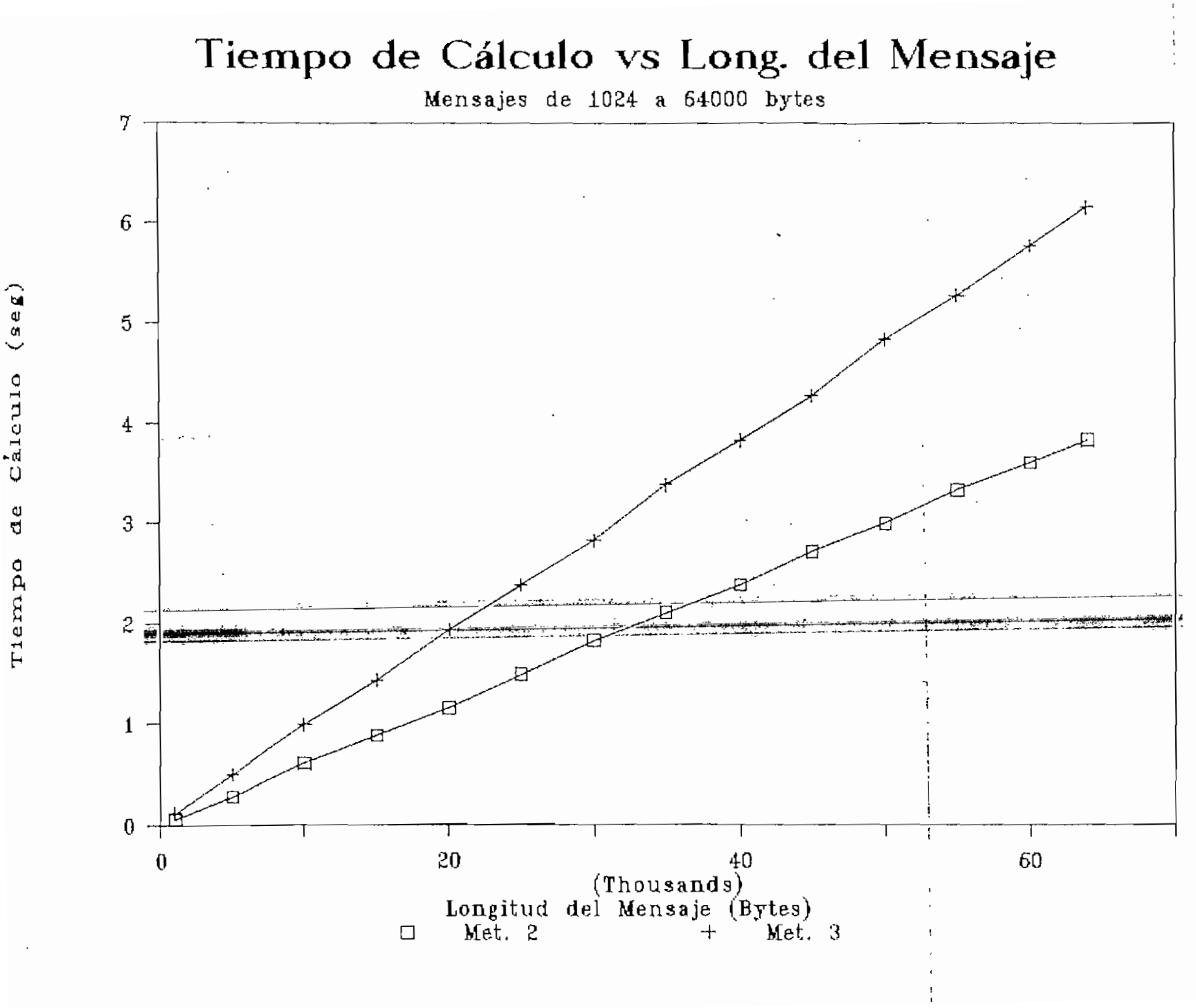

Tiempo de Cálculo vs Long. del Mensaje
Mensajes de 1 a 1024 bytes
Tiempo de Cálculo (mseg)
500
400
300
200
100
0
0
0.2
0.4
0.6
0.8
1
(Thousands)
Longitud del Mensaje (Bytes)
Met. 1
Met. 2
Met. 3

Tiempo de Cálculo vs Long. del Mensaje

Mensajes de 1 a 1024 bytes

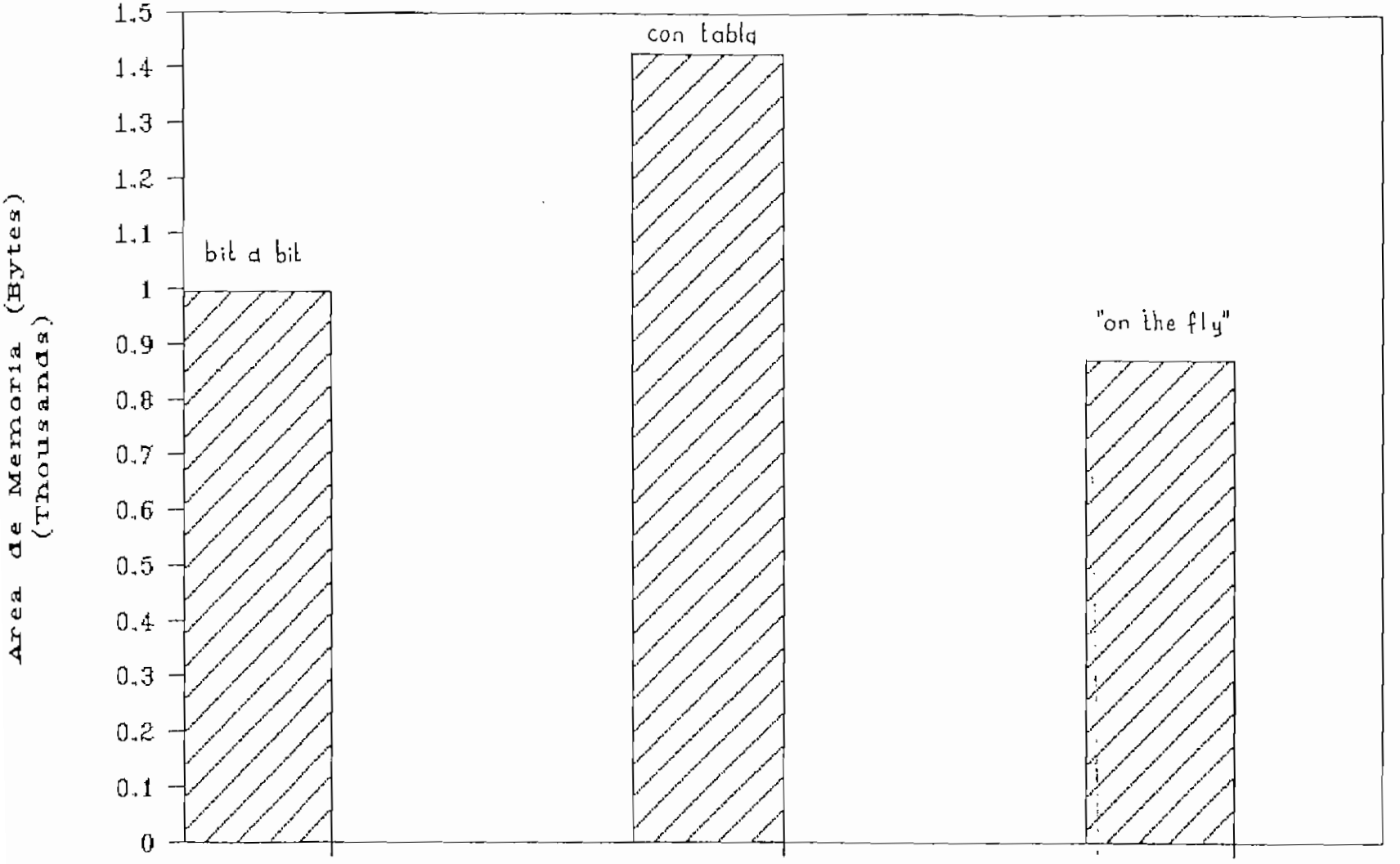



Tiempo de Cálculo vs Long. del Mensaje
Mensajes de 1024 a 64000 bytes
Tiempo de Cálculo (seg)
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
0
20
40
60
(Thousands)
Longitud del Mensaje (Bytes)
Met. 1
Met. 2
Met. 3


Tiempo de Cálculo vs Long. del Mensaje
Mensajes de 1024 a 64000 bytes
Tiempo de Cálculo (seg)
7
6
5
4
3
2
1
0
0
20
40
60
(Thousands)
Longitud del Mensaje (Bytes)
Met. 2
Met. 3

AREA DE MEMORIA

OCUPADA POR CADA ALGORITMO



que con el algoritmo de búsqueda en una tabla; para el mismo mensaje.

- Comparando los métodos 2 y 3 se observa que la diferencia es de aproximadamente 1.6, siendo más alto el tiempo calculando el CRC de un byte dentro del proceso ("on the fly"), que cuando se busca éste en la tabla.

Con la finalidad de comparar el área de memoria ocupada por los algoritmos se han realizado programas separados para calcular el CRC de un buffer, éstos son: BIT.EXE, TABLA.EXE y FLY.EXE (Apéndice E).

Los programas ejecutables sin considerar el mensaje ocupan las áreas que se señalan a continuación:

Método bit a bit (BIT.EXE): 996 bytes.

Método por byte con búsqueda en una tabla (TABLA.EXE): 1428 bytes.

Método por byte "on the fly" (FLY.EXE): 875 bytes.

Con la tabla de CRC de un byte se requiere un 30% más de memoria que con el método de cálculo por bit y el 38% más que con el algoritmo "on the fly".

5.3 Conclusiones y Recomendaciones.

En esta Tesis se ha dado fundamental importancia al estudio teórico de los códigos de detección y corrección de errores, que son la base para la aplicación del chequeo

redundante cíclico (CRC), como estrategia de control de errores en transmisión de datos.

Para tener un conocimiento más amplio fue necesario detallar definiciones y propiedades particularmente sobre los códigos cíclicos, aclarando conceptos más generales que se presentaron mientras se avanzaba.

El desarrollo matemático nos permite establecer procedimientos de cálculo, los cuales se han tomado como referencia al momento de justificar los métodos para la aplicación práctica del CRC.

Partiendo de los algoritmos para el cálculo del CRC presentados en un artículo de la publicación MICRO de la IEEE; realizando las adaptaciones necesarias - para el polinomio del CRC-CCITT - y además manteniendo concordancia con la parte teórica, se llegaron a obtener los algoritmos que se detallan en el tercer capítulo. Estos presentan dos maneras de procesar el mensaje: tomando de bit en bit o a nivel de bytes.

Para establecer las características de los algoritmos en cuanto se refiere a su velocidad y la memoria que ocupan, se han desarrollado los programas correspondientes en software utilizando el lenguaje Assembly del 8086, por su facilidad para el manejo de los datos.

Cuando se considera la información por bytes el proceso se vuelve más rápido que en el caso de tomar de bit en bit; siendo bastante significativo el ahorro de tiempo, si se observan los resultados obtenidos en las pruebas. Esto es una ventaja a tomar en cuenta sobre todo cuando la longitud del mensaje que se desea enviar es grande.

Se probó con algunos mensajes siendo el mayor de 64000 bytes, y se observa que existe una relación lineal de crecimiento del tiempo conforme al aumento de longitud, la que se mantendrá con mensajes más extensos.

En relación al área de memoria que se necesita con los programas de aplicación de los tres algoritmos, no hay una diferencia muy grande entre ellos, y se requiere en el peor de los casos algo más de 1 Kbyte.

Los algoritmos tratados pueden modificarse para el caso de trabajar con un polinomio que no sea el señalado para el CRC-CCITT, sin mayor dificultad.

Se ha desarrollado como aplicación práctica de los algoritmos de cálculo del CRC un programa que integra los diferentes pasos de la transmisión de datos incluyendo el control de errores, éste es el CRC.EXE. Por ser didáctico, permite observar los resultados de cada proceso, para ésto se da la posibilidad de empezar con la edición del mensaje,

de manera que es conocida la información a enviar. Siguiendo la secuencia tanto en la transmisión como en la recepción se conocen los resultados del CRC y de la detección de errores respectivamente, que son nuestro principal interés.

Puede mejorarse el programa en algunos sentidos, por ejemplo con una rutina que permita obtener el tiempo de cálculo de los algoritmos con mayor precisión, para algún caso particular, ya que para una comparación cualitativa como la que se ha realizado no es muy necesario.

En la aplicación se ha incluido un editor para escribir el texto; con la finalidad de permitir a la persona que utilice el CRC.EXE conocer los datos a transmitir y recibir y para la simulación de ocurrencia de errores, pero con algunas modificaciones en el programa se podria ampliar el envio a cualquier tipo de archivos, asi como aumentar la longitud máxima del mensaje que es ahora de 8 Kbytes.

En relación a la parte teórica de la Tesis; no se ha incluido el estudio de la probabilidad de error de los códigos de chequeo redundante cíclico, lo que seria interesante y sin duda merece un tratamiento especial.

En la publicación MICRO de la IEEE de Octubre de 1990, siguiendo con los articulos que se refieren al cálculo del

CRC (utilizados en la tesis), presentan un método de diseño en hardware de codificadores CRCs paralelos, basados en la teoría de filtros digitales, es decir considerando un registro de desplazamiento con realimentación como un filtro digital; y con la transformada z. Además señala el método que permite derivar las ecuaciones lógicas del circuito para cualquier polinomio generador.

El CRC, por lo que se puede ver en el trabajo que se ha citado, sigue siendo estudiado y se buscan continuamente los métodos teóricos que faciliten el diseño de circuitos en hardware.

De esta manera, se dispone de nueva información para analizar y seguir actualizándose en el tema, el mismo que podría también orientarse en otros sentidos.

REVISION DE LA TEORIA DE GRUPOS Y LA ARITMETICA DE CAMPO BINARIO

Siendo G un conjunto de elementos. Una operación binaria * sobre G es una regla que asigna a cada par de elementos a y b un tercer elemento c = a * b, también en G.

En este caso se dice que G es cerrado bajo la operación binaria *.

Definición.- Un conjunto G sobre el cual se ha definido una operación binaria * se denomina GRUPO si cumple las siguientes propiedades:

1. Para cualquier par de elementos a y b en el conjunto G, a * b está también en G.

2. La operación binaria * cumple la propiedad asociativa. Esto es, para a,b y c en el conjunto G:

$$a * (b * c) = (a * b) * c$$

3. Existe en el conjunto un elemento idéntico, e, tal que para todo a en G:

$$a * e = e * a = a$$

4. Para cada elemento a, existe un elemento inverso a^{-1} en el conjunto, que satisface:

$$a * a^{-1} = a^{-1} * a = e$$

Un GRUPO se denomina Abeliano o conmutativo, si cumple la propiedad conmutativa. Es decir que para todo a y b en el conjunto:

$$a * b = b * a$$

Considerando el conjunto de dos elementos, G = {0,1}.

Si se define la operación binaria denotada por $\oplus$ como sigue:

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

la misma que se denomina SUMA MODULO-2, se tiene que el conjunto G es un grupo bajo $\oplus$.

El elemento idéntico es 0. El inverso de 0 es si mismo. y el de 1 también es si mismo.

Se puede comprobar que G cumple la propiedad asociativa. Por ejemplo:

$$0 \oplus 1 \oplus 1 = (0 \oplus 1) \oplus 1 = 1 \oplus 1 = 0$$
$$= 0 \oplus (1 \oplus 1) = 0 \oplus 0 = 0$$

Por tanto, $(0 \oplus 1) \oplus 1 = 0 \oplus (1 \oplus 1)$

Además, como se observa en la tabla de la suma módulo-2, satisface la propiedad conmutativa, por lo que G es un grupo Abeliano bajo $\oplus$.

Puesto que $1 \oplus 1 = 0$, $1 = -1$. En aritmética módulo-2 la sustracción es idéntica a la adición.

Dado un número primo p (p=2,3,5,7,11....). Considerando el conjunto de enteros $G = \{1,2,3.....,p-1\}$ y definiendo

la operación binaria $\boxdot$ sobre G como sigue:

Para i y j en G, $\quad i \boxdot j = r$

Donde 1 es el idéntico y r el inverso del elemento i; puede decirse que G es cerrado bajo la operación binaria denominada MULTIPLICACION MODULO-P.

Si p=2 se tiene, G = {1}. Definiendo la operación binaria MULTIPLICACION MODULO-2 de la siguiente manera:

$$0 \boxdot 0 = 0$$

$$0 \boxdot 1 = 0$$

$$1 \boxdot 0 = 0$$

$$1 \boxdot 1 = 1$$

Para el conjunto G = {1} se tiene como idéntico e inverso el elemento 1. Además cumple la primera propiedad de grupos. Por lo que G es un grupo bajo la multiplicación módulo-2.

Dado que G satisface a la vez la propiedad conmutativa, éste es un grupo Abeliano bajo $\boxdot$.

CAMPOS.-

Un campo es un conjunto de elementos en el que puede realizarse las operaciones de suma, resta, multiplicación y división sin salir del conjunto.

Definición.- Un conjunto F de elementos, en el cual dos operaciones binarias llamadas adición "+" y multiplicación "·" están definidas, es un campo, si las operaciones + y · cumplen con las siguientes condiciones:

1. F es un grupo conmutativo bajo la adición. El elemento idéntico con respecto a la adición se llama cero o idéntico aditivo y es denotado por 0.

2. El conjunto de elementos diferentes de cero en F, es un grupo conmutativo bajo la multiplicación. El elemento idéntico para esta operación es llamado unidad o idéntico multiplicativo de F y es denotado por 1.

3. La multiplicación es distributiva sobre la adición, ésto es, que para tres elementos a,b y c en F se tiene:

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

El número de elementos en un campo es llamado el orden del campo. Así, para un número finito de elementos se denomina campo finito.

En un campo el aditivo inverso de a es denotado -a, y el multiplicativo inverso de a es a^{-1} siempre que $a \neq 0$. La sustracción de un elemento b de otro a equivale a:

$$a - b \triangleq a + (-b)$$

La división de a por b se define como:

$$a \div b \triangleq a \cdot b^{-1}$$

Se considera el conjunto {0,1} con la adición y multiplicación módulo-2 definidas en las tablas 1 y 2.

TABLA 1

ADICION MOD-2

+	0	1
0	0	1
1	1	0

TABLA 2

MULTIPLICACION MOD-2

·	0	1
0	0	0
1	0	1

Se demostró que {0,1} es un grupo Abeliano bajo la adición módulo-2 y que el conjunto {1} es un grupo conmutativo bajo la multiplicación módulo-2.

La propiedad conmutativa puede comprobarse calculando: a·(b+c) y a·b+a·c para las ocho posibles combinaciones de a,b y c (a=0 o 1, b=0 o 1, c=0 o 1). Por ejemplo:

a=0, b=0, c=0

$$0\cdot(0+0) = 0\cdot 0 = 0$$
$$0\cdot 0 + 0\cdot 0 = 0$$

a=1, b=1, c=1

$$1\cdot(1+1) = 1\cdot 0 = 0$$
$$1\cdot 1 + 1\cdot 1 = 1+1 = 0$$

Por tanto, el conjunto {0,1} es un campo de dos elementos bajo la adición y multiplicación módulo-2.

Este campo es denominado CAMPO BINARIO y se denota por GF(2).

El campo binario tiene un papel importante en la teoría de la codificación y es ampliamente utilizado en la transmisión o almacenamiento de datos digitales.

POLINOMIOS EN CAMPO BINARIO.-

Para polinomios cuyos coeficientes son del campo binario GF(2), un polinomio f(X) con una variable X y con coeficientes de GF(2) tiene la siguiente forma:

$$f(X) = f_0 + f_1 X + f_2 X^2 + \ldots\ldots + f_n X^n$$

donde f_i=0 o 1 para $0 \leq i \leq n$. El grado del polinomio es la mayor potencia de X con coeficiente diferente de cero.

En el polinomio anterior, si f_n=1, f(X) es de grado n; si f_n=0, f(X) es de grado menor que n.

Hay dos polinomios sobre GF(2) con grado 1 : X y 1+X. Son cuatro los de grado 2: X^2, $1+X^2$, $X+X^2$, y $1+X+X^2$.

En general, hay 2^n polinomios sobre GF(2) con grado n.

Los polinomios sobre GF(2) pueden sumarse (sustraerse), multiplicarse y dividirse de la forma que se indica a continuación:

Siendo $g(x) = g_0 + g_1 X + g_2 X^2 + \ldots\ldots + g_m X^m$ otro polinomio sobre GF(2). Para sumar f(X) y g(X), simplemente se suma los coeficientes de la misma potencia de X en f(X) y g(X), asumiendo que $m \leq n$ resulta:

$$f(X)+g(X) = (f_0+g_0) + (f_1+g_1)X + \ldots\ldots\ldots$$
$$+ (f_m+g_m)X^m + f_{m+1}X^{m+1} + \ldots\ldots + f_nX^n$$

donde f_i+g_i se realiza como suma en módulo-2.

Por ejemplo: sumando:

$a(X) = 1 + X + X^3 + X^5$ y $b(X) = 1 + X^2 + X^3 + X^4 + X^7$, se obtiene:

$$a(X)+b(X) = (1+1) + X + X^2 + (1+1)X^3 + X^4 + X^5 + X^7$$
$$= X + X^2 + X^4 + X^5 + X^7$$

Cuando se multiplica $f(X)$ y $g(X)$ resulta:

$$f(X)\cdot g(X) = c_0 + c_1X + c_2X^2 + \ldots\ldots + c_{n+m}X^{n+m}$$

donde.

$$c_0 = f_0g_0$$
$$c_1 = f_0g_1 + f_1g_0$$
$$c_2 = f_0g_2 + f_1g_1 + f_2g_0$$
$$\vdots$$
$$c_i = f_0g_i + f_1g_{i-1} + f_2g_{i-2} + \ldots\ldots + f_ig_0$$
$$\vdots$$
$$c_{n+m} = f_ng_m$$

La multiplicación se realiza en módulo-2. Si $g(X)=0$, entonces $f(X)\cdot 0 = 0$.

Suponiendo que el grado de $g(X)$ es diferente de cero. Cuando se divide $f(X)$ por $g(X)$ se obtiene un único par de polinomios sobre $GF(2)$ - $q(X)$, llamado cuociente y $r(X)$, llamado residuo - tales que

$$f(X) = q(X)\, g(X) + r(X)$$

con $r(X)$ de grado menor que $g(X)$. Este se conoce como el

Algoritmo de División Euclidiana.

Por ejemplo. Dividiendo $f(X) = 1 + X + X^4 + X^5 + X^6$ por $g(X) = 1 + X + X^3$, se tiene:

$$
\begin{array}{rl}
 & X^3 + X^2 \quad \text{(Cuociente)} \\
 & \hline \\
X^3 + X + 1 \quad : & X^6 + X^5 + X^4 \qquad + X + 1 \\
 & X^6 \qquad + X^4 + X^3 \\
 & \hline \\
 & \quad X^5 \qquad + X^3 \quad + X + 1 \\
 & \quad X^5 \qquad + X^3 + X^2 \\
 & \hline \\
 & \qquad\qquad\qquad\quad X^2 + X + 1 \quad \text{(Residuo)}
\end{array}
$$

Por tanto: $X^6 + X^5 + X^4 + X + 1 = (X^3 + X^2)(X^3 + X + 1) + X^2 + X + 1$

Si el residuo $r(X)$ es cero, se dice que $f(X)$ es divisible por $g(X)$, y $g(X)$ es un factor de $f(X)$.

Para números reales, si a es una raíz de un polinomio $f(X)$ [e.d. $f(a)=0$], $f(X)$ es divisible por $x-a$. Esto se cumple para $f(X)$ sobre GF(2).

Por ejemplo, para $f(X) = 1 + X^2 + X^3 + X^4$, sustituyendo $X=1$ se obtiene:

$$f(1) = 1 + 1^2 + 1^3 + 1^4 = 1+1+1+1 = 0$$

por lo que $f(X)$ tiene a 1 como raíz, y es divisible por $X+1$.

$$
\begin{array}{rl}
 & X^3 + X + 1 \\
 & \hline \\
X + 1 \quad : & X^4 + X^3 + X^2 \qquad + 1 \\
 & X^4 + X^3 \\
 & \hline \\
 & \qquad\quad X^2 \qquad + 1 \\
 & \qquad\quad X^2 + X \\
 & \hline \\
 & \qquad\qquad\quad X \qquad + 1 \\
 & \qquad\qquad\quad X \qquad + 1 \\
 & \hline \\
 & \qquad\qquad\qquad\quad 0
\end{array}
$$

NOCIONES BASICAS DE VECTORES ESPACIALES Y CODIGOS BLOQUE LINEALES

VECTORES ESPACIALES.-

Sea V un conjunto de elementos en el que la operación binaria adición (+) está definida. Sea F un campo. Y sea (·) la operación multiplicación definida entre los elementos de F y V. El conjunto V se denomina un vector espacial sobre el campo F si satisface las siguientes condiciones:

1. V es un grupo conmutativo bajo la adición.
2. Para cualquier elemento "a" en "F" y cualquier elemento "v" en "V", "a · v" es también un elemento de "V".
3. Para los elementos u,v en V y a,b en F,

$$a \cdot (u + v) = a \cdot u + a \cdot v$$

$$(a + b) \cdot v = a \cdot v + b \cdot v$$

4. Para cualquier v en V y a en F.

$$(a \cdot b) \cdot v = a \cdot (b \cdot v)$$

5. Siendo 1 el elemento unidad de F. Para cualquier v en V, $1 \cdot v = v$.

Los elementos de V se llaman vectores y los de F escalares.

Si V es un vector espacial sobre F, puede haber un subconjunto S de V que también sea un vector espacial sobre F. Tal subconjunto se denomina subespacio de V.

donde la suma y multiplicación se realizan en módulo-2. El producto punto es un escalar en GF(2). Si $u \cdot v = 0$, u y v se dice que son **ortogonales** entre si.

Si S es un subespacio k-dimensional de V_n y S_d es el conjunto de vectores en V_n tales que, para u en S y v en S_d, $u \cdot v = 0$; S_d se denomina **espacio nulo (o dual)** de S.

CODIGOS BLOQUE LINEALES.-

Un código bloque es lineal si y sólo si la suma módulo-2 de dos palabras código es otra palabra código.

Puesto que un código lineal C(n,k) es un subespacio k-dimensional de V_n, es posible encontrar k palabras código linealmente independientes, $g_0, g_1, \ldots, g_{k-1}$, tales que cualquier palabra de código v en C es una combinación lineal de éstas, asi,

$$v = u_0 g_0 + u_1 g_1 + \ldots + u_{k-1} g_{k-1}$$

donde $u_i = 0$ o 1 para $0 \le i < k$. Arreglando en forma de una matriz kxn se obtiene:

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ \vdots \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & \ldots & g_{0,n-1} \\ g_{10} & g_{11} & \ldots & g_{1,n-1} \\ & & \vdots & \\ & & \vdots & \\ & & \vdots & \\ g_{k-1,0} & g_{k-1,1} & & g_{k-1,n-1} \end{bmatrix}$$

donde $g_i = (g_{i0}, g_{i1}, \ldots, g_{i,n-1})$ para $0 \le i < k$. Si $u = (u_0, u_1, \ldots u_{k-1})$ es el mensaje a codificarse, la correspondiente palabra de

mensaje y n-k bits son redundantes, y resultan de la suma de los bits de información. Las n-k ecuaciones (de v_j), se llaman ecuaciones de chequeo de paridad del código.

Para cualquier matriz G kxn con k filas linealmente independientes, existe una matriz H de orden (n-k)xn. con n-k filas linealmente independientes, de manera que cualquier vector en el subespacio S de G es ortogonal a las filas de H.

Así, una n_tupla v es una palabra código generada por G si y sólo si $v \cdot H^T = 0$. H se denomina matriz de chequeo de paridad del código. Las 2^{n-k} combinaciones lineales de las filas de la matriz H forman un código lineal C_e llamado dual de C. Este código es el espacio nulo de C, es decir que para cualquier vector v en C y cualquier vector w en C_e $v \cdot w = 0$

Para un código lineal (n,k) generado por la matriz en forma sistemática, la matriz de chequeo de paridad tiene la siguiente forma:

$$H = [I_{n-k}\ P^T]$$

$$H = \left[\begin{array}{ccccc|ccc} 1 & 0 & 0 & \cdots & 0 & p_{00} & p_{10} \cdots & p_{k-1,0} \\ 0 & 1 & 0 & \cdots & 0 & p_{01} & p_{11} \cdots & p_{k-1,1} \\ 0 & 0 & 1 & \cdots & 0 & p_{02} & p_{12} \cdots & p_{k-1,2} \\ & & & & & & \vdots & \\ & & & & & & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & p_{0,n-k-1} & p_{1,n-k-1} \cdots & p_{k-1,n-k-1} \end{array}\right]$$

donde P^T es la matriz transpuesta de P.

El producto punto de la i_ésima fila de G y la j_ésima fila de H es:

$$g_i \cdot h_j = p_{ij} + p_{ij} = 0$$

para $0 \leq i < k$ y $0 \leq j < n-k$. Esto implica que $G \cdot H_T = 0$

DISTANCIA MINIMA DE UN CODIGO BLOQUE.-

Siendo $v = (v_0, v_1, \ldots, v_{n-1})$ una n_tupla. El ancho de Hamming de v, denotado por w(v), es definido como el número de componentes diferentes de cero de v.

Para v y w dos n_tuplas, la distancia de Hamming entre v y w denotada por d(v,w), se define como el número de lugares en que éstos difieren.

CAPACIDAD DE DETECCION Y CORRECCION DE ERRORES.-

Si la distancia mínima de un código bloque es d_{min}. dos vectores de código difieren en al menos d_{min} lugares. Un modelo de error (vector de error e), de $d_{min}-1$ o menos errores no puede cambiar un vector de código en otro.

Un código bloque de d_{min} puede detectar todos los errores de $d_{min}-1$ o menos. No los de longitud d_{min}, ya que en este caso existen por lo menos dos vectores con esta distancia, por lo que podría cambiar de uno a otro.

Para un código bloque (n,k), se tienen 2^n-1 posibles modelos de error, de éstos hay 2^k-1 que son idénticos a 2^k-1 palabras de código diferentes de cero, que podrían cambiar un vector de código v en otro w y se decodificaría incorrectamente. Así, son 2^k-1 los modelos de error no detectables y 2^n-2^k los que pueden detectarse.

Un código bloque de distancia mínima d_{min} garantiza corregir todos los modelos de error de $t = [(d_{min}-1)/2]$ o menos errores, donde $t = [(d_{min}-1)/2]$ denota el máximo entero no mayor que $(d_{min}-1)/2$.

El parámetro t es la capacidad de corrección de errores aleatorios.

ARREGLO ESTANDAR.-

Siendo $v_1, v_2, \ldots, v_{2^k}$ los vectores de código en C. Un esquema de decodificación es una regla de partición de los 2^n posibles vectores recibidos en los 2^k subconjuntos $D_1, D_2, \ldots, D_{2^k}$, de manera que el vector de código v_i está contenido en D_i para $1 \leq i \leq 2^k$.

Se decodifica correctamente cuando el vector r está en el subconjunto D_i que corresponde al vector de código transmitido.

Para un código lineal C se tiene un arreglo de filas y

```
; qlcrc.ass
; contains: qlcrc()

                    include    model.h
                    include    prologue.h
                    name calccrc
                    xseg       _calcrc

;==)--     unsigned int qlcrc(count,startvalue,buffer)
;
;;        ARGUMENTS:
;          (unsigned int)       count    -  Size of buffer in bytes
;          (unsigned int) startvalue -     Initial value of CRC
;          (char *)  buffer        -       Pointer to buffer
;
;
;;        DESCRIPTION:
;          Calculate CRC X^16+X^12+X^5+1 for buffer of count length.  Note:
;          This function can be used to calculate the CRC for an entire
;          buffer at once or for calculating a buffer a character at a time.
;
;          Example for calculating the CRC of an entire buffer:
;
;                    unsigned int crcvalue;
;                    extern char *buffer;
;
;                    crcvalue=qlcrc(1024,0,buffer);
;
;           The return value (crcvalue) would be the CRC for the entire
;           1024 byte buffer.
;
;
;          Example for calculating the CRC of a buffer one byte at a time with
;          other processing in a loop.
;
;                    unsigned int crcvalue;
;                    extern char *buffer;
;                    int buffersize;
;
;                    crcvalue=0;
;
;                    while(buffersize--) {
;                              crcvalue=qlcrc(1,crcvalue,buffer);
;                                        .
;                                        .         (other processing)
;                                        .
;                    }
;                    The CRC for the buffer can be found at this point in
;                    the variable crcvalue.
;
;;        RETURNS:
;          CRC for the buffer of length count
```

```
;
;;        AUTHOR:
;          David Nienhiser   20-JUN-1988  14:07:52.76
;          Copyright (C)1988 Greenleaf Software Inc.
;
;;        MODIFICATIONS:
;
;;;
                    cproc     glcrc
                    if        _LDATA
                     lds      bx,parm3_  ;DS:BX points to buffer
                    else
                     mov      bx,parm3_  ;DS:BX points to buffer
                    endif
                    mov       cx,parm1_  ;CX = number of bytes in buffer
                    mov       ax,parm2_  ;AX = Previous CRC
loop2:              xor       al,[bx]              ;AH^=data byte
                    mov       dh,8                 ;repeat 8 times
loop1:              shr       ax,1                 ;AX<<=1
                    jnc       glcrc1               ;if was not set
                    xor       ax,0840BH  ;AX^=0X8408
glcrc1:             dec       dh         ;decrement counter
                    jnz       loop1
                    inc       bx         ;point to next byte in buffer
                    loop      loop2      ;do next byte if count != 0
                    if        _LDATA
                     pop      ds
                    endif
                    cproce
                    endps
                    end
```

SECCIÓN 4

PROTECCIÓN CONTRA ERRORES

Recomendación V.40

INDICACIÓN DE ERRORES EN CASO DE UTILIZARSE EQUIPO ELECTROMECÁNICO

(Mar del Plata, 1968)

La explotación por medio de un código que prevea la introducción en cada señal de carácter de una unidad suplementaria para el control de paridad permite detectar errores con aparatos electromecánicos, no sólo en el canal de transmisión, sino también en una parte del equipo mecánico de traducción o de transmisión.

En consecuencia, cabe prever entonces que cuando se detecte un error en una señal de carácter, aparezca una indicación de error en la posición en que se ha advertido el mismo.

Esta indicación podría consistir en una perforación suplementaria en las cintas de los aparatos que trabajan con cinta perforada, o en una impresión especial en los equipos de impresión directa.

Estos dispositivos serían, sin embargo, muy costosos o sólo parcialmente eficaces (por ejemplo, muchas señales de caracteres del Alfabeto Internacional N.° 5 no corresponden a ninguna impresión, de forma que la señal normal correspondiente a tales caracteres no puede ser sustituida por un signo «error»).

Por estos motivos, el CCITT *recomienda por unanimidad*

el empleo de un dispositivo de alarma o contador de errores como el método más adecuado para indicar localmente que se ha detectado un error en una señal de carácter.

Recomendación V.41

SISTEMAS DE PROTECCIÓN CONTRA ERRORES INDEPENDIENTES DEL CÓDIGO EMPLEADO

(Mar del Plata, 1968; modificada en Ginebra, 1972)

1 Consideraciones generales

La presente Recomendación concierne especialmente a los sistemas de protección contra errores constituidos por un órgano intermedio que puede acompañar al equipo de terminación del circuito de datos o al equipo terminal de datos. En las figuras 1/V.41 y 2/V.41 se representan los interfaces adecuados. El sistema no está particularmente previsto para su uso con sistemas de computadores multiacceso. No se excluye el empleo de cualquier otro sistema de protección contra errores que responda mejor a necesidades especiales.

Los modems que se utilicen habrán de comprender canales simultáneos de ida y de retorno. El sistema emplea un modo de transmisión síncrono por el canal de ida y un modo de transmisión asíncrono por el canal de retorno. Las disposiciones de la Recomendación V.5 [1] son aplicables a modems conformes con la Recomendación V.23 [2], a velocidades binarias de 1200 o de 600 bit/s por la red telefónica general con conmutación, considerándose al equipo de protección contra errores como un equipo de transmisión. El margen del receptor síncrono ha de ser de ± 45% como mínimo.

El sistema utiliza la transmisión de la información por bloques de longitud determinada [240, 480, 960 o 3840[1] bits]; por ello, es especialmente apropiado para la transmisión de mensajes de mediana y gran longitud. No obstante, para mejorar la eficacia en la transmisión de mensajes más cortos, incluye un procedimiento de arranque rápido.

La protección contra los errores está asegurada por la repetición automática de un bloque a petición del receptor de datos (ARQ). Si el receptor incluye una memoria, los errores detectados se eliminan antes de la salida del sistema (texto «limpio»). El transmisor debe comprender una memoria con una capacidad mínima de dos bloques de datos.

El tren de bits transmitido hacia adelante se divide en bloques, cada uno de los cuales comprende cuatro bits de servicio, bits de información y 16 bits para la detección de errores (o de control), por este orden; los bits de control los genera un codificador cíclico. Así, cada bloque transmitido por la línea contiene 260, 500, 980 o 3860[1] bits.

El sistema de protección contra errores detecta:

a) todos los bloques con un número impar de errores;

b) toda ráfaga de errores cuya longitud no exceda de 16 bits, y un gran porcentaje de otras formas de distribución de errores.

En el supuesto de que los errores se distribuyan como se indica en la referencia [3], una prueba simulada con computador demuestra que el coeficiente de mejora de la tasa de errores es del orden de 5×10^{-4} para bloques de 260 bits.

El empleo de este sistema de bloques de longitud fija está limitado a las líneas en las que el tiempo de propagación en bucle no es superior a los valores indicados en el cuadro 1/V.41. Estos valores incluyen un retardo total de 40 ms en el modem y de 50 ms para la detección de la señal RQ.

CUADRO 1/V.41

Tiempo de propagación en bucle máximo admisible (ms)

Velocidades binarias (bit/s) / Dimensión de los bloques (bits)	200	600	1200	2400	3600	4800
260	1210	343	127	18	–	–
500	2410	743	327	118	49	14
980	4810	1543	727	318	182	114
3860	19210	6343	3127	1518	982	714

2 Procesos de codificación y de verificación

Considerados en conjunto, los bits de servicio y los bits de información corresponden numéricamente a los coeficientes de un polinomio de mensaje cuyos términos van de x^{n-1} (n = número total de bits en un bloque o *secuencia*) a x^{16}, por orden decreciente. Este polinomio se divide en módulo 2 por el polinomio generador $x^{16} + x^{12} + x^5 + 1$. Los bits de control corresponden numéricamente a los coeficientes de los términos que van de x^{15} a x^0 del polinomio que queda como resto de esta división. El bloque completo, compuesto de los bits de servicio y de los bits de información, seguidos de los bits de control, corresponde numéricamente a los coeficientes de un polinomio perfectamente divisible en módulo 2 por el polinomio generador.

En el transmisor, los bits de servicio y de información se someten a un proceso de codificación que equivale a una división por el polinomio generador. El resto que se obtiene se transmite a línea inmediatamente después de los bits de información, por orden decreciente de términos.

Al llegar al receptor, cada bloque se somete a un proceso de decodificación que equivale a una división por el polinomio generador; esta división dará un resto cero si no hay errores. La presencia de un resto significa que hay errores.

---

[1] Esta longitud de bloques es apropiada para circuitos establecidos mediante satélites geoestacionarios.

Estos procesos pueden realizarse fácilmente con un registro de desplazamiento cíclico de 16 pasos, con puertas de realimentación adecuadas (véanse las figuras I-1/V.41 y I-2/V.41). Antes de comenzar el tratamiento de un bloque, se pone el conjunto del registro en la posición 0. En el receptor, la condición general 0 al final del tratamiento de un bloque indica una recepción exenta de errores.

Empleo de aleatorizadores — Si se utilizan aleatorizadores de sincronización automática (es decir, aleatorizadores que efectivamente dividan el polinomio del mensaje por el polinomio del aleatorizador en el transmisor, y multipliquen el polinomio recibido por el polinomio del aleatorizador del receptor) para lograr el funcionamiento satisfactorio del sistema de detección de errores, el polinomio del aleatorizador y el polinomio generador de la Recomendación V.41 no deben tener ningún factor común. Si esta condición no puede asegurarse, el proceso de aleatorización debe preceder al proceso de codificación para la detección de errores, y el proceso de desaleatorización debe seguir al proceso de decodificación para la detección de errores. No es necesario observar esta precaución si se utilizan aleatorizadores aditivos (es decir, que no sean de sincronización automática).

3 Bits de servicio

3.1 *Numeración de los bloques*

Los cuatro bits de servicio al principio de cada bloque transmitido en línea indican el orden de sucesión de los bloques y dan una información de control independiente de la información contenida en el mensaje. Una de estas informaciones de control permite controlar el orden de sucesión de los bloques de información durante las repeticiones, garantizando así que no se pierde, se añade o se transpone información alguna. Se emplean cíclicamente tres indicadores A, B, y C del orden de sucesión de los bloques.

Cualquier indicador del orden de sucesión asociado a un bloque de información, se mantiene unido al bloque hasta la recepción correcta de éste. El examen de este indicador es un elemento adicional del proceso de control.

3.2 *Atribución de los bits de servicio*

La atribución de las 16 combinaciones posibles de los cuatro bits de servicio se indica en los cuadros 2/V.41 y 3/V.41. El cuadro 2/V.41 indica las combinaciones esenciales y, por lo tanto, obligatorias, y el cuadro 3/V.41 las combinaciones facultativas.

CUADRO 2/V.41

Combinaciones esenciales

Grupo	Combinación	Función
a	0011	Indicador de un bloque A
b	1001	Indicador de un bloque B
c	1100	Indicador de un bloque C
d	0101	Prefijo de una secuencia de sincronización

Observación — La cifra que aparece primero es la de la izquierda.

CUADRO 3/V.41

Combinaciones facultativas

Grupo	Combinación	Función
e	0110	Retención de bloque
f	1000	Fin de transmisión (este bloque no contiene datos)
g	0001	Comienzo de mensaje 1 (códigos de cinco unidades)
h	1010	Comienzo de mensaje 2 (códigos de seis unidades)
j	1011	Comienzo de mensaje 3 (códigos de siete unidades)
k	0010	Comienzo de mensaje 4 (código de ocho unidades)
l	0100	Fin de mensaje (este bloque no contiene datos)
m	0111	Escape del enlace de datos (bloque de control general)
n	1101	Para atribuir por acuerdo bilateral
p	1110	
q	1111	
r	0000	

La sincronización es la única función esencial de control que deben realizar los bits de servicio.

El bloque facultativo de *escape de enlace de datos* (control general) contiene datos sobre cuyas particularidades los usuarios pueden ponerse de acuerdo.

Las funciones facultativas suplementarias son las indicaciones de *comienzo de mensaje 1* (para los códigos de cinco unidades), *comienzo de mensaje 2* (para los códigos de seis unidades), *comienzo de mensaje 3* (para los códigos de siete unidades), *comienzo de mensaje 4* (para los códigos de ocho unidades), *fin de mensaje* y *fin de transmisión*.

Pueden atribuirse por acuerdo bilateral cuatro combinaciones suplementarias de bits de servicio.

La parte «información» de los bloques que no contienen datos *(retención, fin de transmisión* y *fin de mensaje)* no tiene una significación particular pero se deben controlar también esos bloques en el receptor.

Cuando no se utilizan las combinaciones facultativas de los grupos g a k, el primer bloque de datos que sigue el paso del estado ABIERTO al CERRADO del circuito *preparado para transmitir*, lleva automáticamente el prefijo del *indicador de secuencia de un bloque A* (grupo a). Los bloques de datos BCABC, etc. se transmiten luego sucesivamente en este orden, a menos que se inserten uno o más bloques de otros tipos.

Cuando se utilizan las combinaciones facultativas de los grupos g a k, el primer bloque de datos lleva como prefijo uno de los *indicadores de comienzo de mensaje* 1, 2, 3 ó 4 (grupos g a k), según el número de bits por carácter que se utilice en la transmisión. Se transmiten luego los bloques de datos ABCAB, etc. Si durante la transmisión se produce una interrupción en un enlace del tipo arrendado, o si la operadora interrumpe la transmisión para pasar al modo conversación, la transmisión se reanudará con el indicador de secuencia siguiente al del último bloque aceptado antes de la interrupción. No es necesario emplear un *indicador de comienzo de mensaje* después de una interrupción de este tipo.

En el caso de conexiones con conmutación, puede ser necesario tomar medidas para asegurarse de que un mensaje interrumpido no va seguido de un nuevo mensaje sin la indicación correspondiente.

4 Método de corrección

El estado binario 1 en el canal de retorno (o canal de supervisión) indica que es necesaria la repetición de información (RQ). Inversamente, el estado binario 0 indica la aceptación de la información transmitida. Las disposiciones que rigen la transmisión y la recepción de estas condiciones se indican a continuación, así como en los § 5 y 6.

4.1 *Secuencia de las operaciones en el transmisor de datos*

El presente punto trata exclusivamente del funcionamiento normal. Las condiciones de arranque y de resincronización se estudian en los § 5 y 6.

Los datos se transmiten bloque por bloque, pero el contenido de cada bloque transmitido se registra, junto con sus bits de servicio, en el transmisor, hasta que se confirme su recepción correcta. Es necesario que la memoria tenga una capacidad mínima de dos bloques.

Durante la transmisión de un bloque, el estado del canal de retorno (circuito 119) se comprueba durante los 45 a 50 milisegundos que preceden a la transmisión del último bit de control. Si en este periodo se recibe una petición RQ, se anula el bloque invirtiendo este último bit. El transmisor repite entonces la transmisión a partir del bloque anterior, recurriendo a la memoria. Durante la transmisión del bloque que sigue a la detección de la señal RQ, se hace caso omiso del estado del canal de retorno.

4.2 *Procedimiento seguido en el receptor de datos*

En funcionamiento normal, se mantiene el estado binario 0 en el canal de retorno mientras los bloques se reciben con sus bits de control correctos y con las combinaciones de bits de servicio admisibles. Los datos contenidos en estos bloques se transfieren a la salida del receptor. Como no puede verificarse un bloque en tanto no se haya recibido por completo, si se requiere una salida «limpia» hay que prever una memoria con una capacidad mínima de un bloque.

Si en la recepción un bloque no responde a las condiciones de protección contra errores, se transmite un 1 binario por el canal de retorno y se registra en el receptor la combinación de bits de servicio esperada.

Normalmente, el primer bloque de datos recibido en el ciclo de repetición con bits de control correctos contendrá también una combinación de bits de servicio admisible, y los datos que contiene pueden ser procesados. Sin embargo, puede ocurrir que el primer bloque cuyo control es correcto contenga una combinación anormal de bits de servicio, debida a un error de transmisión por el canal de retorno (lo que provoca la mutilación o la

imitación de una señal 0 binario). En ambos casos se descartan los datos del primer bloque. Si el control de este bloque es correcto pero la combinación de bits de servicio indica que se trata del bloque anterior al que se espera, se transmite por el canal de retorno un 0 binario.

Después de comprobar que el bloque siguiente es correcto y que contiene una combinación de bits de servicio admisible, se pueden procesar sus datos y volver al funcionamiento normal. Si la combinación de bits de servicio indica que un bloque es incorrecto, se transmite un 1 binario; además, si la combinación de bits de servicio corresponde al bloque siguiente al deseado, ello significa que se ha imitado un 0 binario durante todo el periodo de 45 ms especificado en el § 4.1, en cuyo caso debe accionarse una alarma, ya que no se puede salir automáticamente de esta situación, por lo demás poco frecuente.

5 Métodos de arranque

5.1 *Métodos aplicables en el transmisor y secuencia de sincronización*

En el intervalo entre las indicaciones *petición de transmitir* y *preparado para transmitir*, el modem transmite condiciones de línea en reposo (1 binario). Cuando el modem está preparado para transmitir, las primeras señales de datos que se transmiten son el prefijo de secuencia de sincronización (0101), seguido de señales de relleno para sincronización y de la secuencia de sincronización. El relleno puede ser de cualquier longitud, siempre que comprenda por lo menos 28 transiciones y no incluya la secuencia de sincronización. Esta secuencia de sincronización es 0101000010100101, comenzando por la cifra de la izquierda (véase una posible formación en el apéndice I). Las 28 transiciones están destinadas a la sincronización de los bits. Estas señales de sincronización van seguidas del *bloque A* o de una combinación de *comienzo de mensaje* (grupos g a k del cuadro 3/V.41). Durante toda esta secuencia, desde el comienzo del prefijo de sincronización, el transmisor hace caso omiso de la condición del canal de retorno y se comporta como si en éste estuviera presente un 0 binario. La condición del canal de retorno recupera luego su significación normal (véase el § 4). Si la condición correspondiera a un 1 binario durante la verificación del segundo bloque, deberá completarse éste con el último bit (invertido) e iniciarse de nuevo el procedimiento de arranque a partir del comienzo del prefijo de secuencia de sincronización.

5.2 *Métodos aplicables en el receptor de datos*

En el terminal de recepción, se transmite un 1 binario por el canal de retorno hasta la detección de la secuencia de sincronización (0101000010100101), momento en que se transmite un 0 binario y se establece la temporización de los bloques. Las únicas combinaciones de bits de servicio admisibles después de la secuencia de sincronización son el indicador de secuencia del bloque A o, en su caso, un indicador de comienzo de mensaje. Si se reciben otras combinaciones de bits de servicio, se devuelve un 1 binario y se inicia de nuevo la búsqueda de la secuencia de sincronización.

6 Procedimiento de resincronización

6.1 *Restablecimiento del sincronismo*

Si el receptor no consigue identificar, en un plazo razonable, un bloque de información aceptable, debe analizar en forma continua el tren de bits recibidos en busca de la secuencia de sincronización. Una vez hallada ésta, se restablece la temporización de los bloques y se envía el estado 0 binario por el canal de retorno. El procedimiento seguido es idéntico al de arranque, salvo que la combinación esperada de bits de servicio es la que sigue al último indicador de secuencia aceptado.

6.2 *Transmisión de la secuencia de sincronización*

Si el ciclo de repetición normal se produce varias veces seguidas (por lo general de 4 a 8 veces), se supone que es necesaria la resincronización. Se sustituye entonces el ciclo de repetición normal por un ciclo de tres bloques que comprende un bloque de sincronización y los dos bloques repetidos anteriormente. El bloque de sincronización contiene el prefijo de la secuencia de sincronización, el relleno y la secuencia de sincronización descritos en el § 5.1.

Observación – Un relleno corto debería permitir una resincronización más rápida, sobre todo si se utilizan bloques de gran longitud. Sin embargo, el relleno corto tiene el inconveniente de que puede perderse el sincronismo correcto si el ruido imita o altera el prefijo, o si se altera la secuencia de sincronización. Este inconveniente se salva utilizando rellenos más largos, a fin de que el bloque sincronización tenga la misma longitud que el de datos. Se puede elegir entre una u otra longitud, ya que ambas son compatibles.

6.3 *Empleo de bloques de sincronización para retardo en transmisión*

Se puede suspender la transmisión de la información insertando un bloque de sincronización. En el caso de rellenos cortos, conviene que el equipo terminal receptor identifique el prefijo de sincronización y pase inmediatamente al modo de búsqueda de la sincronización, pues de lo contrario se perderá la sincronización.

Cuando el relleno utilizado produce un bloque de longitud normal, es conveniente pasar al modo de búsqueda sin abandonar la temporización de los bloques, transmitiéndose un 0 binario al final del bloque si se identifica el prefijo y los bits de control se ajustan a la secuencia de sincronización.

Puede suceder que el transmisor de datos transmita un ciclo de resincronización antes que el receptor haya pasado al modo de búsqueda de la sincronización. El método que deberá seguir el receptor es idéntico al que acaba de describirse para el empleo de un bloque de sincronización a fin de suspender la transmisión de información.

7 Interfaces

7.1 *Interfaces de los modems*

En el caso normal en que los modems no son parte integrante del equipo terminal de datos, sus interfaces son los indicados por los puntos A-A de las figuras 1/V.41 y 2/V.41. Si se utilizan modems síncronos, se incluirán también en estos interfaces los circuitos de temporización para los elementos de señal apropiados.

7.2 *Interfaces de los equipos terminales de datos*

Si el equipo de protección contra errores (comprendidas las memorias) no es parte integrante del equipo terminal de datos, este equipo se intercala entre el modem y el equipo terminal de datos. Los interfaces del equipo terminal de datos son, en ese caso, los indicados por los puntos B-B y C-C de las figuras 1/V.41 y 2/V.41, respectivamente. Cada uno de estos interfaces lleva incorporado un circuito de temporización para los elementos de señal.

7.2.1 En el equipo terminal transmisor, todos los circuitos de enlace cumplen sus funciones normales, pero el circuito *preparado para transmitir* funciona como sigue, de acuerdo con lo indicado en el último párrafo de la definición dada en la Recomendación V.24.

Circuito preparado para transmitir (véase la figura 1/V.41)

En combinación con el circuito de temporización para los elementos de señal, este circuito informa al equipo terminal de datos del instante en que se piden datos en respuesta al circuito *petición de transmitir*. El circuito *preparado para transmitir* pasa al estado CERRADO cuando se piden datos y al estado ABIERTO cuando no se requieren datos (por regla general, durante la transmisión de los bits de control y de servicio y de cualquier repetición). Este circuito no pasa al estado CERRADO mientras el circuito *petición de transmitir* no pase al estado CERRADO. Todas las transiciones de este circuito coinciden con las transiciones del estado CERRADO al ABIERTO de la temporización para los elementos de señal. En consecuencia, la transición del estado CERRADO al ABIERTO del circuito coincidirá con la transición del estado CERRADO al ABIERTO de la temporización para los elementos de señal durante el 240.°, el 480.°, el 960.° o el 3840.° bit de información de un bloque, según los casos.

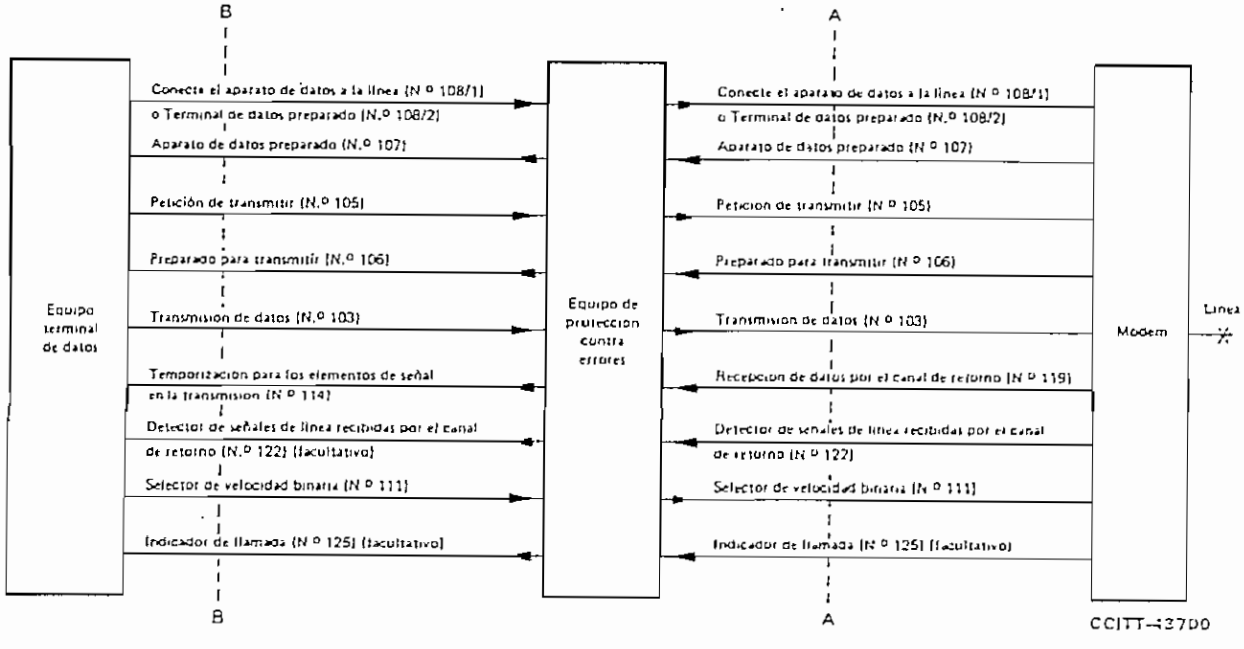


FIGURA 1/V.41

Circuitos de enlace – Equipo terminal de transmisión

7.2.2 En el caso del equipo terminal receptor, se introducen dos nuevos circuitos, pero, dado que dos (o más) circuitos de enlace del modem no se utilizan en este interfaz, el total de circuitos no aumenta. En este interfaz no se dispone del circuito 118 — *transmisión de datos por el canal de retorno.*

Debe preverse una función *preparado para recibir* para informar al equipo de protección contra errores del estado del equipo terminal de datos. Esta función puede ser asegurada por el circuito 108, en cuyo caso la conexión por la red telefónica con conmutación se libera cuando este circuito pasa del estado CERRADO al ABIERTO. Otra solución consiste en prever un circuito de control separado para conservar la conexión a la línea durante los breves periodos en los cuales el equipo terminal de datos no está en condiciones de recibir datos. Este nuevo circuito podría ocupar el lugar del circuito 120 y funcionar de la siguiente manera:

Preparado para recibir (véase la figura 2/V.41)

Sentido: del equipo terminal de datos al equipo de protección contra errores

El equipo terminal de datos mantendrá este circuito en estado CERRADO cuando esté dispuesto a recibir datos. Como el equipo de protección contra errores recibe los datos por bloques, el equipo terminal de datos debe poder recibirlos por bloques. En consecuencia, el equipo terminal de datos únicamente hará pasar este circuito al estado CERRADO si puede aceptar un bloque de datos (de 240, 480, 960 ó 3840 elementos), y le hará volver al estado ABIERTO si no puede aceptar otro bloque en los 15 intervalos elementales que sigan al fin del bloque precedente de datos transferido.

Observación — Si el circuito *preparado para recibir* está en estado ABIERTO, al final de este periodo de 15 intervalos elementales se transmitirá una señal RQ.

El segundo de los nuevos circuitos está encargado de responder a la función preparado para recibir: por lo tanto, es análogo al circuito 121 — *canal de retorno preparado.* Funciona de la siguiente manera:

Datos recibidos presentes (véase la figura 2/V.41)

Sentido: del equipo de protección contra errores al equipo terminal de datos

En combinación con el circuito de temporización para los elementos de señal, este circuito informa al equipo terminal de datos del instante en que se va a dar salida a los datos en respuesta a la indicación *conecte el aparato de datos a la línea*, dada por el equipo terminal de recepción de datos (y, si existe, del circuito *preparado para recibir*), cuando los datos precedentes del otro extremo se consideran correctos. Este circuito pasa al estado CERRADO cuando están a punto de transferirse los datos y permanece en estado ABIERTO el resto del tiempo. Todas las transiciones de este circuito coinciden con las transiciones del estado CERRADO al ABIERTO de la temporización para los elementos de señal. Por lo tanto, la transición del estado CERRADO al ABIERTO en este circuito coincidirá con la transición del estado CERRADO al ABIERTO de la temporización para los elementos de señal correspondiente al 240.º, 480.º, 960.º o 3840.º bit de información de un bloque, según los casos.

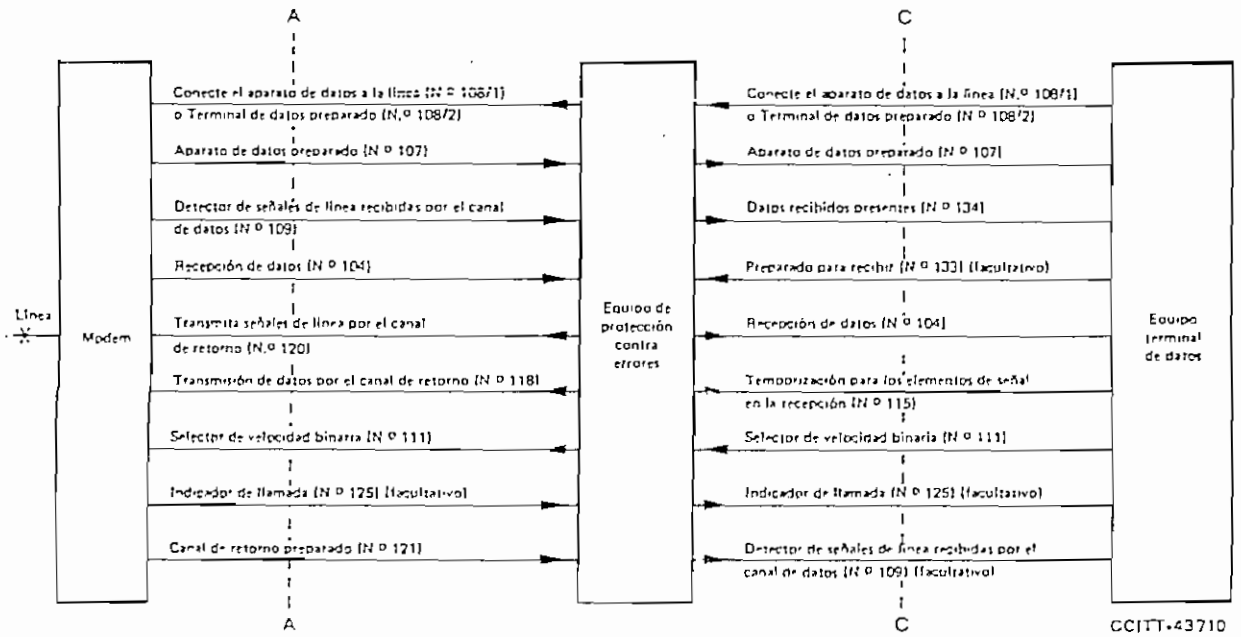


FIGURA 2/V.41

Circuitos de enlace – Equipo terminal de recepción

7.2.3 Pueden preverse otros circuitos de enlace en el interfaz del equipo terminal de datos mediante acuerdos bilaterales entre los usuarios. Los circuitos adicionales pueden servir para introducir funciones de control por bits de servicio distintas de las funciones fundamentalmente previstas. Estos circuitos no deben perturbar el funcionamiento de los circuitos recomendados.

8 Utilización de funciones de control

El *escape del enlace de datos* figura entre las combinaciones facultativas del cuadro 3/V.41, y su uso debe ser objeto de acuerdo entre los usuarios. Por ejemplo, puede servir para indicar a la estación receptora que la estación transmisora desea hablar por esa conexión. En este caso, el equipo receptor hará funcionar un timbre u otro dispositivo de llamada similar y transferirá la línea, del modem a un teléfono. Otra solución consiste en ordenar la impresión de un breve mensaje de teleimpresor destinado al operador.

Se considera que la función *fin de transmisión* da al receptor una indicación precisa de que ha terminado la transmisión y de que puede liberarse la conexión. Según otra solución, el equipo terminal de datos interpreta los datos recibidos para saber en qué momento puede liberar la conexión.

Los indicadores facultativos de comienzo de mensaje y el indicador de fin de mensaje pueden utilizarse para el encaminamiento de los mensajes hacia diferentes destinos o hacia un equipo terminal del extremo receptor, eventualmente con selección del equipo apropiado al código que se emplee.

No hay necesidad de recurrir al bloque de retención en el transmisor, ya que las secuencias de sincronización pueden emplearse como complementación entre los bloques de información si el equipo terminal de datos no dispone de momento de datos preparados para su transmisión, pero de ser preciso puede utilizársele con este fin.

APÉNDICE I

(a la Recomendación V.41)

Codificación y decodificación en los sistemas de códigos cíclicos

I.1 *Codificación*

La figura I.1/V.41 representa un dispositivo de codificación con registro de desplazamiento. En la codificación, los pasos de memoria están puestos a cero, las puertas A y B están activadas, la puerta C está bloqueada y los k bits de servicio e información se cuentan y se introducen. Aparecerán al mismo tiempo a la salida.

Una vez introducidos los bits, se bloquean las puertas A y B y se activa la puerta C, procediendo el registro a 16 nuevos cómputos. Durante este cómputo, aparecen sucesivamente en la salida los bits de control apropiados.

La composición de la secuencia de sincronización puede hacerse con $k = 4$, siendo los cuatro bits 0101. Se suspende el cómputo mientras dura el relleno de sincronización.

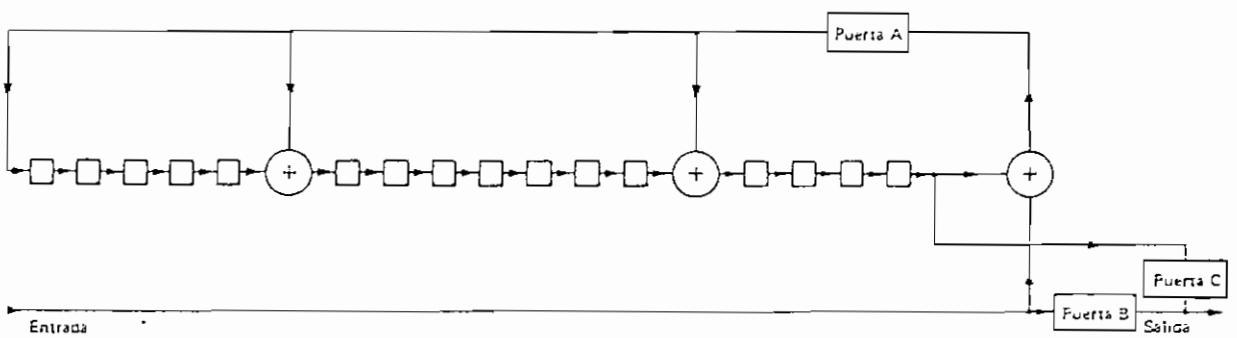


FIGURA I-1/V.41

Dispositivo de codificación

1.2 *Decodificación*

La figura I-2/V.41 representa un dispositivo de decodificación con registro de desplazamiento. Para la decodificación, se activan las puertas A, B y E, se bloquea la puerta D y se ponen a cero los pasos de la memoria.

Los k bits de información o de prefijo se cuentan entonces y se introducen, y se bloquea la puerta B después de k cómputos. A continuación, se cuentan e introducen los 16 bits de control y se examina el contenido de los pasos de la memoria. Este contenido será cero si el bloque no contiene errores. Un contenido distinto de cero indica que el bloque es erróneo.

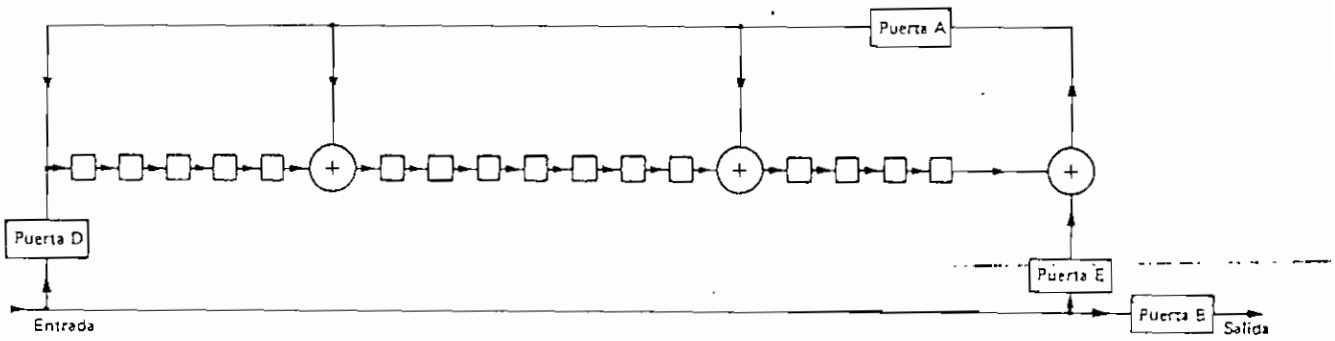


FIGURA I-2/V.41

Dispositivo de decodificación

1.3 *Sincronización en el receptor*

En el caso de una sincronización por bloque, se activa la puerta D (figura I-2/V.41), se bloquean las puertas A, B y E y se examina el registro durante varios intervalos de bit consecutivos para ver si contiene la secuencia de 16 bits requerida. Una vez identificada esta secuencia, el registro y el contador se ponen a cero y se prosigue normalmente la decodificación.

Referencias

[1] *Measurements on switched and leased telephone lines transmitting data at speeds of 250, 800 and 1000 bauds.* Libro Azul, Tomo VIII, suplemento N.º 22 (edición en francés y en inglés), UIT, Ginebra, 1964.

REGISTROS DEL 8250 UART

Este apéndice* describe los registros del chip 8250 UART usado en el puerto serie del IBM PC. Los registros del 8250 son accesibles mediante lecturas o escrituras en los puertos de entrada/salida 3F8-3FF (COM1) o en los puertos 2F8-2FF (COM2). Los registros en los que se puede leer y escribir son listados como de lectura/escritura (read/write - R/W); aquellos de los que únicamente se puede leer son listados como de sólo-lectura (read-only - R/O); y aquellos en los que únicamente se puede escribir, como de sólo-escritura (write-only - W/O).

Registro de control de la línea (LCR) 3FB R/W

El LCR especifica el formato del carácter y controla el acceso a otros registros tal y como se muestra en la Tabla H-1.

Mapa de bits del LCR

7	6	5	4	3	2	1	0
DLAB	SET BREAK	STICK PARITY	EPS	PEN	STB	WSL1	WSL0

* Reimpreso, con permiso de Hayes Microcomputer Products, del *Hayes 1200B Hardware Reference Manual*, 4-2,4-9.

Tabla II-1 Definiciones de los bits del registro de control de la línea

Bit	*Nombre*	*Función*
0	WLS0 Bit 0 de selección de longitud de palabra	Establece la longitud de cada carácter
1	WLS1 Bit 1 de selección de longitud de palabra	Bit 1 / Bit 0 / Longitud de palabra 0 / 0 / 5 bits 0 / 1 / 6 bits 1 / 0 / 7 bits 1 / 1 / 8 bits Se requieren al menos 7 bits para los caracteres ASCII.
2	STB Número de bits de stop	1 2 bits de stop por carácter (1 1/2 sólo para palabras de 5 bits). 0 1 bit de stop por carácter.
3	PEN Capacitación de paridad	1 Capacita la generación de paridad o el chequeo. 0 Descapacita la misma.
4	EPS Selector de paridad par	1 Paridad par. 0 Paridad impar. El bit 3 debe ser 1 lógico.
5	Stick Parity	No usado normalmente.
6	Set Break	1 Provoca que el modem transmita una señal de break continuamente. 0 Para la señal de break.
7	DLAB Bit de acceso al latch del divisor	1 Capacita el acceso a los registros latch del divisor del generador de razón de baudios durante una operación de lectura o de escritura. 0 Capacita el acceso a los registros buffer de recepción, de mantenimiento de transmisión y de capacitación de interrupciones.

Latch del divisor (DLL)
3F8 R/W (byte menos significativo)

El DLL contiene los ocho bits menos significativos del divisor de razón de baudios usado para configurar el Generador de Razón de Baudios. El byte más significativo está contenido en la siguiente dirección de memoria. Juntas, ambas direcciones deben contener el equivalente hexadecimal del divisor prefijado para generar la razón de baudios deseada. El valor hexadecimal a introducir será como sigue:

Para	110 bps	*introducir*	17 hex
	300 bps		80 hex
	1.200 bps		60 hex

Note que para acceder a este registro, el DLAB del LCR debe ser 1.

Latch del divisor (DLM) 3F9 R/W (byte más significativo)

El DLM contiene los ocho bits más significativos del divisor de razón de baudios. Completa el divisor con el valor apropiado:

Para	110 bps	*introducir*	4 hex
	300 bps		1 hex
	1.200 bps		0 hex

La tabla de división se reproduce en el manual IBM *Technical Reference*. El divisor para seleccionar otra razón de baudios (dentro del rango del modem) puede ser obtenido mediante la fórmula:

$$\text{Divisor} = \frac{115.200}{\text{Razón de baudios}}$$

Note que para acceder a este registro, el DLAB del LCR debe ser 1.

Registro de estado de la línea (LSR) 3FD R/O (el bit 0, R/W)

El LSR proporciona información (mostrada en la Tabla H-2) sobre el estado de la transferencia de datos y de las condiciones de error asociadas. Las condiciones señaladas por los bits 0-5 producen una interrupción, suponiendo que las interrupciones están capacitadas (*ver* "Registro de capacitación de interrupciones").

Mapa de bits del LSR

7	6	5	4	3	2	1	0
—	TSRE	THRE	BI	FE	PE	OE	DR

Registro de control del modem (MCR) 3FC R/W

El MCR maneja el interfase con el modem. Los bits del MCR se definen en la Tabla H-3.

Mapa de bits del MCR

7	6	5	4	3	2	1	0
—	—	—	LOOP	OUT 2	OUT 1	RTS	DTR

Tabla 11-2. Definiciones de los bits del registro de estado de la línea

Bit	*Nombre*	*Función*
0	DR Datos listos	1 Indica que se ha recibido un carácter y que está contenido en el registro buffer de recepción (*ver* "Registro buffer de recepción"). Puesto a 0 por la CPU cuando el dato es leído; también puede ser escrito bajo control de programa.
1	OE Error de solapamiento	1 Indica que el carácter del registro buffer de recepción no fue leído antes de la recepción del siguiente carácter, el carácter anterior fue destruido Se pone a 0 siempre que se lee el registro de estado de la línea.
2	PE Error de paridad	1 Indica que la paridad del carácter recibido no cuadra con la especificada por el EPS del registro de control de la línea Se pone a 0 siempre que se lee el registro de estado de la línea
3	FE Error de construcción	1 Indica que al carácter recibido le falta un bit de stop válido. Se pone a 0 siempre que se lee el registro de estado de la línea.
4	BI Interrupción del break	1 Indica que se ha recibido una señal de break; no indica cuando finaliza la señal. Se pone a 0 siempre que se lee el registro de estado de la línea.
5	THRE Registro de mantenimiento de transmisión vacío	1 Indica que el UART está preparado para aceptar un nuevo carácter para transmisión. Se pone a 0 cuando se carga el registro de mantenimiento de transmisión (*ver* "Registro de mantenimiento de transmisión").
6	TSRE Registro de desplazamiento de transmisión vacío	1 Indica que el último carácter en el registro de desplazamiento de transmisión ha sido transmitido. Puesto a 0 cuando se transfiere un dato del registro de mantenimiento de transmisión al registro de desplazamiento de transmisión.
7	—	No usado; siempre 0.

Tabla H-3. Definiciones de los bits del registro de control del modem.

Bit	*Nombre*	*Función*
0	DTR Terminal de datos listo	1 Capacita la operación del modem. 0 Descapacita el modem (por ejemplo, el modem nunca acepta órdenes ni autorrespuestas y si está en línea, el modem se desconecta).
1	RTS Petición de emisión	La acción depende del modem.
2	OUT 1 Salida 1	Salida designada por el usuario.
3	OUT 2 Salida 2	Salida designada por el usuario.
4	LOOP	1 Activa la característica de "loopback" para test de diagnóstico del UART. 0 Para operación normal.
5-7	—	No usados; siempre 0.

Registro de estado del modem (MSR) 3FE R/O

El MSR proporciona el estado actual de las señales de control del modem tal y como se definen en la Tabla H-4. Siempre que el bit 2 o el 3 es un 1 lógico, se genera una interrupción de estado de modem si es que está capacitada (*ver* "Registro de capacitación de interrupciones").

Mapa de bits del MSR

7	6	5	4	3	2	1	0
RLSD	RI	DSR	CTS	DRLSD	TERI	DDSR	DCTS

Registro buffer de recepción (RBR) 3F8 R/O

El RBR contiene el carácter que se acaba de recibir. El carácter es recibido serialmente, llegando primero el bit menos significativo (bit 0). Note que para acceder a este registro, el DLAB del LCR debe ser 0.

Tabla H-4. Definiciones de los bits del registro de estdo del modem.

Bit	*Nombre*	*Función*
0	DCTS Delta libre para emitir	Siempre 0.
1	DDSR Listo para operación (delta)	Siempre 0.
2	TERI Rastro del indicador de timbre	1 Indica que el indicador de timbre (bit 6) ha cambiado de 1 lógico a 0 logico. Puesto a 0 cuando se lee el MSR.
3	DRLSD Rastro del detector de señal	1 Indica que el bit de señal detectada en la línea de recepción ha cambiado de estado. Puesto a 0 cuando el MSR es leído.
4	CTS Libre para emitir	Siempre 1.
5	DSR Listo para operación	Siempre 1.
6	RI Indicador de timbre	1 Indica que el modem detecta una señal de timbre en la línea telefónica.
7	RLSD Señal detectada en la línea de recepción	1 Indica que el modem detecta una señal de timbre en la línea telefónica. 0 Indica ausencia de señal de portador.

Registro de mantenimiento de transmisión (THR) 3F8 W/O

El THR contiene el siguiente carácter a ser transmitido serialmente. El bit menos significativo (bit 0) es el que primero se transmite. Note que para acceder a este registro, el DLAB del LCR debe ser 0.

Registro de capacitación de interrupciones (IER) 3F9 R/W

El IER capacita las fuentes de interrupción para que puedan activar la señal de interrupción saliente. Los bits del IER se definen en la Tabla H-5.

Mapa de bits del IER

7	6	5	4	3	2	1	0
—	—	—	—	EDSSI	ELSI	ETBEI	ERBFI

Tabla H-5. Definiciones de los bits del registro de capacitación de interrupciones.

Bit	*Nombre*	*Función*
0	ERBFI Capacitación de interrupción por dato recibido disponible	1 Provoca que el UART genere una interrupción siempre que el bit DR del registro de estado de la línea se ponga a 1 lógico.
1	ETBEI Capacitación de interrupción por registro de mantenimiento de transmisión vacío	1 Produce que el UART genere una interrupción siempre que el THRE del registro de estado de la línea se ponga a 1 lógico.
2	ELSI Capacitación de interrupción por estado de recepción de la línea	1 Produce que el UART genere una interrupción siempre que ocurra un error de solapamiento, un error de paridad, un error de construcción o una interrupción de break; ver el registro de estado de la línea.
3	EDSSI Capacitación de interrupción por estado del modem	1 Produce que el UART genere una interrupción siempre que el RI o el RLSD del registro de estado del modem se ponga a 1 lógico.
4-7	—	No usados; siempre 0.

Note que para acceder a este registro, el DLAB del LCR debe ser 0.

Registro de identificación de interrupción (IIR) 3FA R/O

El IIR guarda información que informa si una interrupción prioritaria está pendiente. Los bits del IIR se definen en la Tabla H-6.

Mapa de bits del IIR

7	6	5	4	3	2	1	0
—	—	—	—	—	ID de interrupción	ID de interrupción	Interrupción pendiente

Tabla H-6. Definiciones de los bits del registro de identificación de interrupción

Bit	*Nombre*	*Función*
0	Interrupción pendiente	0 Indica que ha ocurrido una condición de interrupción.
1-2	ID de interrupción	Indica la prioridad de la condición de interrupción. Bit 2 / Bit 1 / Prioridad / Fuente de interrupción 0 / 1 / 1 (la mayor) / Error de solapamiento (OE) o error de paridad (PE) o error de construcción (FT) o interrupción de break (BI) (ver el registro de estado de la línea) 1 / 0 / 2 / Datos listos (DR) (ver el registro de estado de la línea) 0 / 1 / 3 / Registro de mantenimiento de transmisión vacío (THRE) (ver el registro de estado de la línea) 0 / 0 / 4 / Indicador de timbre (RI) o señal detectada en la línea de recepción (RLSD) (ver el registro de estado del modem)
3-7	—	No usados; siempre 0.

Note que la información contenida en los bits del 0 al 2 esta siempre disponible incluso aunque la capacidad de interrupción no haya sido capacitada por el registro de capacitacion de interrupciones (IER).

```
;CRC.ASM


;AREA DEL STACK.

    SSEG SEGMENT STACK
    DB 256 DUP (?)
    SSEG ENDS

;SIMBOLOS:Constantes a utilizarse en las rutinas.

CR          EQU         13
LF          EQU         10
;   Lectura de archivos
NB_L        EQU         13400           ;longitud maxima del mensaje
NB          EQU         1024            ;longitud de un bloque
NB_1        EQU         1027
;   Transmision serial
INTA00      EQU         20h             ;registro de control IRQ del 8259
INTA01      EQU         21h             ;registro de mascara IRQ del 8259
;   Calculo del CRC
CRC_INIC    EQU         0000H           ;valor del CRC inicial
SHFT        EQU         8               ;desplazamientos ciclicos por byte
CRC_IT      EQU         0000H           ;CRC inicial para la tabla
LONG_T      EQU         256             ;longitud de la tabla de CRC's
;   Tiempo de calculo
CNT_REP     EQU         100             ;numero de repeticiones para
                                        ;obtener el tiempo de calculo

;AREA DE DATOS.

    DSEG SEGMENT

;   Datos para la pantalla
VIDEOSEG    DW          0               ;segmento de video
NORM        DB          7               ;video normal
BOLD        DB          0FH             ;video resaltado
REVERS      DB          70H             ;video reverso
SUBR        DB          09H             ;video subrayado
ATRIB       DB          ?
BUFF_PANT   DB          4000 DUP (0)    ;buffer para guardar pantalla

;   Datos para macros de pantalla
PALTEC      DW          0
PDDIR       DW          0
X           DB          0
Y           DB          0

;   Datos para dibujo de lineas
LIN         DB          2 DUP (0)
ESQ_SI      DB          2 DUP (0)
ESQ_SD      DB          2 DUP (0)
```

```
ESQ_II      DB          2 DUP (0)
ESQ_ID      DB          2 DUP (0)

;   Datos para pantalla de presentacion
MENS61      DB          'ESCUELA POLITECNICA NACIONAL',0
MENS62      DB          'FACULTAD DE INGENIERIA ELECTRICA',0
MENS63      DB          'CRC.EXE',0
MENS64      DB          'TRANSMISION SERIAL DE DATOS',0
MENS65      DB          'CON CONTROL DE ERRORES',0
MENS66      DB          'CRC-CCITT',0
MENS67      DB          'Alicia Medina V.',0
MENS68      DB          '1.991',0
MENS69      DB          'Pulse una tecla',0
MENSB7      DB          'E.P.N.',0
MENS99      DB          'Editor.COM en A:',0

;   Datos para ingreso a los menues
NIVEL       DB          0               ;Nivel=0 es nivel inicial
TAB_MENU    DW          MENU_1,MENU_2,MENU_3,MENU_4,MENU_5,MENU_6

;   Datos de procedimientos y teclas de cada menu
TAB_PROC1   DW          FIN,CAM_DIR,CAM_ARCH,MOD_RCP
TEC_PROC1   DB          66H,64H,61H,6DH,0FFH
TAB_PROC2   DW          FIN,EDIT
TEC_PROC2   DB          66H,65H,0FFH
TAB_PROC3   DW          FIN,EDIT,CALC_CRC
TEC_PROC3   DB          66H,65H,72H,0FFH
TAB_PROC4   DW          FIN,COM_SER,IN_ERROR
TEC_PROC4   DB          66H,74H,69H,0FFH
TAB_PROC5   DW          FIN,COM_SER
TEC_PROC5   DB          66H,74H,0FFH
TAB_PROC6   DW          FIN,CAM_DIR,CAM_ARCH,MOD_RCP,COM_SER,IN_ERROR
TEC_PROC6   DB          66H,64H,61H,6DH,74H,69H,0FFH

;   Banderas
FLAG_E      DB          0               ;bandera para edicion
FLAG_C      DB          0               ;bandera para calculo del CRC
FLAG_ERR    DB          0               ;bandera para introducir errores
F_ESC       DB          0               ;bandera de ESC en directorio
F_MRX       DB          0               ;bandera de entrada a MOD_RCP

;   Datos para el menu principal de opciones
MENS1       DB          'TRANSMISION SERIAL DE DATOS',0
MENS2       DB          'CON CONTROL DE ERRORES',0
MENS3       DB          '### CRC-CCITT ###',0
MENS4       DB          'Fin',0
MENS5       DB          'Edicion',0
MENS6       DB          'Calculo del CRC',0
MENS7       DB          'Transmision del mensaje',0
MENS8       DB          'Cambio de Directorio',0
MENS9       DB          'Cambio de Archivo',0
MENS10      DB          'Introduccion de errores',0
```

```
MENS53     DB          'Modo de recepcion',0
MENS11     DB          'Salir del Programa (S/N):',0
MENS15     DB          'DIRECTORIO ACTUAL:',0
MENS16     DB          'ARCHIVO DE MENSAJE:',0

;- Datos para cambio de directorio
D_RAIZ     DB          'A:\',0
TEC_STR    DB          8 DUP (0)
TEC_SA     DB          4 DUP (0)
LONGIN     DB          0
UNID       DB          2 DUP (0)        ;recibe unidad ingresada
DIREC      DB          17 DUP (0)       ;recibe directorio ingresado
ASCZD      DB          'A:',0           ;cadena asciiz de unidad y directorio
           DB          20 DUP (0)
T_UNID     DB          'AaBbCc',0ffh
MENS17     DB          'Unidad (A-C):',0
MENS18     DB          'Directorio:',0
MENS19     DB          'Sendero no Existe',0
MENS79     DB          '  Directorio nuevo. Crear (S/N):  ',0
BLANCOS    DB          '               ',0
BLANCOS1   DB          '                                  ',0
F_ESC1     DB          0
F_B        DB          0
MENS95     DB          'Coloque diskettte con EDITOR y',0
MENS96     DB          'Presione una tecla',0

;   Datos para cambio de archivo
ARCHIVO    DB          8 DUP (0)        ;recibe archivo ingresado
ASCZAR     DB          15 DUP (0)       ;cadena asciiz del archivo
MENS20     DB          'Nombre del Archivo:',0
MENS21     DB          'Archivo nuevo. Crear (S/N):',0
MENSE5     DB          'Error en Cambio de Archivo',0
MENSE80    DB          ' Sendero no existe',0
MENSE81    DB          ' Demasiados archivos',0
MENSE82    DB          ' Acceso denegado',0
MENSE83    DB          ' Archivo no existe',0
MENSE84    DB          ' Error en el codigo de acceso',0
MENSE85    DB          ' Error en el file handle',0
TAB_ARCH   DB          "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
           DB          "abcdefghijklmnopqrstuvwxyz"
           DB          "0123456789!#$%&()@<>'{}|~_^",0ffh
X_COL      DB          0
Y_FIL      DB          0

;   Datos para ingresar al editor
BPX        DW          ?
SPX        DW          ?
SSX        DW          ?
BDS        DW          ?
PROGRAMA   DB           'A:\EDITOR.COM',0

PARAM      DB          PARAMCR-PARAM-1
```

```
ASCMSG    DB        27 DUP (0)        ;archivo que carga el editor
PARAMCR   DB        CR

P1        DW        0
P2        DW        0
P3        DD        0
P4        DD        0
MENS30    DB        'EDICION DEL MENSAJE',0
MENSE1    DB        'ERROR AL MODIFICAR LA MEMORIA',CR,LF,'$'
MENSE3    DB        'MEMORIA INSUFICIENTE',CR,LF,'$'
MENSE4    DB        'ERROR EN DIRECCION DE BLOQUE',CR,LF,'$'

;  Datos para calculo del CRC con los diferentes algoritmos
L_TOT     DW        0                 ;longitud del archivo de mensaje
LONG      DW        0                 ;longitud de los bloques
N_BLOQ    DW        0                 ;numero de bloques
RESTO     DW        0
CRC_M1    DW        0                 ;CRC con metodo1
CRC_M2    DW        0                 ;CRC con metodo2
CRC_M3    DW        0                 ;CRC con metodo3
CRC_B     DW        0                 ;CRC de un bloque temporal
CRC_BLQ   DW        13 DUP (0)        ;almacena CRC de todos los bloques

TABLA_1   DW        512 DUP (?)       ;buffer para tabla de CRC's un byte
BINARIO   DB        8 DUP (?)         ;buffer para cambio hex_binario
PERMUT    DB        N8 DUP (?)        ;buffer de bytes permutados
TABHA     DB        '0','1','2','3','4','5','6','7','8'
          DB        '9','A','B','C','D','E','F'

CRC_M1A   DB        5 DUP (?)
CRC_M2A   DB        5 DUP (?)
CRC_M3A   DB        5 DUP (?)
CNT_RA    DB        6 DUP (?)
LNG_A     DB        6 DUP (?)
MENS22    DB        'CALCULO DEL CRC-CCITT',0
MENS23    DB        'Polinomio: X^16 +X^12 +X^5 +1',0
MENS24    DB        'ALG. BIT A BIT:',0
MENS25    DB        'CRC=     h',0
MENS26    DB        'Tiempo Calc.= ',0
MENS70    DB        'Tiempo Calc.=         mseg',0
MENS27    DB        'ALG.POR BYTE CON TABLA:',0
MENS28    DB        "ALG.POR BYTE 'ON THE FLY':",0
MENS29    DB        'ESC para retornar',0
MENS31    DB        'CALCULO DEL CRC',0
MENS86    DB        ' Espere por Favor ',0
MENS90    DB        '                  ',0
MENS49    DB        'Num. Repeticiones=',0
MENS50    DB        'Long. Bloque=       bytes',0
MENS88    DB        ' Utilizar modo de reemplazo: F6 Ins. ',0
MENS89    DB        '                                     ',0

;  Datos para el tiempo de calculo del CRC
```

```
INT1CH     DD      ?               ;direccion del vector de int 1ch
CONTS      DW      0               ;contador sup. del numero de int.
CONTI      DW      0               ;contador inf. del numero de int.
TIME_I     DW      0
TIME_S     DW      0

HOR        DB      0               ;presentacion del tiempo de calculo
MIN        DB      0               ;para mayores a 600 mseg de CNT_REP
SGDS       DB      0               ;formato: mm:ss:dcm
DEC        DB      0
CEN        DB      0
MIL        DB      0
HORA       DB      10 DUP ('?')
HORA_M1    DB      10 DUP ('?')
HORA_M2    DB      10 DUP ('?')
HORA_M3    DB      10 DUP ('?')
MENS59     DB      'FORMATO DE TIEMPO: mm:ss:dcm',0

RES_M      DW      0               ;presentacion del tiempo en
DEC_1      DB      0               ;formato: milisegundos para
DEC_2      DB      0               ;el calculo una sola vez
HR_M1      DB      7 DUP ('?')
HR_M2      DB      7 DUP ('?')
HR_M3      DB      7 DUP ('?')
F_MIN      DB      0
MENS71     DB      'FORMATO DE TIEMPO: milisegundos',0

;   Datos para introduccion de errores
ARCH_ERR   DB      27 DUP(0)       ;cadena asciiz archivo con errores
NB_AGR1    DW      ?
NB_GR1     DW      ?
BUFF_ERR   DB      NB_L DUP (0)    ;buffer mensaje con errores
MENS46     DB      'EDICION PARA ERRORES',0

;   Datos para transmision serial
MSG_ERR    DB      'COM1 NO INSTALADO',CR,LF
           DB      '$'
BUFF_DAT   DB      1028 DUP (0)    ;buffer de entrada
BUFF_START DW      OFFSET BUFF_DAT ;comienzo buffer de direcciones
BUFF_END   DW      OFFSET BUFF_START;fin buffer de direcciones
BUFF_HEAD  DW      OFFSET BUFF_DAT ;localizacion proxima escritura
BUFF_TAIL  DW      OFFSET BUFF_DAT ;localizacion proxima lectura
UART_ADDR  DW      ?               ;direccion base del UART
INT0CH     DD      ?               ;vector anterior de int 0ch
PARM_INC   DB      0e3h            ;parametros de transmision

LNG_MSG    DW      0
LNG1_MSS   DW      0
NB_ENV     DW      0               ;numero de bytes a enviar
BLQ_TX     DW      0               ;numero de bloque transmitido
BUFF_ENV   DB      NB_1 DUP (?)    ;buffer de bloque a enviar
ACK        DB      06H             ;caracter ACK
```

```
NACK      DB        15H             ;caracter NACK
EOT       DB        04H             ;caracter EOT
F_COM     DB        0

N_CHART   DW        0               ;datos para presentar los
POS_MIXT  DB        5               ;caracteres que se transmiten
POS_MIYT  DB        2               ;en pantalla
POS_MAXT  DB        74
POS_MAYT  DB        10
CURS_XT   DB        ?
CURS_YT   DB        ?

N_CHARR   DW        0               ;datos para presentar los
POS_MIXR  DB        5               ;caracteres que se reciben
POS_MIYR  DB        14              ;en pantalla
POS_MAXR  DB        74
POS_MAYR  DB        22
CURS_XR   DB        ?
CURS_YR   DB        ?

MENS33    DB        ' TRANSMISION ',0
MENS34    DB        ' RECEPCION ',0
MENS35    DB        'ESC Salir al DOS',0
MENS36    DB        'COMUNICACION SERIAL',0

;   Datos para detección de errores
BUFF_REC  DB        NB_1 DUP (?)    ;buffer de bloque recibido
LNG_REC   DW        0               ;longitud del bloque recibido
CRC_REC   DW        0               ;CRC recibido
CRC_RECA  DB        5 DUP (?)
CRC_CR    DW        0               ;CRC calculado en recepción
CRC_CRA   DB        5 DUP (?)
END_TX    DB        0               ;bandera al recibir EOT
MENS37    DB        'DETECCION DE ERRORES',0
MENS38    DB        'CRC RECIBIDO=     h',0
MENS39    DB        'SINDROME DEL MENSAJE RECIBIDO=     h',0
MENS40    DB        'MENSAJE CORRECTO',0
MENS41    DB        'ERROR EN EL MENSAJE',0

;   Datos para grabar el mensaje recibido en un archivo
ARCH_RCP  DB        25 DUP (0)      ;cadena asciiz archivo de recepcion
HAND_RCP  DW        ?
NUM_RCP   DB        0
BUFF_RCP  DB        NB_L DUP (0)    ;buffer del mensaje completo en Rx
NB_AGR    DW        ?               ;número de bytes a grabar
NB_GR     DW        ?               ;número de bytes grabados
ER_GR     DW        0               ;error al grabar
MENSE7    DB        'Error al Crear Archivo',0
MENSE8    DB        'Error al Grabar Archivo',0
MENS43    DB        '"A" Para Archivar Mensaje',0
MENS44    DB        'ARCHIVO DE MENSAJE RECIBIDO: ',0
LST_BLQ   DW        0               ;datos para listado del mensaje
```

```
LST_RESTO  DW        0                ;recibido

;   Datos para retransmision
F_RETX     DB        0                ;registro temporal ACK,NACK
ORD_RTX    DB        0                ;orden de retransmision
ULTIMA     DB        0
ARCH_ORG   DB        27 DUP (0)       ;cadena asciiz archivo original
BUFF_ORG   DB        NB_L DUP (0)     ;buffer archivo mensaje original
HAND_ERR   DW        ?

MENS45     DB        '"R" Para Retransmision',0
MENS47     DB        '"L" Presentacion del Mensaje',0
MENS48     DB        'MENSAJE RECIBIDO',0
MENS52     DB        'ENTER Prox. Pag.',0

;   Datos para funciones con archivos
HANDLE     DW        ?                ;identificador de archivo
ER_ABR     DW        0                ;error al abrir archivo
ER_CR      DW        0                ;error al crear archivo
ACCESO     DB        0

;   Datos para lectura de archivo
ARCH_MSG   DB        25 DUP (0)
NB_ALR     DW        NB_L             ;número bytes a leer
NB_LEC     DW        ?                ;número bytes leidos
ER_LEC     DW        ?                ;error de lectura
BUFF_MSG   DB        NB_L DUP (?)     ;buffer guardar informacion leida
MENSE6     DB        'Error en Lectura del Archivo',0

    DSEG ENDS

IF1
   include mac_crc.asm
   include mac_pant.asm
   include mac_arch.asm
ENDIF

;AREA DE CODIGO DEL PROGRAMA

    CSEG SEGMENT
    ASSUME CS:CSEG,DS:DSEG,SS:SSEG,ES:NOTHING

;   Procedimiento de inicio

START PROC FAR

;ETIQUETA  OPC   OPER                COMENTARIOS

          push  ds                  ;poner direccion de retorno al
          sub   ax,ax               ;DOS en la pila
          push  ax
```

```
        push   ds

        mov    ax,dseg             ;direcciona segmento de datos con
        mov    ds,ax               ;ds

        mov    ax,351ch            ;obtiene y graba el vector
        int    21h                 ;de la interrupcion 1ch
        mov    word ptr int1ch,bx
        mov    word ptr [int1ch+2],es

        assume ds:nothing          ;pone como nuevo vector de
        push   ds                  ;la interrupcion 1ch la dirección
        mov    ax,cs               ;de la rutina CNT_INT.
        mov    ds,ax
        mov    ax,251ch
        mov    dx,offset cnt_int
        int    21h
        pop    ds
        assume ds:dseg

        mov    ax,40h              ;obtiene la dirección base de
        mov    es,ax               ;COM1 del area de datos del BIOS
        mov    ax,word ptr es:[0]
        mov    uart_addr,ax
        or     ax,ax               ;sale si COM1 no esta instalado
        jnz    vector              ;con mensaje de error
        mov    ah,9
        mov    dx,offset msg_err
        int    21h
        mov    ax,4c01h

        int    21h

vector: sti
        mov    ax,350ch            ;obtiene y graba el vector de la
        int    21h                 ;interrupción 0ch
        mov    word ptr int0ch,bx
        mov    word ptr [int0ch+2],es

        assume ds:nothing          ;pone como nuevo vector
        push   ds                  ;de la interrupción 0ch la direccion
        mov    ax,cs               ;de la rutina READ_COM
        mov    ds,ax
        mov    ax,250ch
        mov    dx,offset read_com
        int    21h
        pop    ds
        assume ds:dseg

        mov    ax,40h              ;obtiene dirección del segmento
        mov    es,ax               ;de video y lo graba en videoseg
        mov    ax,es:[63h]
```

```
            mov     dx,0b000h
            cmp     ax,3b4h
            je      dirvid
            add     dx,800h
dirvid:     mov     videoseg,dx
            mov     es,dx

            mac_crc                     ;ejecuta los macros que se
            mac_pant                    ;utilizan en el programa
            mac_arch


            call    pant_pre            ;llama a pantalla de presentacion
            mov     ah,00h              ;y espera una tecla
            int     16h

            mov     dl,0                ;unidad por defecto= A
            mov     ah,0eh
            int     21h
            mov     dx,offset d_raiz    ;inicio en directorio raiz
            mov     ah,3bh
            int     21h
            jnc     progen1
            call    sal_dos


progen1:    mov     bl,nivel            ;valor de nivel para llamar
            xor     bh,bh               ;al menu correspondiente
            shl     bl,1                ;nivel=0 para menu_1 (inicio)
            call    tab_menu.[bx]

            mov     al,flag_err         ;si flag_err=1 llama a EDITAR
            cmp     al,1                ;para introducir errores
            je      editar
            mov     al,flag_c           ;si flag_c=1 va al siguiente nivel
            cmp     al,1                ;luego de calcular el CRC
            je      progen1
            mov     al,flag_e           ;si flag_e=1 llama a EDITAR
            cmp     al,1                ;para mensaje
            je      editar
            jmp     progen1


;   ¡¡¡¡¡   INGRESO AL EDITOR   ¡¡¡¡¡
;   El programa principal entrega el control al editor, y lo recupera
;   cuando se graba el mensaje. El editor utilizado en la tesis es el
;   Norton Editor.

EDITAR:
            guarpant
            borrpant
            ventana 21,20,24,58,norm,3
```

```
          mov   al,flag_err         ;presenta mensajes de acuerdo
          cmp   al,1                ;al nivel de edicion
          je    edit7
          print 29,22,mens30,bold
          jmp   edit8
edit7:    print 29,22,mens46,bold
          print 40,3,mens88,norm
          print 40,2,mens89,norm
          print 40,4,mens89,norm

edit8:    mov   bds,ds              ;salva registros en memoria
          mov   bpx,bp
          mov   spx,sp
          mov   ssx,ss
                                    ;modifica bloques de memoria
          pop   es                  ;es=direccion segmento de programa
          mov   bx,10000            ;bx=nuevo tamano del bloque
                                    ;en parrafos 1 parrafo=16 bytes
          mov   ah,4ah              ;funcion setblock
          mov   al,0
          int   21h

          cmp   ax,8                ;chequea posibles errores
          je    erred3              ;en control de memoria
          cmp   ax,9
          jne   edit1

          mov   ax,seg mense4       ;mensaje de error
          mov   ds,ax
          mov   dx,offset mense4
          mov   ah,09h
          int   21h
          jmp   edit2

erred3:   mov   ax,seg mense3       ;mensaje de error
          mov   ds,ax
          mov   dx,offset mense3
          mov   ah,09h
          int   21h
          jmp   edit2

edit1:                              ;obtiene cadena asciiz del archivo
          lea   si,ascmsg           ;de mensaje
          call  nom_arch

          mov   dx,offset programa  ;direcciona nombre del programa

          mov   p2,offset param     ;construye bloque de parametros
          mov   p2+2,seg param

          mov   ax,seg p1           ;apunta bloque de parametros con
          mov   es,ax               ;es:bx
```

```
          mov   bx,offset pi

          mov   ah,4bh              ;carga y ejecuta el programa
          mov   al,0
          int   21h

edit2:    mov   ax,dseg             ;direcciona segmento de datos
          mov   ds,ax               ;con ds

          mov   bp,bpx              ;restaura registros
          mov   sp,spx
          mov   ss,ssx
          mov   ds,bds

          add   sp,2                ;saltar registro ds en la pila

          mov   al,flag_err         ;coloca valor en nivel para ir
          cmp   al,1                ;al siguiente menu,segun se
          je    edit5               ;encuentre en edicion del mensaje
          mov   al,flag_e           ;o en introduccion de errores
          cmp   al,1
          je    edit3
edit5:    mov   al,4
          mov   nivel,al
          jmp   edit6
edit3:    mov   al,2
          mov   nivel,al
edit6:
          mov   dx,videoseg
          mov   es,dx
          recpant
          jmp   progen1

RET
START ENDP
;   Fin de procedimiento


;   Procedimientos adicionales

;   ***** PANTALLA DE PRESENTACION *****
;   Rutina para la presentacion del programa CRC.EXE

PANT_PRE  PROC  NEAR
          guarda
          lmppant
          call  esquina
          ventana 4,14,21,65,norm,17
          print 26,6,mens61,norm
          print 24,7,mens62,norm
          print 35,11,mens63,revers
          print 25,13,mens64,bold
```

```
            print  28,14,mens65,bold
            print  35,15,mens66,bold
            print  20,19,mens67,norm
            print  54,19,mens68,norm
            lin_h  196,14,4,52,bold
            lin_h  196,14,21,52,bold
            lin_v  179,13,5,16,bold
            lin_v  179,66,5,16,bold
            print  13,4,esq_si,bold
            print  66,4,esq_sd,bold
            print  13,21,esq_ii,bold
            print  66,21,esq_id,bold
            print  55,23,mens69,bold
            print  5,23,mens99,norm
            poscur 24,79
            recupera
            ret
PANT_PRE    ENDP

;   *****    RUTINAS PARA PRESENTAR EL MENU DE CADA NIVEL    *****

;   Rutina para el menu_1 o inicial. Espera por una tecla de
;   procedimiento resaltada en la pantalla y se encamina a la
;   rutina correspondiente.
;   Salida:   Variable NIVEL actualizada

MENU_1      PROC   NEAR
            guarda
            mov    al,0
            mov    f_mrx,al
            call   pantalla_1
men_11:     poscur 24,79
            esp_tec                     ;espera se presione una tecla
            mov    ax,paltec
            xor    ah,ah
            cmp    al,01bh              ;si es ESC llama a FIN
            jne    men_16
            call   fin
            poscur 0,0
            jmp    men_11
men_16:     mov    cl,0
men_12:     inc    cl
            mov    bl,0
            lea    si,tec_proc1         ;SI=desplazamiento de teclas de
men_13:     mov    dl,ds:[si]           ;menu_1
            inc    si
            cmp    al,dl                ;compara letras mayusculas
            je     men_15               ;si es una tecla correcta adelanta
            cmp    dl,0ffh
            je     men_14
            inc    bl
            jmp    men_13
```

```
men_14:   cmp    cl,2              ;si no es tecla correcta regresa
          je     men_11            ;a esperar tecla
          add    al,20h            ;compara letras minusculas
          jmp    men_12
men_15:   xor    bh,bh
          shl    bl,1
          call   tab_proc1.[bx]    ;llama al procedimiento que
                                   ;corresponde a la tecla presionada
          mov    al,nivel          ;encamina al siguiente
          cmp    al,1              ;procedimiento
          je     men_17
          cmp    al,2
          jne    men_11
men_17:   recupera
          ret
MENU_1    ENDP

;   Rutina para presentar en pantalla el menu_1

PANTALLA_1 PROC   NEAR
          guarda
          borrpant
          ventana 1,3,3,30,norm,2

          print   14,1,mens63,bold
          print   4,2,mens67,norm
          print   24,2,mens87,norm

          call    vent_menu
          print   28,12,mens4,bold
          print   28,13,mens5,norm
          print   28,14,mens6,norm
          print   28,15,mens7,norm
          print   28,16,mens8,bold
          print   28,17,mens9,bold
          print   28,18,mens10,norm
          print   28,19,mens53,bold
          ilumlet 0,70,subr
          ilumlet 4,68,subr
          ilumlet 5,65,subr
          ilumlet 7,77,subr
          ventana 8,21,10,57,norm,2
          print   23,8,mens15,bold
          print   42,8,d_raiz,norm
          print   23,9,mens16,bold
          recupera
          ret
PANTALLA_1 ENDP

;   Rutina para menu_2. Espera por una tecla de procedimiento
;   resaltada en la pantalla y se encamina a la rutina que le
;   corresponde
```

```
;   Salida:   Variable NIVEL actualizada
;             Bandera FLAG_E = 1

MENU_2      PROC   NEAR
            guarda
            mov    al,0
            mov    flag_e,al
            mov    flag_err,al
            mov    flag_c,al
            call   pantalla_2
men_21:     poscur 24,79
            esp_tec                     ;espera se presione una tecla
            mov    ax,paltec
            xor    ah,ah
            cmp    al,01bh              ;si es ESC llama a FIN
            jne    men_26
            call   fin
            poscur 0,0
            jmp    men_21
men_26:     mov    cl,0
men_22:     inc    cl
            mov    bl,0
            lea    si,tec_proc2         ;SI=desplazamiento de teclas
men_23:     mov    dl,ds:[si]           ;de menu_2
            inc    si
            cmp    al,dl                ;compara letras mayusculas
            je     men_25               ;si tecla correcta adelanta
            cmp    dl,0ffh
            je     men_24
            inc    bl
            jmp    men_23
men_24:     cmp    cl,2                 ;si no es tecla correcta regresa
            je     men_21               ;a esperar tecla
            add    al,20h               ;compara letras minusculas
            jmp    men_22
men_25:     cmp    bl,1
            je     men_27
            xor    bh,bh
            shl    bl,1
            call   tab_proc2.[bx]       ;llama al procedimiento que
            mov    al,nivel             ;corresponde a la tecla presionada
            cmp    al,2                 ;si nivel sigue en 1 permanece en
            jne    men_21               ;menu_2
men_27:     mov    cl,1                 ;si nivel=2, flag_e=1
            mov    flag_e,cl
            recupera
            ret
MENU_2      ENDP

;   Rutina para presentar en pantalla el menu_2

PANTALLA_2 PROC   NEAR
```

```
          print   28,12,mens4,bold
          print   28,13,mens5,bold
          print   28,14,mens6,norm
          print   28,15,mens7,norm
          print   28,16,mens8,norm
          print   28,17,mens9,norm
          print   28,18,mens10,norm
          print   28,19,mens53,norm
          ilumlet 0,70,subr
          ilumlet 1,69,subr
          ret
PANTALLA_2 ENDP

;   Rutina para menu_3. Espera por una tecla de procedimiento
;   resaltada en la pantalla y se encamina a la rutina que le
;   corresponde
;   Salida:   Variable NIVEL actualizada
;             Bandera FLAG_C = 1

MENU_3    PROC   NEAR
          guarda
          mov    al,0
          mov    flag_e,al
          mov    flag_err,al
          mov    flag_c,al
          call   pantalla_3
men_31:   poscur 24,79
          esp_tec                  ;espera se presione una tecla
          mov    ax,paltec
          xor    ah,ah
          cmp    al,01bh           ;si es ESC llama a FIN
          jne    men_36
          call   fin
          poscur 0,0
          jmp    men_31
men_36:   mov    cl,0
men_32:   inc    cl
          mov    bl,0
          lea    si,tec_proc3      ;SI=desplazamiento de teclas
men_33:   mov    dl,ds:[si]        ;de menu_3
          inc    si
          cmp    al,dl             ;compara con letras mayusculas
          je     men_35            ;si tecla correcta adelanta
          cmp    dl,0ffh
          je     men_34
          inc    bl
          jmp    men_33
men_34:   cmp    cl,2              ;si no es tecla correcta regresa
          je     men_31            ;a esperar tecla
          add    al,20h            ;compara con letras minusculas
          jmp    men_32
men_35:   cmp    bl,1
```

```
            je      men_37
            xor     bh,bh
            shl     bl,1
            call    tab_proc3.[bx]      ;llama al procedimiento que
            mov     al,nivel            ;corresponde a la tecla presionada
            cmp     al,3                ;si nivel sigue en 2 permanece en
            jne     men_31              ;menu_3
            mov     cl,1                ;si nivel=3, flag_c=1
            mov     flag_c,cl
            jmp     men_38
men_37:     mov     cl,1
            mov     flag_e,cl

men_38:     recupera
            ret
MENU_3      ENDP

;   Rutina para presentar en pantalla el menu_3

PANTALLA_3 PROC   NEAR
           print   28,12,mens4,bold
           print   28,13,mens5,BOLD
           print   28,14,mens6,bold
           print   28,15,mens7,norm
           print   28,16,mens8,norm
           print   28,17,mens9,norm
           print   28,18,mens10,norm
           print   28,19,mens53,norm
           ilumlet 0,70,subr
           ilumlet 1,69,subr
           ilumlet 2,82,subr
           ret
PANTALLA_3 ENDP

;   Rutina para menu_4. Espera por una tecla de procedimiento
;   resaltada en la pantalla y se encamina a la rutina que le
;   corresponde
;   Salida:   Variable NIVEL actualizada

MENU_4      PROC   NEAR
            guarda
            mov     al,0
            mov     flag_e,al
            mov     flag_err,al
            mov     flag_c,al
            call    pantalla_4
men_41:     poscur 24,79
            esp_tec                     ;espera se presione una tecla
            mov     ax,paltec
            xor     ah,ah
            cmp     al,01bh             ;si es ESC llama a FIN
            jne     men_46
```

```
            call    fin
            poscur 0,0
            jmp     men_41
men_46:     mov     cl,0
men_42:     inc     cl
            mov     bl,0
            lea     si,tec_proc4        ;SI=desplazamiento de teclas
men_43:     mov     dl,ds:[si]          ;de menu_4
            inc     si
            cmp     al,dl               ;compara con letras mayusculas
            je      men_45              ;si es tecla correcta adelanta
            cmp     dl,0ffh
            je      men_44
            inc     bl
            jmp     men_43
men_44:     cmp     cl,2                ;si no es tecla correcta regresa
            je      men_41              ;a esperar tecla
            add     al,20h              ;compara con letras minusculas
            jmp     men_42
men_45:     xor     bh,bh
            shl     bl,1
            call    tab_proc4.[bx]      ;llama al procedimiento que
            mov     al,flag_err         ;corresponde a la tecla presionada
            cmp     al,1                ;si flag_err=1 salta a retornar
            je      men_47
            mov     al,nivel
            cmp     al,5
            je      men_47
            jmp     men_41
men_47:     recupera
            ret
MENU_4      ENDP

;   Rutina para presentar en pantalla el menu_4

PANTALLA_4 PROC   NEAR
           print   28,12,mens4,bold
           print   28,13,mens5,norm
           print   28,14,mens6,norm
           print   28,15,mens7,bold
           print   28,16,mens8,norm
           print   28,17,mens9,norm
           print   28,18,mens10,bold
           print   28,19,mens53,norm
           ilumlet 0,70,subr
           ilumlet 3,84,subr
           ilumlet 6,73,subr
           ret
PANTALLA_4 ENDP

;   Rutina para menu_5. Espera por una tecla de procedimiento
;   resaltada en la pantalla y se encamina a la rutina que le
```

```
;   corresponde
;   Salida:   Variable NIVEL actualizada

MENU_5      PROC   NEAR
            guarda
            mov    al,0
            mov    flag_e,al
            mov    flag_err,al
            mov    flag_c,al
            call   pantalla_5
men_51:     poscur 24,79
            esp_tec                    ;espera se presione una tecla
            mov    ax,paltec
            xor    ah,ah
            cmp    al,01bh             ;si es ESC llama a FIN
            jne    men_56
            call   fin
            poscur 0,0
            jmp    men_51
men_56:     mov    cl,0
men_52:     inc    cl
            mov    bl,0
            lea    si,tec_proc5        ;SI=desplazamiento de teclas
men_53:     mov    dl,ds:[si]          ;de menu_5
            inc    si
            cmp    al,dl               ;compara con letras mayusculas
            je     men_55              ;si tecla correcta adelanta
            cmp    dl,0ffh
            je     men_54
            inc    bl
            jmp    men_53
men_54:     cmp    cl,2                ;si no es tecla correcta regresa
            je     men_51              ;a esperar tecla
            add    al,20h              ;compara con letras minusculas
            jmp    men_52
men_55:     xor    bh,bh
            shl    bl,1
            call   tab_proc5,[bx]      ;llama al procedimiento que
            mov    al,nivel
            cmp    al,5
            je     men_57
            jmp    men_51
men_57:     recupera
            ret
MENU_5      ENDP

;   Rutina para presentar en pantalla el menu_5

PANTALLA_5 PROC   NEAR
           print   28,12,mens4,bold
           print   28,13,mens5,norm
           print   28,14,mens6,norm
```

```
            print   28,15,mens7,bold
            print   28,16,mens8,norm
            print   28,17,mens9,norm
            print   28,18,mens10,norm
            print   28,19,mens53,norm
            ilumlet 0,70,subr
            ilumlet 3,84,subr
            ret
PANTALLA_5  ENDP

;
MENU_6      PROC    NEAR
            guarda
            mov     al,0
            mov     flag_e,al
            mov     flag_err,al
            mov     flag_c,al
            mov     f_mrx,al
            call    pantalla_6
men_61:     poscur 24,79
            esp_tec                     ;espera se presione una tecla
            mov     ax,paltec
            xor     ah,ah
            cmp     al,01bh             ;si es ESC llama a FIN
            jne     men_66
            call    fin
            poscur 0,0
            jmp     men_61
men_66:     mov     cl,0
men_62:     inc     cl
            mov     bl,0
            lea     si,tec_proc6        ;SI=desplazamiento de teclas
men_63:     mov     dl,ds:[si]          ;de menu_5
            inc     si
            cmp     al,dl               ;compara con letras mayusculas
            je      men_65              ;si tecla correcta adelanta
            cmp     dl,0ffh
            je      men_64
            inc     bl
            jmp     men_63
men_64:     cmp     cl,2                ;si no es tecla correcta regresa
            je      men_61              ;a esperar tecla
            add     al,20h              ;compara con letras minusculas
            jmp     men_62
men_65:     xor     bh,bh
            shl     bl,1
            call    tab_proc6.[bx]      ;llama al procedimiento que
            mov     al,flag_err
            cmp     al,1
            je      men_67
            mov     al,nivel
            cmp     al,0
```

```
          je     men_67
          cmp    al,1
          je     men_67
          cmp    al,2
          je     men_67
          cmp    al,5
          je     men_67
          jmp    men_61

men_67:   recupera
          ret
MENU_6    ENDP

;  Rutina para presentar en pantalla el menu_5

PANTALLA_6 PROC   NEAR
          print   28,12,mens4,bold
          print   28,13,mens5,norm
          print   28,14,mens6,norm
          print   28,15,mens7,bold
          print   28,16,mens8,bold
          print   28,17,mens9,bold
          print   28,18,mens10,bold
          print   28,19,mens53,bold
          ilumlet 0,70,subr
          ilumlet 3,84,subr
          ilumlet 4,68,subr
          ilumlet 5,65,subr
          ilumlet 6,73,subr
          ilumlet 7,77,subr
          ret
PANTALLA_6 ENDP


;  *****   DIBUJO DE LA VENTANA DEL MENU   *****

VENT_MENU  PROC   NEAR
          ventana 11,25,20,53,norm,9
          lin_h  196,25,11,29,norm
          lin_h  196,25,20,29,norm
          lin_v  179,24,12,8,norm
          lin_v  179,54,12,8,norm
          print  24,11,esq_si,norm
          print  24,20,esq_ii,norm
          print  54,11,esq_sd,norm
          print  54,20,esq_id,norm
          ret
VENT_MENU  ENDP

;  Rutina para almacenar en memoria los caracteres de esquinas
;  para dibujo de ventanas
```

```
ESQUINA    PROC   NEAR
           lea    si,esq_si           ;SI=desplazamiento de ESQ_SI
           mov    al,218              ;AL=caracter esq_si
           mov    [si],al
           inc    si
           mov    al,0
           mov    [si],al

           lea    si,esq_sd           ;SI=desplazamiento de ESQ_SD
           mov    al,191              ;AL=caracter esq_sd
           mov    [si],al
           inc    si
           mov    al,0
           mov    [si],al

           lea    si,esq_ii           ;SI=despalzamiento de ESQ_II
           mov    al,192              ;AL=caracter esq_ii
           mov    [si],al
           inc    si
           mov    al,0
           mov    [si],al

           lea    si,esq_id           ;SI=despalzamiento de ESQ_ID
           mov    al,217              ;AL=caracter esq_id
           mov    [si],al
           inc    si
           mov    al,0
           mov    [si],al
           ret
ESQUINA    ENDP

;  ¡¡¡¡¡  PROCEDIMIENTOS DEL MENU    ¡¡¡¡¡

;  ¡¡¡¡¡  FIN  ¡¡¡¡¡
;  Rutina para salir al DOS

FIN        PROC   NEAR
           guarda
           guarpant
           borrpant
           ventana 21,20,24,58,norm,3  ;pide confirmacion de salida
           print   25,22,mens11,bold   ;al DOS
           poscur  51,22
           esp_tec
           mov    ax,paltec
           xor    ah,ah
           cmp    al,73h
           je     fin1                ;si es S llama a la rutina
           cmp    al,53h              ;SAL_DOS. Si no retorna.
           jne    fin2
fin1:      call   sal_dos
fin2:      recpant
```

```
            recupera
            ret
FIN         ENDP

;   Rutina para colocar los vectores de las interrupciones 0ch y 1ch
;   en los valores anteriores al comienzo del programa, donde fueron
;   alterados

SAL_DOS     PROC   NEAR

            mov    ax,251ch             ;coloca el vector anterior para
            lds    dx,[int1ch]          ;la interrupcion 1ch
            int    21h
            mov    ax,250ch             ;coloca el vector anterior para
            lds    dx,[int0ch]          ;la interrupcion 0ch
            int    21h
            lmppant
            mov    ah,4ch
            int    21h
            ret
SAL_DOS     ENDP

;   ¦¦¦¦¦   EDICION   ¦¦¦¦¦
;   La rutina para editar se encuentra dentro del programa principal
;   se la nombra en este punto para tener concordancia con los
;   procedimientos

EDIT        PROC   NEAR
            ret
EDIT        ENDP

;   ¦¦¦¦¦   CAMBIO DE DIRECTORIO    ¦¦¦¦¦
;   Rutina que permite ingresar la unidad y directorio donde se
;   desea grabar el archivo de mensaje.

CAM_DIR     PROC   NEAR
            guarda
            guarpant
            ventana 22,20,24,58,norm,2
            print   22,22,mens17,bold
camd10:     poscur  37,22
            mov     f_b,0
            mov     f_esc,0
            esp_sa  unid,2               ;espera ingreso de la unidad
            cmp     f_esc,1
            je      camd10
            cmp     f_b,1
            jne     camd20
            guarpant
camd20:     print   22,23,mens18,bold
camd12:     poscur  34,23
            mov     f_esc,0              ;F_ESC=0
```

```
            mov     f_esc1,0
            esp_str direc,9             ;espera ingreso de directorio
            mov     cl,longin
            lea     si,asczd            ;SI=desplazamiento de ASCZD
            mov     bx,offset unid
            mov     al,ds:[bx]          ;graba en ASCZD la unidad
            mov     ds:[si],al
            mov     byte ptr ds:[si+1],':'
            mov     byte ptr ds:[si+2],'\'
            inc     si
            inc     si
            inc     si
            cmp     f_esc,1             ;si F_ESC=1 no hay directorio y
            je      camd4               ;salta
            mov     bx,offset direc     ;graba en ASCZD el directorio
camd1:      mov     al,ds:[bx]
            mov     ds:[si],al
            inc     si
            inc     bx
            loop    camd1
            jmp     camd13
camd4:      mov     f_esc1,1
camd13:     mov     al,0                ;aumenta byte de ceros
            mov     ds:[si],al
            mov     dx,offset asczd     ;DS:DX apunta a la cadena ASCZD
            mov     ah,3bh              ;funcion cambio de subdirectorio
            int     21h                 ;activo
            jnc     camd14              ;si no hay error salta

            print   22,24,mens79,norm   ;pide confirmacion para
            poscur  58,24               ;crear directorio nuevo
            esp_tec
            mov     ax,paltec
            cmp     al,73h
            je      camd11
            cmp     al,53h
            je      camd11
            cmp     f_mrx,1
            jne     camd22
            print   34,23,blancos,norm
            borrvent 20,24,39
            jmp     camd12
camd14:     jmp     camd2
camd11:     mov     dx,offset asczd     ;funcion crear directorio
            mov     ah,39h
            int     21h
            jc      camd3
            jmp     camd23
camd22:     jmp     camd3
camd23:     mov     dx,offset asczd     ;DS:DX apunta a la cadena ASCZD
            mov     ah,3bh              ;funcion cambio de subdirectorio
            int     21h                 ;activo
```

```
            jc      camd3

camd2:      cmp     f_b,1
            jne     camd21
            recpant
camd21:     print   42,8,blancos,norm   ;imprime nombre del directorio
            print   42,8,asczd,norm
camd3:      borrvent 20,22,39
            borrvent 20,23,39
            borrvent 20,24,39
            print    42,9,blancos,norm
            recupera
            ret
CAM_DIR     ENDP

;   ***** CAMBIO DE ARCHIVO *****
;   Rutina para ingresar el nombre del archivo de mensaje.Maximo seis
;   caracteres. Si el archivo no existe previamente solicita confirmacion
;   para crearlo.
;   Salida:  Variable NIVEL actualizada

CAM_ARCH    PROC   NEAR
            guarda
            guarpant
            ventana 22,20,24,58,norm,2
            print   22,22,mens20,bold
camar1:     mov    f_esc,0
            poscur  43,22
            esp_str archivo,7           ;espera ingreso de nombre
            cmp    f_esc,1
            je     camar13
            mov    cl,longin            ;del archivo
            lea    si,asczar            ;SI=desplazamiento de ASCZAR
            mov    bx,offset archivo    ;graba en ASCZAR nombre del archivo
camar2:     mov    al,ds:[bx]
            mov    ds:[si],al
            inc    si
            inc    bx
            loop   camar2
            mov    byte ptr ds:[si],'.' ;aumenta extension .MSG al nombre
            inc    si
            mov    byte ptr ds:[si],'m'
            inc    si
            mov    byte ptr ds:[si],'s'
            inc    si
            mov    byte ptr ds:[si],'g'
            inc    si
            mov    al,0                 ;aumenta byte de ceros
            mov    ds:[si],al
            jmp    camar10
camar13:    jmp    camar9
camar10:    lea    si,arch_msg
```

```
            call   nom_arch
            abrir  arch_msg,2,handle,er_abr ;abrir archivo de ASCZAR
            mov    ax,er_abr
            cmp    ax,0                ;salte si no hay error
            je     camar7
            cmp    ax,2                ;archivo no existe?
            jne    camar26              ;salte si error
            print   22,23,mens21,norm   ;mensaje para confirmar creacion
            poscur  50,23               ;del archivo
            esp_tec
            mov    ax,paltec
            cmp    al,73h
            je     camar3
            cmp    al,53h
            je     camar3              ;si es positivo salte a crear
camar9:     cmp    f_mrx,1
            jne    camar25
            print  43,22,blancos,norm
            print  22,23,blancos1,norm
            jmp    camar1
camar7:     jmp    camar4
camar25:    jmp    camar12
camar26:    jmp    camar6
camar3:     mov    er_cr,0
            crear  arch_msg,0,handle,er_cr ;crear archivo nombrado en ASCZAR
            mov    ax,er_cr
            cmp    ax,0                ;verificar posible error
            je     camar27
            cmp    ax,2
            jne    camar28
            print  22,24,mense83,bold
            jmp    camar6
camar28:    cmp    ax,4
            jne    camar20
            print  22,24,mense81,bold
            jmp    camar6
camar20:    cmp    ax,5
            jne    camar21
            print  22,24,mense82,bold
            jmp    camar6
camar27:    cerr   handle,er_cr
            mov    ax,er_cr
            cmp    ax,0
            jne    camar29
            jmp    camar4
camar21:    cmp    ax,6
            jne    camar22
            print  22,24,mense85,bold
camar29:    jmp    camar6
camar22:    cmp    ax,12
            jne    camar23
```

```
            print  22,24,mense84,bold
            jmp    camar6
camar23:    cmp    ax,3
            jne    camar5
            print  22,24,mense80,bold
            jmp    camar6
camar5:     print  22,24,mense5,bold
camar12:    jmp    camar6
camar4:     print  42,9,blancos,norm
            print  43,9,asczar,norm     ;imprime nombre del archivo
            lea    si,arch_msg
            call   nom_arch
            call   lec_msg
            mov    ax,nb_lec
            cmp    ax,0
            je     camar11
            mov    al,2
            mov    nivel,al
            jmp    camar6
camar11:    mov    al,1
            mov    nivel,al
camar6:     borrvent 20,22,39
            borrvent 20,23,39
            recupera
            ret
CAM_ARCH    ENDP
;
;   *****   CAMBIO DE ARCHIVO   *****
;   Rutina para ingresar el nombre del archivo de recepcion. Maximo seis
;   caracteres. Si el archivo no existe previamente solicita confirmacion
;   para crearlo.

CAM_ARCH1   PROC   NEAR
            guarda
            guarpant
            ventana 22,20,24,58,norm,2
            print   22,22,mens20,bold
camaria:    mov    f_esc,0
            poscur  43,22
            esp_str archivo,7           ;espera ingreso de nombre
            cmp    f_esc,1
            je     camar13a
            mov    cl,longin            ;del archivo
            lea    si,asczar            ;SI=desplazamiento de ASCZAR
            mov    bx,offset archivo    ;graba en ASCZAR nombre del archivo
camar2a:    mov    al,ds:[bx]
            mov    ds:[si],al
            inc    si
            inc    bx
            loop   camar2a
            mov    byte ptr ds:[si],'.' ;aumenta extension .MSG al nombre
            inc    si
```

```
            mov   byte ptr ds:[si],'r'
            inc   si
            mov   byte ptr ds:[si],'c'
            inc   si
            mov   byte ptr ds:[si],'p'
            inc   si
            mov   al,0                ;aumenta byte de ceros
            mov   ds:[si],al
            jmp   camar10a
camar13a:   jmp   camar9a
camar10a:   lea   si,arch_rcp
            call  nom_arch
            abrir arch_rcp,2,handle,er_abr ;abrir archivo de ASCZAR
            mov   ax,er_abr
            cmp   ax,0                ;salte si no hay error
            je    camar7a
            cmp   ax,2                ;archivo no existe?
            jne   camar8a              ;salte si error
            print  22,23,mens21,norm  ;mensaje para confirmar creacion
            poscur  50,23             ;del archivo
            esp_tec
            mov   ax,paltec
            cmp   al,73h
            je    camar3a
            cmp   al,53h
            je    camar3a              ;si es positivo salte a crear
camar9a:    cmp   f_mrx,1
            jne   camar12a
            print 43,22,blancos,norm
            print 22,23,blancos1,norm
            jmp   camar1a
camar8a:    jmp   camar5a
camar7a:    jmp   camar4a
camar3a:    crear arch_rcp,0,handle,er_cr ;crear archivo nombrado en ASCZAR
            mov   ax,er_cr
            cmp   ax,0                ;verificar posible error
            je    camar4a
camar5a:    print 22,24,mense5,bold
camar12a:   jmp   camar6a
camar4a:    cerr  handle,er_cr
            print 42,9,blancos,norm
            print 43,9,asczar,norm    ;imprime nombre del archivo
            lea   si,arch_msg
            call  nom_arch
            call  lec_msg
            mov   ax,nb_lec
            cmp   ax,0
            je    camar11a
            mov   al,2
            mov   nivel,al
            jmp   camar6a
camar11a:   mov   al,1
```

```
          mov   nivel,al
camaróa:  borrvent 20,22,39
          borrvent 20,23,39
          recupera
          ret
CAM_ARCH1 ENDP

;  ¡¡¡¡¡  CALCULO DEL CRC Y TIEMPO DE EJECUCION  ¡¡¡¡¡
;  Rutina principal para el calculo del CRC y la obtencion del tiempo
;  con los tres algoritmos para el bloque numero 1 del mensaje.
;  Calculo del CRC para todos los bloques .
;  Entrada:  Cadena asciiz del archivo de mensaje
;  Salida:   CRC para todos los bloques
;            CRC por los tres metodos para el bloque 1
;            Variable NIVEL actualizada
CALC_CRC  PROC  NEAR
          guarda
          guarpant
          borrpant
          ventana 20,20,24,58,norm,3
          print   31,21,mens31,bold
          print   60,2,mens86,norm
          print   60,1,mens90,norm
          print   60,3,mens90,norm

          lea   si,arch_msg
          call  nom_arch
          call  lec_msg

          mov   ax,nb_lec
          sub   ax,1
          mov   l_tot,ax
          cmp   l_tot,1024
          jle   calcrc6
          mov   long,1024
          jmp   calcrc7
calcrc6:  mov   ax,l_tot
          mov   long,ax

calcrc7:  call  crcxi
          call  permutar

          mov   ax,0
          mov   conti,ax
          mov   conts,ax

          mov   cx,cnt_rep
calcrc3:  call  bit
          loop  calcrc3

          cli
          call  time
```

```
            lea   si,hora
            mov   bx,offset hora_m1
            call  prt_time
            call  prt_t1


            lea   si,crc_m1
            mov   bx,offset crc_m1a
            call  ascii

            mov   ax,0
            mov   conti,ax
            mov   conts,ax
            sti
            mov   cx,cnt_rep
calcrc4:    call  byte_tab
            loop  calcrc4

            cli
            call  time
            lea   si,hora
            mov   bx,offset hora_m2
            call  prt_time
            call  prt_t2
            sti

            lea   si,crc_m2
            mov   bx,offset crc_m2a
            call  ascii

            mov   ax,0
            mov   conti,ax
            mov   conts,ax
            sti
            mov   cx,cnt_rep
calcrc5:    call  byte_fly
            loop  calcrc5

            cli
            call  time
            lea   si,hora
            mov   bx,offset hora_m3
            call  prt_time
            call  prt_t3

            lea   si,crc_m3
            mov   bx,offset crc_m3a
            call  ascii

            mov   ax,cnt_rep
            mov   bx,0
            mov   cx,5
```

```
num_1:    mov   byte ptr cnt_ra[bx],' '
          inc   bx
          loop  num_1
          push  bx
          mov   si,10
num_2:    sub   dx,dx
          div   si
          add   dl,'0'
          dec   bx
          mov   byte ptr cnt_ra[bx],dl
          or    ax,ax
          jnz   num_2
          pop   bx
          mov   dl,0
          mov   byte ptr cnt_ra[bx],dl

          mov   ax,long
          mov   bx,0
          mov   cx,5
num_3:    mov   byte ptr lng_a[bx],' '
          inc   bx
          loop  num_3
          push  bx
          mov   si,10
num_4:    sub   dx,dx
          div   si
          add   dl,'0'
          dec   bx
          mov   byte ptr lng_a[bx],dl
          or    ax,ax
          jnz   num_4
          pop   bx
          mov   dl,0
          mov   byte ptr lng_a[bx],dl

          mov   ax,videoseg
          mov   es,ax
          cmp   f_min,0
          je    calcrc14
          call  pant_call
          jmp   calcrc15
calcrc14: call  pant_calc

calcrc15: mov   ax,l_tot
          cmp   ax,1024
          jle   calcrc8
          xor   dx,dx
          mov   bx,1024
          div   bx
          mov   n_bloq,ax
          cmp   dx,0
          je    calcrc9
```

```
            inc     n_bloq
            mov     resto,dx
calcrc9:    mov     long,1024
            lea     si,buff_msg
            mov     cx,n_bloq
            mov     bx,offset crc_blq
calcrc11:   call    fly_blq
            mov     ax,crc_b
            mov     ds:[bx],ax
            inc     bx
            inc     bx
            add     si,1024
            cmp     resto,0
            je      calcrc10
            cmp     cx,2
            jne     calcrc10
            mov     dx,resto
            mov     long,dx
calcrc10:   loop    calcrc11
            jmp     calcrc12
calcrc8:    mov     bx,offset crc_blq
            mov     ax,crc_m3
            mov     ds:[bx],ax
            mov     n_bloq,1

calcrc12:   print   55,23,mens29,bold
            poscur 72,23
calcrc1:    esp_tec
            mov     ax,paltec
            cmp     al,01bh
            je      calcrc2
            jmp     calcrc1
calcrc2:    mov     al,3
            mov     nivel,al
            recupera
            recpant
            ret
CALC_CRC    ENDP


;   Rutina que calcula el CRC con el metodo bit a bit (emulacion de
;   hardware)
;   Entada:   Buffer con los bytes del mensaje permutados (PERMUT)
;             Longitud del bloque (LONG)
;             Valor inicial del CRC (CRC_INIC)
;   Salida:   CRC (CRC_M1)

BIT         PROC  NEAR
            guarda
            mov     ax,crc_inic         ;AX=CRC inicial
            mov     crc_m1,ax           ;CRC_M1=AX
            mov     dx,long             ;DX=LONG
```

```
          cld
          lea    si,permut          ;SI=desplazamiento de PERMUT
salto2:   lodsb
          mov    bl,al              ;BL=byte de mensaje permutado
          mov    cx,shft            ;CX=numero de desp. ciclicos
          push   dx                 ;salva registro DX
          mov    dl,80h             ;DL=10000000
salto1:   test   bl,dl              ;ver si el MSB es 1 o 0
          push   bx                 ;salva registro BX
          jnz    uno
          mov    bx,0               ;si es cero BX=0
          jmp    salto3
uno:      mov    bx,1               ;si es uno BX=1
salto3:   mov    ax,crc_m1          ;AX=CRC_M1
          push   ax                 ;salva registro AX
          and    ax,0001h           ;separar el LSB del CRC
          xor    ax,bx              ;AX=AX xor BX
          test   ax,1               ;bit en AX es 1?
          jnz    nocero
          mov    bx,0000h           ;si es cero BX=0000h
          jmp    desp
nocero:   mov    bx,8408h           ;si es uno BX=8408h
desp:     pop    ax                 ;recupera registro AX (CRC)
          shr    ax,1               ;AX= AX===>1 lugar
          xor    ax,bx              ;AX=AX xor BX
          mov    crc_m1,ax          ;CRC_M1=AX (CRC temporal)
          shr    dl,1               ;DL= DL===>1 lugar
          pop    bx                 ;recupera registro BX
          loop   salto1             ;lazo hasta que CX=0
          pop    dx                 ;recupera registro DX
          dec    dx                 ;DX=DX-1
          loopnz salto2             ;lazo hasta que DX=0
          recupera
          RET
BIT       ENDP

;   Rutina para el calculo del CRC con el metodo de busqueda del CRC
;   de un byte en una tabla
;   Entrada:    Buffer con los bytes del mensaje (BUFF_MSG)
;               Tabla con los CRC de un byte (TABLA_1)
;               Longitud del bloque (LONG)
;               CRC inicial (CRC_INIC)
;   Salida:     CRC (CRC_M2)


BYTE_TAB  PROC   NEAR
          guarda
          cld
          mov    cx,long            ;CX=LONG
          mov    ax,crc_inic        ;AX=CRC inicial
          mov    crc_m2,ax          ;CRC_M2=AX
          lea    bx,buff_msg        ;BX=desplazamiento de BUFF_MSG
```

```
b_tab1:   push   cx                ;guardo registro CX
          sub    ax,ax
          mov    al,[bx]           ;AL=byte del mensaje
          push   bx                ;salva registro BX
          mov    bx,crc_m2         ;BX=CRC_M2
          xor    al,bl             ;AL=AL xor BL
          mov    cx,ax             ;CX=AX
          lea    si,tabla_1        ;SI=desplazamiento de TABLA_1
          add    si,cx             ;CX=indice de la tabla
          add    si,cx
          lodsw                    ;AX=CRC de la tabla
          mov    bx,crc_m2         ;BX=CRC_M2
          mov    cl,8
          shr    bx,cl             ;BX= BX===>8 lugares
          xor    ax,bx             ;AX=AX xor BX
          mov    crc_m2,ax         ;CRC_M2=AX
          pop    bx                ;recupera registro BX
          inc    bx                ;incrementa BX
          pop    cx                ;recupera registro CX
          loop   b_tab1            ;lazo hasta que CX=0
          recupera
          ret
BYTE_TAB  ENDP

;   Rutina para obtener el CRC del buffer de mensaje calculando el CRC
;   de un byte dentro del proceso
;   Entrada:   Buffer con los bytes del mensaje (BUFF_MSG)
;              Longitud del bloque (LONG)
;              CRC inicial (CRC_INIC)
;   Salida:    CRC (CRC_M3)

BYTE_FLY  PROC   NEAR
          guarda
          lea    si,buff_msg       ;SI=desplazamiento de BUFF_MSG
          mov    cx,long           ;CX=LONG
          mov    bx,crc_inic       ;BX=CRC inicial
          mov    crc_m3,bx         ;CRC_M3=BX
b_fly1:   sub    ax,ax
          sub    dx,dx
          lodsb                    ;AL=byte del mensaje
          push   cx                ;salva registro CX
          xor    al,bl             ;AL=AL xor BL
          mov    bx,crc_it         ;BX=CRC_IT
          xor    al,bl             ;AL=AL xor BL
          mov    di,al             ;DL=AL (VECTOR X)

;   la siguiente es la combinacion de los componentes del vector
;   X que dan el CRC de un byte, indicada en la TABLA 3.5

          push   dx                ;salvar DX (vector X)
          push   ax                ;salvar AX (vector X)
          mov    cl,8
```

```
            shl     ax,cl               ;AX=AX <=== 8 lugares (AX=b0-b7)
            mov     cl,3
            shl     dx,cl               ;DX=DX <=== 3 lugares (DX=b5-b12)
            xor     dx,ax               ;DX=DX xor AX (DX=b0-b12)
            pop     ax                  ;recupera registro AX
            and     al,0f0h             ;AL=nibble mas significativo de X
            mov     cl,4
            shr     al,cl               ;AL=AL ===> 4 lugares (AL=b12-b15)
            xor     dx,ax               ;DX=DX xor AX (b0-b15)
            mov     bx,dx               ;BX=DX
            pop     dx                  ;recupera registro DX
            and     dx,0fh              ;DX=nibble menos significativo de X
            xor     bx,dx               ;BX=BX xor DX
            mov     cl,7
            shl     dx,cl               ;DX=DX <=== 7 lugares (DX=b5-b8)
            xor     bx,dx               ;BX=BX xor DX
            mov     cl,5
            shl     dx,cl               ;DX=DX <=== 5 lugares (DX=b0-b3)
            xor     bx,dx               ;BX=BX xor DX
            mov     ax,bx               ;AX=BX
            mov     bx,crc_m3           ;BX=CRC_M3
            mov     cl,8
            shr     bx,cl               ;BX=BX ===> 8 lugares (BX=c0-c7)
            xor     bx,ax               ;BX=BX xor AX
            mov     crc_m3,bx           ;CRC_m3=BX (CRC temporal)
            pop     cx                  ;recupera CX
            loop    b_fly1              ;lazo hasta que CX=0
            recupera
            ret
BYTE_FLY    ENDP

;   Rutina para el calculo del CRC para cada bloque de mensaje
;   utilizando el metodo "on the fly"
;   Entrada:  SI= desplazamiento del bloque de mensaje
;             Longitud del bloque de mensaje (LONG)
;             CRC inicial (CRC_INIC)
;   Salida:   CRC del bloque (CRC_B)

FLY_BLQ     PROC    NEAR
            guarda
            mov     cx,long             ;CX=LONG
            mov     bx,crc_inic         ;BX=CRC_INIC
            mov     crc_b,bx            ;CRC_B=BX
fly_bi:     sub     ax,ax
            sub     dx,dx
            lodsb                       ;AL=byte del mensaje
            push    cx
            xor     al,bl               ;AL=AL xor BL
            mov     bx,crc_it
            xor     al,bl               ;AL=AL xor BL (VECTOR X)

;   el procedimiento es el mismo que en la rutina BYTE_FLY (TABLA 3.5)
```

```
        mov     dl,al
        push    dx
        push    ax
        mov     cl,8
        shl     ax,cl
        mov     cl,3
        shl     dx,cl
        xor     dx,ax
        pop     ax
        and     al,0f0h
        mov     cl,4
        shr     al,cl
        xor     dx,ax
        mov     bx,dx
        pop     dx
        and     dx,0fh
        xor     bx,dx
        mov     cl,7
        shl     dx,cl
        xor     bx,dx
        mov     cl,5
        shl     dx,cl
        xor     bx,dx
        mov     ax,bx
        mov     bx,crc_b
        mov     cl,8
        shr     bx,cl
        xor     bx,ax
;
        mov     crc_b,bx            ;CRC_B=BX (CRC temporal)
        pop     cx                  ;recupera registro CX
        loop    fly_b1              ;lazo hasta que CX=0
        recupera
        ret
FLY_BLQ ENDP

;  Rutina para el calculo de la tabla de las posibles combinaciones
;  de 8 bits utilizando el metodo del vector X
;  Entrada:  CRC inicial (CRC_IT)
;            Longitud de la tabla (LONG_T)
;  Salida:   Buffer con los CRC calculados (TABLA_1)

CRCXI   PROC    NEAR
        guarda
        mov     ax,dseg
        mov     es,ax
        lea     di,tabla_1          ;DI=desplazamiento de TABLA_1
        mov     bl,00h              ;BL=00h valor inicial
        mov     cx,long_t           ;CX=LONG_T
crcxi1: mov     ax,crc_it           ;AX=CRC inicial
        push    cx                  ;salva CX
```

```
          push   bx                  ;salva BX
          sub    dx,dx
          xor    al,bl               ;AL=AL xor BL (AL=VECTOR X)
          mov    ah,0

;   el procedimiento es el mismo que en la rutina BYTE_FLY (TABLA 3.5)

          mov    dl,al
          push   dx
          push   ax
          mov    cl,8
          shl    ax,cl
          mov    cl,3
          shl    dx,cl
          xor    dx,ax
          pop    ax
          and    al,0f0h
          mov    cl,4
          shr    al,cl
          xor    dx,ax
          mov    bx,dx
          pop    dx
          and    dx,0fh
          xor    bx,dx
          mov    cl,7
          shl    dx,cl
          xor    bx,dx
          mov    cl,5
          shl    dx,cl
          xor    bx,dx
;
          mov    ax,bx               ;AX=BX (CRC de un byte)
          stosw                      ;graba el CRC en TABLA_1
          pop    bx                  ;recupera registro BX
          inc    bx                  ;incrementa BX
          pop    cx                  ;recupera registro CX
          loop   crcxi1              ;lazo hasta que CX=0
          recupera
          ret
CRCXI     ENDP

;   Rutina para permutar los bytes del mensaje, es decir cambiar el
;   orden de los bits, de manera que el bit mas significativo quede
;   como el menos significativo
;   Entrada:   Buffer con el mensaje (BUFF_MSG)
;              Buffer para la conversion a binario (BINARIO)
;              Longitud del bloque (LONG)
;   Salida:    Buffer con los bytes de mensaje permutados (PERMUT)

PERMUTAR    PROC   NEAR
          guarda
          mov    cx,long             ;CX=LONG
```

```
            lea   si,buff_msg          ;SI=desplazamiento de BUFF_MSG
            lea   di,binario           ;DI=desplazamiento de BINARIO
            lea   bx,permut            ;BX=apunta a PERMUT
perm1:      push  cx                   ;salva registro CX
            push  si                   ;salva registro SI
            push  bx                   ;salva registro BX
            add   di,7                 ;DI=DI+7
            mov   ch,0
            mov   cl,8                 ;CX=contador de bits
            lodsb                      ;AL=byte del mensaje
            mov   bl,80h               ;BL=10000000
perm2:      test  al,bl                ;el bit en AL es 1?
            jnz   perm3
            mov   dl,0                 ;si es cero DL=0
            jmp   perm4
perm3:      mov   dl,1                 ;si es uno DL=1
perm4:      mov   [di],dl              ;[DI]<== DL
            dec   di                   ;decrementa DI
            shr   bl,1                 ;BL=BL ===> 1 lugar
            loop  perm2                ;lazo hasta que CX=0
            inc   di                   ;incrementa DI
            pop   bx                   ;recupera registro BX
            call  bin_hex              ;llama a rutina BIN_HEX
            pop   si                   ;recupera registro SI
            inc   si                   ;incrementa SI
            pop   cx                   ;recupera registro CX
            loop  perm1                ;lazo hasta que CX=0
            recupera
            ret
PERMUTAR    ENDP

;   Rutina para convertir un numero en binario en hexadecimal
;   Entrada:   Buffer de datos binarios (BINARIO)
;   Salida:    Graba en buffer de datos permutados el byte

BIN_HEX     PROC   NEAR

            push   bx                  ;salva registro BX
            mov    ch,0
            mov    cl,7                ;CX=contador de bits
            cld
            lea    si,binario          ;SI=desplazamiento de BINARIO
            lodsb                      ;AL=primer bit
b_hex1:     shl    al,1                ;AL=AL <=== 1 lugar
            mov    bl,al               ;BL=AL
            lodsb                      ;AL=siguiente bit
            xor    al,bl               ;AL=AL xor BL
            loop   b_hex1              ;lazo hasta que CX=0
            pop    bx                  ;recupera registro BX
            mov    [bx],al             ;[BX] <== AL
            inc    bx
            ret
```

```
BIN_HEX    ENDP

;   Rutina para contar el numero de interrupciones 1ch del TIMER
;   que se producen durante el tiempo de calculo del CRC.
;   Se considera que se producen 18 interrupciones de este tipo en un
;   segundo.

CNT_INT    PROC   FAR
           cli                        ;deshabilita interrupciones
           push   ax                  ;salva registros
           push   bx
           push   ds
           mov    ax,dseg
           mov    ds,ax
           inc    conti               ;incrementa contador inferior
           mov    ax,conti
           cmp    ax,0ffffh           ;si CONTI llega a 0ffffh
           je     cnt_1               ;salte a cnt_1
           jmp    cnt_fin             ;si no llega salte a fin
cnt_1:     mov    conti,0             ;CONTI=0 e incrementa CONTS
           inc    conts
cnt_fin:   pop    ds                  ;recupera registros
           pop    bx
           pop    ax
           sti                        ;habilita interrupciones
           IRET
CNT_INT    ENDP

;   Rutina para obtener el tiempo de calculo con los tres algoritmos
;   en formato de tiempo a partir del numero de interrupciones
;   contabilizadas con CNT_INT,para un numero de veces dado en CNT_REP.
;   Entrada:   Numero de interrupciones (CONTS,CONTI)
;   Salida:    Tiempo de calculo almacenado en memoria
;   Nota:   Se considera que se producen 18 interrupciones por segundo
;   el tiempo es aproximado y para comparaciones cualitativas.

TIME       PROC   NEAR
           guarda
           mov    ax,conti
           mov    time_i,ax           ;TIME_I=CONTI
           mov    ax,conts
           mov    time_s,ax           ;TIME_S=CONTS
           mov    ax,time_i           ;AX=TIME_I
           mov    dx,time_s           ;DX=TIME_S

;   calculo del tiempo para el formato mm:ss:dcm

           mov    bx,32771            ;se obtiene las horas
           div    bx                  ;65535 int. en una hora
           shr    ax,1
           mov    hor,al
```

```
        mov     ax,dx           ;se obtiene los minutos
        mov     dx,0            ;1092 int. en un minuto
        mov     bx,1092
        div     bx
        mov     min,al

        mov     ax,dx           ;se obtiene los segundos
        mov     dx,0            ;18 int. en un segundo
        mov     bx,18
        div     bx
        mov     sgds,al

        mov     ax,dx           ;decimas de segundo
        mov     bx,10
        mul     bx
        mov     dx,0
        mov     bx,18
        div     bx
        mov     dec,al

        mov     ax,dx           ;centesimas de segundo
        mov     bx,10
        mul     bx
        mov     dx,0
        mov     bx,18
        div     bx
        mov     cen,al

        mov     ax,dx           ;milesimas de segundo
        mov     bx,10
        mul     bx
        mov     dx,0
        mov     bx,18
        div     bx
        mov     mil,al

        mov     bh,10           ;conversion a ascii de los minutos
        mov     ah,0            ;se almacena en memoria
        mov     al,min
        div     bh
        add     ax,3030h
        mov     word ptr hora,ax
        mov     hora+2,':'

        mov     ah,0            ;conversion a ascii de los segundos
        mov     al,sgds
        div     bh
        add     ax,3030h
        mov     word ptr hora+3,ax
        mov     hora+5,':'

        mov     ah,0            ;conversion a ascii de las decimas
```

```
            mov     al,dec              ;de segundo
            div     bh
            add     ah,30h
            mov     byte ptr hora+6,ah

            mov     ah,0                ;conversion a ascii de las
            mov     al,cen              ;centesimas de segundo
            div     bh
            add     ah,30h
            mov     byte ptr hora+7,ah

            mov     ah,0                ;conversion a ascii de las
            mov     al,mil              ;milesimas de segundo
            div     bh
            add     ah,30h
            mov     byte ptr hora+8,ah
            mov     al,0
            mov     ds:[hora+9],al

            cmp     min,0               ;si hay minutos la bandera
            je      tim1                ;F_MIN=1
            mov     f_min,1
tim1:       call    time_1
            recupera
            ret
TIME        ENDP

;   Rutina para obtener el tiempo requerido para realizar el calculo
;   del CRC una vez. Presentacion en formato :milisegundos
;   Entrada:    Segundos (SGDS)
;   Salida:     Tiempo de calculo almacenado en memoria

TIME_1      PROC    NEAR
            guarda
            mov     ah,0
            mov     al,sgds             ;AL=SGDS
            mov     bx,1000             ;transforma segundos en miliseg
            mul     bx
            mov     cx,ax
            mov     ah,0
            mov     al,dec              ;transforma decimas en miliseg
            mov     bl,100
            mul     bl
            add     cx,ax
            mov     ah,0
            mov     al,cen              ;transforma centesimas en miliseg
            mov     bl,10
            mul     bl
            add     cx,ax
            mov     ah,0
            mov     al,mil
            add     cx,ax
```

```
          mov   ax,cx               ;AX=tiempo total en milisegundos
          mov   bx,cnt_rep          ;BX=CNT_REP
          div   bx                  ;calcula el tiempo para una sola
          mov   res_m,ax            ;ejecucion en milisegundos
          mov   ax,dx               ;con dos decimales
          mov   bx,10
          mul   bx
          mov   dx,0
          mov   bx,cnt_rep
          div   bx
          mov   dec_1,al
          mov   ax,dx
          mov   bx,10
          mul   bx
          mov   dx,0
          mov   bx,cnt_rep
          div   bx
          mov   dec_2,al
          recupera
          ret
TIME_1    ENDP

;   Rutina para almacenar en memoria el tiempo de calculo para el
;   algoritmo bit a bit en formato: milisegundos
;   Entrada:   RES_M, DEC_1, DEC_2
;   Salida:    Graba en memoria el tiempo (HR_M1)

PRT_T1    PROC  NEAR
          guarda
          mov   ax,res_m            ;transforma a ascii el valor de
          mov   bx,0                ;RES_M
          mov   cx,3
pr_1:     mov   byte ptr hr_m1[bx],' '
          inc   bx
          loop  pr_1
          mov   si,10
pr_2:     sub   dx,dx
          div   si
          add   dl,'0'
          dec   bx
          mov   byte ptr hr_m1[bx],dl
          or    ax,ax
          jnz   pr_2
          mov   byte ptr [hr_m1+3],','
          mov   ah,0
          mov   al,dec_1            ;transforma a ascii el valor de
          mov   bx,4                ;DEC_1
          mov   cx,1
pr_3:     mov   byte ptr hr_m1[bx],' '
          inc   bx
          loop  pr_3
          mov   si,10
```

```
pr_4:   sub   dx,dx
        div   si
        add   dl,'0'
        dec   bx
        mov   byte ptr hr_m1[bx],dl
        or    ax,ax
        jnz   pr_4
        mov   ah,0
        mov   al,dec_2          ;transforma a ascii el valor de
        mov   bx,5              ;DEC_2
        mov   cx,1
pr_5:   mov   byte ptr hr_m1[bx],' '
        inc   bx
        loop  pr_5
        mov   si,10
pr_6:   sub   dx,dx
        div   si
        add   dl,'0'
        dec   bx
        mov   byte ptr hr_m1[bx],dl
        or    ax,ax
        jnz   pr_6
        mov   dl,0              ;aumenta byte de ceros
        mov   byte ptr [hr_m1+6],dl
        recupera
        ret
PRT_T1  ENDP

;  Rutina para almacenar en memoria el tiempo de calculo para el
;  algoritmo por byte con tabla en formato: milisegundos
;  Entrada:   RES_M, DEC_1, DEC_2
;  Salida:    Graba en memoria el tiempo (HR_M2)

PRT_T2  PROC  NEAR
        guarda
        mov   ax,res_m          ;transforma a ascii el valor en
        mov   bx,0              ;RES_M
        mov   cx,3
pr_7:   mov   byte ptr hr_m2[bx],' '
        inc   bx
        loop  pr_7
        mov   si,10
pr_8:   sub   dx,dx
        div   si
        add   dl,'0'
        dec   bx
        mov   byte ptr hr_m2[bx],dl
        or    ax,ax
        jnz   pr_8
        mov   byte ptr [hr_m2+3],','
        mov   ah,0
        mov   al,dec_1          ;transforma en ascii el valor en
```

```
            mov     bx,4                ;DEC_1
            mov     cx,1
pr_9:       mov     byte ptr hr_m2[bx],' '
            inc     bx
            loop    pr_9
            mov     si,10
pr_10:      sub     dx,dx
            div     si
            add     dl,'0'
            dec     bx
            mov     byte ptr hr_m2[bx],dl
            or      ax,ax
            jnz     pr_10
            mov     ah,0
            mov     al,dec_2            ;transforma en ascii el valor en
            mov     bx,5                ;DEC_2
            mov     cx,1
pr_11:      mov     byte ptr hr_m2[bx],' '
            inc     bx
            loop    pr_11
            mov     si,10
pr_12:      sub     dx,dx
            div     si
            add     dl,'0'
            dec     bx
            mov     byte ptr hr_m2[bx],dl
            or      ax,ax
            jnz     pr_12
            mov     dl,0
            mov     byte ptr [hr_m2+6],dl
            recupera
            ret
PRT_T2      ENDP

;   Rutina para almacenar en memoria el tiempo de calculo para el
;   algoritmo por byte "on the fly" en formato: milisegundos
;   Entrada:   RES_M, DEC_1, DEC_2
;   Salida:    Graba en memoria el tiempo (HR_M3)

PRT_T3      PROC    NEAR
            guarda
            mov     ax,res_m            ;transforma en ascii el valor en
            mov     bx,0                ;RES_M
            mov     cx,3
pr_13:      mov     byte ptr hr_m3[bx],' '
            inc     bx
            loop    pr_13
            mov     si,10
pr_14:      sub     dx,dx
            div     si
            add     dl,'0'
            dec     bx
```

```
            mov     byte ptr hr_m3[bx],dl
            or      ax,ax
            jnz     pr_14
            mov     byte ptr [hr_m3+3],','
            mov     ah,0
            mov     al,dec_1            ;transforma en ascii el valor en
            mov     bx,4                ;DEC_1
            mov     cx,1
pr_15:      mov     byte ptr hr_m3[bx],' '
            inc     bx
            loop    pr_15
            mov     si,10
pr_16:      sub     dx,dx
            div     si
            add     dl,'0'
            dec     bx
            mov     byte ptr hr_m3[bx],dl
            or      ax,ax
            jnz     pr_16
            mov     ah,0
            mov     al,dec_2            ;transforma en ascii el valor en
            mov     bx,5                ;DEC_2
            mov     cx,1
pr_17:      mov     byte ptr hr_m3[bx],' '
            inc     bx
            loop    pr_17
            mov     si,10
pr_18:      sub     dx,dx
            div     si
            add     dl,'0'
            dec     bx
            mov     byte ptr hr_m3[bx],dl
            or      ax,ax
            jnz     pr_18
            mov     dl,0
            mov     byte ptr [hr_m3+6],dl
            recupera
            ret
PRT_T3      ENDP

;   Rutina qyue almacenar en memoria el tiempo de calculo para
;   los tres algoritmos en formato: mm:ss:dcm
;   Entrada:  Buffer con los datos de tiempo
;   Salida:   Buffer en el que se almacena el tiempo para cada metodo.

PRT_TIME    PROC    NEAR
            mov     ch,0
            mov     cl,10               ;CX=contador de caracteres
ptime1:     lodsb                       ;lee caracter de un buffer temporal
            mov     ds:[bx],al          ;y lo pasa al buffer adecuado
            inc     bx
            loop    ptime1
```

```
          ret
PRT_TIME  ENDP

;  Rutina para cambiar a ascii una palabra en hexadecimal
;  Entrada:  Palabra en hexadecimal
;  Salida:   Digitos separados, en ascii en un buffer de salida

ASCII     PROC  NEAR
          mov   cl,2
          mov   ch,0
          inc   si
asciil:   lodsb                    ;AL=byte en hexadecimal
          heasc                    ;macro heasc
          mov   ds:[bx],dh         ;separa en dos digitos y los
          inc   bx                 ;guarda en memoria
          mov   ds:[bx],dl
          inc   bx
          dec   si
          dec   si
          loop  asciil
          mov   dl,0
          mov   ds:[bx],dl
          ret
ASCII     ENDP

;  Rutina para presentar la pantalla de calculo del CRC y el tiempo
;  en formato: milisegundos.

PANT_CALC PROC  NEAR
          guarda
          borrpant
          ventana 4,20,9,60,norm,4
          print   30,5,mens22,bold
          print   26,6,mens23,norm
          ventana 11,3,19,76,norm,9
          print   4,13,mens24,bold
          print   33,13,mens25,bold
          print   48,13,mens70,bold
          print   4,15,mens27,bold
          print   33,15,mens25,bold
          print   48,15,mens70,bold
          print   4,17,mens28,bold
          print   33,17,mens25,bold
          print   48,17,mens70,bold
          print   38,13,crc_m1a,norm
          print   38,15,crc_m2a,norm
          print   38,17,crc_m3a,norm
          print   63,13,hr_m1,norm
          print   63,15,hr_m2,norm
          print   63,17,hr_m3,norm
          print   3,23,mens71,norm
          print   35,21,mens50,norm
```

```
          print   48,21,lng_a,norm
          recupera
          ret
PANT_CALC   ENDP

;   Rutina para presentar la pantalla de calculo del CRC y el tiempo
;   en formato: mm:ss:dcm

PANT_CAL1  PROC   NEAR
          guarda
          borrpant
          ventana 4,20,9,60,norm,4
          print   30,5,mens22,bold
          print   26,6,mens23,norm
          ventana 11,3,19,76,norm,9
          print   4,13,mens24,bold
          print   33,13,mens25,bold
          print   48,13,mens26,bold
          print   4,15,mens27,bold
          print   33,15,mens25,bold
          print   48,15,mens26,bold
          print   4,17,mens28,bold
          print   33,17,mens25,bold
          print   48,17,mens26,bold
          print   38,13,crc_m1a,norm
          print   38,15,crc_m2a,norm
          print   38,17,crc_m3a,norm
          print   63,13,hora_m1,norm
          print   63,15,hora_m2,norm
          print   63,17,hora_m3,norm
          print   3,23,mens59,norm
          print   3,21,mens49,norm
          print   21,21,cnt_ra,norm
          print   35,21,mens50,norm
          print   48,21,lng_a,norm
          recupera
          ret
PANT_CAL1   ENDP

;   ¡¡¡¡¡   INTRODUCCION DE ERRORES    ¡¡¡¡¡
;   Rutina que se utiliza para la simulacion de la ocurrencia de errores
;   en la linea de transmision
;   Entrada:   Archivo de mensaje original
;   Salida:    Archivo de mensaje modificado (con errores)
;              Bandera FLAG_ERR=1

IN_ERROR   PROC   NEAR
          guarda
;         guarpant
          mov    ax,dseg
          mov    ds,ax
          mov    es,ax
```

```
            lea   si,arch_msg          ;se obtiene el nombre del archivo
            call  nom_arch             ;de mensaje (cadena asciiz)
            call  lec_msg              ;lectura del archivo

            mov   ax,nb_lec            ;AX=numero de bytes leidos
            mov   nb_agr1,ax           ;NB_AGR1=numero de bytes a grabar
            cld
            lea   si,buff_msg          ;movimiento de los datos de
            lea   di,buff_err          ;BUFF_MSG a BUFF_ERR
            mov   cx,nb_agr1
            rep   movsb
            mov   ax,videoseg
            mov   es,ax
            lea   si,asczar            ;SI=cadena ASCZAR
in_e1:      lodsb                      ;AL=[SI], SI=SI+1
            cmp   al,02eh              ;compara AL con caracter "."
            jne   in_e1
            dec   si                   ;cambiar la extension del nombre
            mov   byte ptr ds:[si],'-' ;del archivo a "-e.msg"
            inc   si
            mov   byte ptr ds:[si],'e'
            inc   si
            mov   byte ptr ds:[si],'.'
            inc   si
            mov   byte ptr ds:[si],'m'
            inc   si
            mov   byte ptr ds:[si],'s'
            inc   si
            mov   byte ptr ds:[si],'g'
            inc   si
            mov   al,0                 ;aumenta byte de ceros
            mov   ds:[si],al

            lea   si,arch_err          ;se pone la cadena asciiz del
            call  nom_arch             ;archivo de mensaje con errores
                                       ;en ARCH_ERR

            crear arch_err,0,hand_err,er_cr ;crear el archivo indicado
            mov   ax,er_cr                  ;en la cadena ARCH_ERR
            cmp   ax,0                 ;verifica posible error al crear
            jne   in_e2

;   grabar el mensaje original en el archivo creado anteriormente

            grabar hand_err,nb_agr1,buff_err,nb_gr1,er_gr
            mov   ax,er_gr             ;verifica posible error
            cmp   ax,0
            jne   in_e3

            cerr  hand_err,er_cr       ;cerrar el archivo
            mov   ax,er_cr             ;verifica posible error
```

```
            cmp     ax,0
            jne     in_e3
            jmp     in_e4
in_e2:      print   22,24,mense7,bold
            jmp     in_e5
in_e3:      print   22,24,mense8,bold
            jmp     in_e5
in_e4:      mov     cl,1
            mov     flag_err,cl          ;FLAG_ERR=1
in_e5:      recupera
            mov     es,videoseg
            ret
IN_ERROR    ENDP

;   ¡¡¡¡¡  TRANSMISION- RECEPCION DE DATOS   ¡¡¡¡¡
;   Rutina principal para la transmision - recepcion de datos. Controla
;   la retransmision de los bloques.
;   Entrada:   Buffer de mensaje a transmitir
;   Salida:    Transmision serial dee datos

COM_SER     PROC   NEAR
            guarda
            mov    ultima,0
            mov    ord_rtx,0
            guarpant
            borrpant
            ventana 21,20,24,58,norm,3
            print   30,22,mens36,bold

            mov    al,parm_inc          ;PARM_INC contiene los parametros
            mov    ah,0                 ;de transmision
            xor    dx,dx                ;inicializa la UART con PARM_INC
            int    14h

            in     al,inta01            ;clear del bit 4 hace no mascarable
            and    al,0efh              ;la interrupcion IRQ4 (COM1)
            out    inta01,al            ;en el registro del 8259

            mov    dx,uart_addr         ;apunta al Line Control Register
            add    dx,3                 ;clear del bit DLAB para habilitar
            in     al,dx                ;las int. de l/e buffer de Rx-Tx
            and    al,07fh
            out    dx,al

            sub    dx,2                 ;set del bit 0 del Interrupt Enable
            mov    al,1                 ;Register para habilitar la int.
            out    dx,al                ;cuando se recibe un dato

            add    dx,3                 ;poner los bits del Modem Control
            mov    al,0bh               ;Register GPO2, RTS y DTR en 1
            out    dx,al
```

```
            lea   si,arch_msg          ;cadena asciiz del archivo de
            call  nom_arch             ;mensaje a transmitir

            call  lec_msg              ;leer el archivo
            mov   ax,nb_lec            ;AX= numero de bytes leidos
            sub   ax,1
            cmp   ax,1024              ;compara longitud con un bloque
            jle   com_s2               ;si es menor o igual salta
            mov   lng_msg,1024         ;LNG_MSG=1024
            mov   cx,n_bloq            ;CX=N_BLOQ
            jmp   com_s3
com_s2:     mov   lng_msg,ax           ;LNG_MSG=NB_LEC
            mov   cx,1
com_s3:     mov   bx,offset crc_blq    ;BX apunta al buffer CRC_BLQ
com_s7:     push  cx                   ;salva CX (numero de bloques)
            cmp   ultima,0             ;ultima es cero?
            je    com_s9
            lea   si,buff_org          ;si es 1 SI=desp. de BUFF_ORG
            jmp   com_s10
com_s9:     lea   si,buff_msg          ;si es 0 SI=desp. de BUFF_MSG
com_s10:    mov   ax,blq_tx            ;AX=BLQ_TX numero de bloque Tx
            mov   dx,1024
            mul   dx
            add   si,ax                ;SI apunta al bloque BLQ_TX

            call  msg_crc
            call  tx_rx
com_s20:    call  esp_ack

            mov   al,f_retx            ;AL=F_RETX bandera de retransmision
            cmp   al,06h               ;si es ACK salta a com_s12
            je    com_s12
            cmp   al,15h               ;si es NACK salta a com_s4
            je    com_s4
            jmp   com_s20
com_s12:    pop   cx                   ;recupera CX
            cmp   n_bloq,1             ;si N_BLOQ=1 salta a com_s11
            je    com_s11
            inc   bx                   ;apunta al CRC correspondiente
            inc   bx
            cmp   resto,0
            je    com_s6               ;si resto=0 salta a com_s6
            cmp   cx,2                 ;si resto=1 compara CX con 2
            jne   com_s6
            mov   dx,resto             ;si CX=2 LNG_MSG=RESTO
            mov   lng_msg,dx
com_s6:     mov   ultima,0             ;con ACK, ULTIMA=0
            inc   blq_tx               ;incrementa BLQ_TX
            mov   ax,0
            mov   n_chart,ax           ;pone en cero los contadores
            mov   n_charr,ax           ;de caracteres Tx y Rx
            mov   ax,offset buff_dat   ;inicializa los buffers
```

```
            mov   buff_head,ax
            mov   buff_tail,ax
            loop  com_s7              ;lazo hasta fin de los bloques
com_s11:    call  final_tx            ;llama a final_tx
            jmp   com_s5              ;va al fin de la rutina
com_s4:     push  bx                  ;salva registro BX (CRC)
            inc   ord_rtx             ;incrementa ORD_RTX
            cmp   ord_rtx,1           ;ORD_RTX es 1?
            jne   com_s8
            lea   si,asczar           ;si es 1 SI=desp. de ASCZAR
ret_1:      lodsb                     ;coloca en la cadena asciiz
            cmp   al,02dh             ;ASCZAR el nombre del archivo
            jne   ret_1               ;de mensaje original
            dec   si
            mov   byte ptr ds:[si],'.'
            inc   si
            mov   byte ptr ds:[si],'m'
            inc   si
            mov   byte ptr ds:[si],'s'
            inc   si
            mov   byte ptr ds:[si],'g'
            inc   si
            mov   al,0                ;aumenta bytes de ceros
            mov   ds:[si],al
            lea   si,arch_org         ;coloca en SI=desp. de ARCH_ORG
            call  nom_arch            ;el nombre del archivo original
            call  lec_org
com_s8:     mov   ultima,1            ;con NACK, ULTIMA=1
            pop   bx                  ;recupera registro BX (CRC)
            mov   ax,0                ;pone en cero los contadores
            mov   n_chart,ax          ;de caracteres Rx y Tx
            mov   n_charr,ax
            mov   ax,offset buff_dat  ;inicializa buffers
            mov   buff_head,ax
            mov   buff_tail,ax
            jmp   com_s7              ;regresa a com_s7 (transmision)

com_s5:     mov   dx,uart_addr        ;salida, se coloca los valores
            add   dx,4                ;anteriores al inicio de la rutina
            in    al,dx               ;en los registros alterados del
            and   al,0f4h             ;UART
            out   dx,al

            sub   dx,3
            xor   al,al
            out   dx,al

            in    al,inta01
            or    al,10h
            out   inta01,al

            mov   al,5
```

```
            mov    nivel,al
            recpant
            jmp    progen1
            recupera
            ret
COM_SER     ENDP

;   ††TRANSMISION††
;   Rutina para anadir al bloque de mensaje el CRC correspondiente
;   Entrada:   SI=desplazamiento del bloque de mensaje
;   Salida:    Buffer con el bloque de mensaje y el CRC (BUFF_ENV)

MSG_CRC     PROC  NEAR
            guarda
            mov    ax,dseg
            mov    ds,ax
            mov    as,ax
            lea    di,buff_env          ;DI=desplazamiento BUFF_ENV
            cld
            mov    cx,lng_msg           ;CX=LNG_MSG
            rep    movsb                ;mueve cadena de SI a DI
            mov    al,ds:[bx]           ;anade al bloque de mensaje
            mov    es:[di],al           ;el CRC apuntado por BX,los
            inc    bx                   ;dos bytes
            inc    di
            mov    al,ds:[bx]
            mov    es:[di],al
            inc    di
            mov    al,1ah               ;coloca como ultimo caracter
            mov    es:[di],al           ;de envio 1ah
            mov    ax,lng_msg
            add    ax,3
            mov    nb_env,ax            ;NB_ENV=LNG_MSG + 3
            recupera
            ret
MSG_CRC     ENDP

;   Rutina para transmision y presentacion en pantalla del mensaje y CRC,
;   ademas chequea si se reciben caracteres que tambien se presentan en la
;   pantalla
;   Entrada:   Numero de bytes a transmitir (NB_ENV)
;              Buffer de datos y CRC (BUFF_ENV)
;   Salida:    AL= caracter a transmitir

TX_RX       PROC   NEAR
            guarda
            call   vent_trans

            mov    ax,nb_env            ;AX=NB_ENV
            cmp    ax,0
            mov    cx,ax
            add    cx,1
```

```
            je      msg_nul
            mov     dx,lng_msg
            add     dx,1
            mov     lngl_msg,dx         ;LNGl_MSG=LNG_MSG + 1
            lea     si,buff_env         ;SI=desplazamiento de BUFF_ENV
tx_loop:    mov     al,[si]             ;AL<==[SI]
            inc     si                  ;incrementa SI
            dec     cx                  ;decrementa CX
            cmp     cx,0                ;si CX=0 salte a tx_fin
            je      tx_fin
            mov     bx,n_chart          ;chequeo del caracter a enviar
            cmp     bx,lng_msg          ; si es un byte de CRC salta a
            je      dist_crc            ;dist_crc
            cmp     bx,lngl_msg
            je      dist_crc
            push    ax
            push    dx
            mov     dl,norm             ;pone atributo para el caracter
            mov     atrib,dl            ;y llama a presentar en pantalla
            call    disp_trans
            pop     dx
            pop     ax
            jmp     trans_car
dist_crc:   push    ax
            push    dx
            mov     dl,bold             ;pone atributo para el caracter
            mov     atrib,dl            ;y llama a presentar en pantalla
            call    disp_trans
            pop     dx
            pop     ax
trans_car:  call    send_char           ;envio del caracter al computador
                                        ;remoto
msg_nul:    mov     ax,buff_tail        ;chequea si buffer Rx esta vacio
            cmp     ax,buff_head
            je      tx_loop
            call    read_char           ;si no esta vacio llama a leer
                                        ;caracter
            push    ax
            push    dx
            mov     dl,norm             ;coloca atributo para el caracter
            mov     atrib,dl            ;y llama a presentar en pantalla
            call    disp_recep
            pop     dx
            pop     ax

            jmp     tx_loop             ;vuelve a transmision

tx_fin:     recupera
            ret
TX_RX       ENDP

;   Rutina para enviar un caracter por el puerto serial
```

```
;   Entrada:   AL=caracter a enviar

SEND_CHAR  PROC   NEAR
           push   ax                  ;salva registro AX
           mov    dx,uart_addr        ;DX apunta al Line Status
           add    dx,5                ;Register
send_loop: in     al,dx               ;lazo mientras el Transmit
           test   al,20h              ;Holding Register esta vacio
           jz     send_loop
           sub    dx,5                ;salida del caracter
           pop    ax
           out    dx,al
           ret
SEND_CHAR  ENDP

;   Rutina para establecer la posicion del cursor para la presentacion en
;   pantalla del caracter trasmitido
;   Entrada:   Numero de caracter transmitido (N_CHART)
;              Posicion anterior del cursor para Tx
;   Salida:    Posicion actual del cursor de Tx
;              Actualiza numero de caracteres transmitidos

DISP_TRANS PROC   NEAR
           push   si
           push   cx
           push   dx
           push   bx
           mov    es,videoseg
           inc    n_chart             ;incrementa N_CHART
           mov    cx,n_chart
           cmp    cx,1                ;si N_CHART=1 salta a disp_tx1
           je     disp_tx1
           mov    dl,pos_mayt         ;chequea dato de la posicion
           mov    cl,curs_yt          ;anterior del cursor con las
           cmp    dl,cl               ;posiciones maximas en X y Y
           je     disp_tx1            ;para establecer nueva posicion
           mov    dl,pos_maxt         ;del cursor
           mov    cl,curs_xt
           cmp    dl,cl
           jne    disp_tx2
           inc    curs_yt
           mov    cl,pos_mixt
           mov    curs_xt,cl
           jmp    disp_tx3
disp_tx1:  ventana 2,3,10,76,norm,9    ;ventana para transmision
           mov    cl,pos_mixt         ;posicion del cursor para
           mov    curs_xt,cl          ;el primer caracter de la
           mov    cl,pos_miyt         ;pantalla
           mov    curs_yt,cl
           jmp    disp_tx3
disp_tx2:  inc    curs_xt
disp_tx3:  poscur curs_xt,curs_yt      ;posiciona el cursor
```

```
            mov     ah,09h              ;escribe en la posicion del
            mov     bl,atrib            ;cursor el caracter en AL
            mov     cx,1
            xor     bh,bh
            int     10h
            pop     bx
            pop     dx
            pop     cx
            pop     si
            ret
DISP_TRANS  ENDP

;Rutina para transmitir el caracter de fin de transmision EOT

FINAL_TX    PROC    NEAR
            guarda
            mov     ax,0                ;pone en cero los contadores
            mov     n_chart,ax          ;de caracteres Tx y Rx
            mov     n_charr,ax
            mov     ax,offset buff_dat  ;inicializa buffers
            mov     buff_head,ax
            mov     buff_tail,ax
            mov     nb_env,2            ;NB_ENV=2
            mov     lng_msg,2
            lea     si,buff_env         ;coloca en el buffer de envio
            mov     al,eot              ;BUFF_ENV los caracteres EOT
            mov     ds:[si],al          ;el de control 1ah
            inc     si
            mov     al,1ah
            mov     ds:[si],al
            call    tx_rx               ;llama a transmision
            recupera
            ret
FINAL_TX    ENDP

;   Rutina para dibujar la ventana de transmision

VENT_TRANS  PROC    NEAR
            mov     es,videoseg
            lmppant
            ventana 1,3,11,76,norm,11
            call    esquina
            lin_h   196,3,1,74,bold
            lin_h   196,3,11,74,bold
            lin_v   179,2,2,9,bold
            lin_v   179,77,2,9,bold

            print    2,1,esq_si,bold
            print    2,11,esq_ii,bold
            print    77,1,esq_sd,bold
            print    77,11,esq_id,bold
```

```
          print   32,1,mens33,bold
          ret
VENT_TRANS ENDP

;   ##RECEPCION##
;   Rutina para leer del buffer de Rx el caracter recibido
;   Salida:   AL=caracter recibido
;             Actualiza buffer de lectura (BUFF_TAIL)

READ_CHAR  PROC   NEAR
no_char:   mov    bx,buff_tail          ;lazo hasta que un caracter
           cmp    bx,buff_head          ;aparezca en el buffer de entrada
           je     no_char               ;serial

           cli                          ;deshabilita interrupciones
           mov    al,[bx]               ;lee un byte del buffer y
           inc    bx                    ;vuelve el tope de lectura al
           cmp    bx,buff_end           ;comienzo si es necesario
           jne    read_exit
           mov    bx,buff_start
read_exit: mov    buff_tail,bx
           sti                          ;habilita interrupciones
           ret
READ_CHAR  ENDP

;   Rutina para manejar las interrupciones generadas por COMi cuando un
;   byte de datos es recibido. El data es leido del registro del buffer
;   de recepcion del UART y se almacena en un buffer circular
;   Salida:   Caracter es insertado en el buffer de recepcion BUFF_DAT

READ_COM   PROC   FAR
           cli
           push   ax
           push   bx
           push   dx
           push   ds

           mov    ax,dseg
           mov    ds,ax
           mov    dx,uart_addr          ;asegura que el bit DLAB es
           add    dx,3                  ;cero para lectura del buffer de Rx
           in     al,dx
           and    al,07fh
           out    dx,al
           mov    dx,uart_addr          ;lee el caracter
           in     al,dx
           mov    bx,buff_head          ;calcula proxima posicion de la
           mov    dx,bx                 ;cabeza de escritura asegurandose
           inc    dx                    ;que el buffer no esta lleno
           cmp    dx,buff_end
           jne    no_wrap
           mov    dx,buff_start
```

```
no_wrap:    cmp     dx,buff_tail
            je      exit_int            ;sale si el buffer esta lleno
            mov     [bx],al             ;inserta caracter en el buffer
            mov     buff_head,dx        ;avanza el puntero de la cabeza
exit_int:   mov     al,20h              ;senal EOI al 8259
            out     inta00,al

            pop     ds
            pop     dx
            pop     bx
            pop     ax
            sti                         ;habilita interrupciones
            iret
READ_COM    ENDP

;   Rutina para establecer la posicion del cursor para la presentacion en
;   pantalla del caracter recibido
;   Entrada:    Numero de caracter recibido (N_CHARR)
;               Posicion anterior del cursor para Rx
;   Salida:     Posicion actual del cursor de Rx
;               Actualiza numero de caracteres recibidos

DISP_RECEP PROC   NEAR
           push   cx
           push   dx
           push   bx
           mov    es,videoseg
           inc    n_charr              ;incrementa N_CHARR
           mov    cx,n_charr
           cmp    cx,1
           je     disp_rx1             ;si es primer caracter salta
           mov    dl,pos_mayr          ;chequea dato de la posicion
           mov    cl,curs_yr           ;del cursor con las maximas de X y
           cmp    dl,cl                ;Y para establecer nueva la posicion
           je     disp_rx1
           mov    dl,pos_maxr
           mov    cl,curs_xr
           cmp    dl,cl
           jne    disp_rx2
           inc    curs_yr
           mov    cl,pos_mixr
           mov    curs_xr,cl
           jmp    disp_rx3
disp_rx1:  ventana 14,3,22,76,norm,9   ;ventana para recepcion
           mov    cl,pos_mixr          ;posicion del cursor
           mov    curs_xr,cl           ;para el primer caracter
           mov    cl,pos_miyr          ;de la ventana de recepcion
           mov    curs_yr,cl
           jmp    disp_rx3
disp_rx2:  inc    curs_xr
disp_rx3:  poscur curs_xr,curs_yr       ;posiciona el cursor
           mov    ah,09h               ;escribe en la posion del cursor
```

```
          mov   bl,atrib            ;el caracter en AL
          mov   cx,1
          xor   bh,bh
          int   10h
          pop   bx
          pop   dx
          pop   cx
          ret
DISP_RECEP ENDP

;  Rutina de espera del asentimiento del computador remoto
;  Salida:  F_RETX=caracter ACK o NACK

ESP_ACK   PROC  NEAR
          guarda
          limpant
          call  vent_recep

          STI                       ;habilita interrupciones
esp_1:    mov   ax,buff_tail        ;lazo mientras el buffer esta
          cmp   ax,buff_head        ;vacio
          je    esp_1
          call  read_char           ;lee caracter
          push  ax
          push  dx
          mov   dl,norm
          mov   atrib,dl
          call  disp_recep          ;presenta en pantalla el caracter
          pop   dx
          pop   ax
          mov   f_retx,al           ;F_RETX=caracter recibido
          recupera
          ret
ESP_ACK   ENDP

;  Rutina para dibujar la ventana de recepcion

VENT_RECEP PROC  NEAR
          mov   es,videoseg
          ventana 13,3,24,76,norm,11
          lin_h  196,3,13,74,bold
          lin_h  196,3,23,74,bold
          lin_v  179,2,14,9,bold
          lin_v  179,77,14,9,bold
;
          print   2,13,esq_si,bold
          print   2,23,esq_ii,bold
          print   77,13,esq_sd,bold
          print   77,23,esq_id,bold
;
          print   34,13,mens34,bold
          ret
```

```
VENT_RECEP ENDP

;  ¡¡¡¡¡  MODO DE RECEPCION    ¡¡¡¡¡
;  Rutina principal para la recepcion del mensaje y la deteccion de
;  errores. Inicializa el puerto serial para transmision y recepcion

MOD_RCP    PROC   NEAR
           guarda
           guarpant
           mov    al,0
           mov    end_tx,al
           mov    num_rcp,al
           mov    f_mrx,1
           call   cam_dir
           call   cam_archi
           mov    al,parm_inc          ;PARM_INC contiene los parametros
           mov    ah,0                 ;de transmision
           xor    dx,dx                ;inicializa el UART con PARM_INC
           int    14h

           in     al,inta01            ;clear del bit 4 hace no mascarable
           and    al,0efh              ;la interrupcion IRQ4 (COM1)
           out    inta01,al            ;en el registro del 8259

           mov    dx,uart_addr         ;apunta al Line Control Register
           add    dx,3                 ;clear del bit DLAB para habilitar
           in     al,dx                ;las int. de l/e buffer de Tx-Rx
           and    al,07fh
           out    dx,al

           sub    dx,2                 ;set del bit 0 del Interrupt Enable
           mov    al,1                 ;Register para habilitar la int.
           out    dx,al                ;cuando se recibe un dato

           add    dx,3                 ;poner los bits del Modem Control
           mov    al,0bh               ;Register GPO2, RTS y DTR en 1
           out    dx,al

           call   recepcion

           mov    dx,uart_addr         ;restaura valores anteriores de
           add    dx,4                 ;los registros del UART
           in     al,dx
           and    al,0f4h
           out    dx,al
           sub    dx,3
           xor    al,al
           out    dx,al
           in     al,inta01
           or     al,10h
           out    inta01,al
```

```
;           call    sal_dos             ;llama a la salida al DOS
            mov     al,0
            mov     nivel,al
            mov     dx,offset d_raiz
            mov     ah,3bh
            int     21h
            jnc     m_rcp1
            call    sal_dos
m_rcp1:     recpant
            print   42,8,blancos,norm
            print   42,8,d_raiz,norm
            print   42,9,blancos,norm
            recupera -
            ret
MOD_RCP     ENDP

;   Rutina para esperar los bloques de mensaje y encaminar a los mismos
;   a la deteccion de errores

RECEPCION   PROC    NEAR
            guarda

            mov     ax,0                ;pone en cero los contadores de
            mov     n_chart,ax          ;caracteres en Rx y Tx
            mov     n_charr,ax
            mov     ax,offset buff_dat  ;inicializa los buffers
            mov     buff_head,ax
            mov     buff_tail,ax
ret_5:      lmppant
            call    vent_recep
            STI                         ;habilita interrupciones

ret_3:      nop
ret_11:     mov     ax,buff_tail        ;lazo de espera por caracteres
            cmp     ax,buff_head        ;mientras el buffer esta vacio
            je      ret_11
            call    read_char           ;lee caracter del buffer
            push    ax
            push    dx
            mov     dl,norm             ;coloca atributo para la
            mov     atrib,dl            ;presentacion en pantalla
            call    disp_recep          ;del caracter recibido
            pop     dx
            pop     ax
            cmp     al,1ah              ;caracter de fin de bloque?
            je      ret_2
            jmp     ret_3               ;si no es 1ah regresa
ret_2:      call    det_error           ;si es 1ah va a detectar errores
            cmp     end_tx,1            ;bandera de fin de Tx es 1?
            je      ret_4               ;si es 1 salta a ret_4 (fin)
            mov     al,f_retx           ;colocar en el buffer de envio
            lea     si,buff_env         ;el caracter en F_RETX que puede
```

```
            mov     ds:[si],al          ;ser ACK o NACK
            mov     nb_env,1
            mov     lng_msg,1
            mov     ax,0
            mov     n_charr,ax
            mov     n_chart,ax
            mov     ax,offset buff_dat
            mov     buff_tail,ax
            mov     buff_head,ax

            call    tx_rx               ;transmision del ACK o NACK

            jmp     ret_5               ;va a esperar informacion
ret_4:      recupera
            ret
RECEPCION   ENDP

;   Rutina que calcula el sindrome del mensaje y CRC recibidos y
;   determina si existe o no errores en el bloque
;   Entrada:   Numero de caracteres recibidos (N_CHARR)
;              Bytes recibidos  (BUFF_DAT)
;   Salida:    F_RETX  (ACK o NACK)

DET_ERROR   PROC    NEAR
            guarda
            mov     ax,dseg
            mov     es,ax
            mov     ax,n_charr          ;AX=numero de caracteres recibidos
            cmp     ax,2                ;son dos caracteres?
            jne     det_e10
            lea     si,buff_dat         ;si son dos caracteres verifica si
            mov     al,ds:[si]          ;es caracter EOT
            cmp     al,04h
            je      det_e13             ;si es EOT va a fin de recepcion
det_e10:    mov     ax,n_charr          ;si N_CHARR > 2
            sub     ax,1                ;restar un caracter por 1ah
            mov     lng_rec,ax          ;LNG_REC=AX
            mov     cx,ax
            push    cx
            cld
            lea     si,buff_dat         ;mover la cadena apuntada por SI
            lea     di,buff_rec         ;al buffer apuntado por DI
            rep     movsb
            mov     bx,offset buff_dat  ;separa los dos bytes del CRC
            pop     cx                  ;y los almacena en CRC_REC
            sub     cx,2
            add     bx,cx
            mov     al,ds:[bx]
            inc     bx
            mov     ah,ds:[bx]
            mov     crc_rec,ax
```

```
            lea   si,crc_rec          ;convierte CRC_REC a ascii
            mov   bx,offset crc_reca
            call  ascii

            call  fly_rec             ;llama a calculo del sindrome

            lea   si,crc_cr           ;convierte sindrome a ascii
            mov   bx,offset crc_cra
            call  ascii

            mov   es,videoseg
            call  pant_det

            mov   ax,crc_cr           ;el sindrome es 0?
            cmp   ax,0
            je    det_e1
            mov   al,15h              ;si es diferente de 0
            mov   f_retx,al           ;F_RETX=NACK
            print 50,20,mens41,revers
            jmp   det_e2
det_e13:    jmp   det_e15
det_e1:     mov   al,06h              ;si es igual a 0
            mov   f_retx,al           ;F_RETX=ACK
            print 10,20,mens40,bold
            print 50,24,mens43,bold
            poscur 78,24
det_e3:     esp_tec                   ;espera tecla para continuar
            mov   ax,paltec           ;con la recepcion del bloque
            cmp   al,41h              ;archivar el mismo
            je    det_e5
            cmp   al,61h
            je    det_e5
            jne   det_e3
det_e15:    jmp   det_e11
det_e5:     call  recep_arch
            jmp   det_e12
det_e2:     print 52,24,mens45,bold   ;espera por una tecla para
            poscur 75,24              ;continuar con la retransmision
det_e7:     esp_tec                   ;retornar con el NACK
            mov   ax,paltec
            cmp   al,52h
            je    det_e14
            cmp   al,72h
            je    det_e14
            jmp   det_e7
det_e11:    mov   es,videoseg
            borrpant
            ventana 7,10,17,69,norm,10 ;presenta el nombre del archivo
            print  15,11,mens44,bold   ;en el cual se ha grabado el
            jmp    det_e17             ;mensaje recibido
det_e14:    jmp    det_e12
det_e17:    print  47,11,arch_rcp,norm
```

```
          print  47,24,mens47,bold
          poscur 78,24
det_e4:   esp_tec                    ;espera tecla para continuar
          mov    ax,paltec           ;con el listado del programa
          cmp    al,4ch
          je     det_e6
          cmp    al,6ch
          je     det_e6
          jne    det_e4
det_e6:   cerr   hand_rcp,er_lec     ;cierra el archivo de recepcion
          cmp    er_lec,0
          jne    det_e12
          call   list_arch           ;llama a listar el mensaje
          mov    end_tx,1
det_e8:   print  55,24,mens29,bold   ;espera la tecla para finalizar
          poscur 75,24               ;la recepcion
          esp_tec
          mov    ax,paltec
          cmp    al,01bh
          jne    det_e8
det_e12:  recupera
          ret
DET_ERROR ENDP

;  Rutina para calcular el sindrome del bloque de mensaje y el CRC
;  recibido. Es la misma rutina que se presenta en el calculo del
;  CRC con el metodo "on the fly"
;  Entrada:  Buffer con los datos recibidos (BUFF_REC)
;            Longitud de caracteres recibidos (LNG_REC
;            CRC inicial (CRC_INIC)
;  Salida:   Sindrome (CRC_CR)

FLY_REC   PROC   NEAR
          guarda
          lea    si,buff_rec
          mov    cx,lng_rec
          mov    bx,crc_inic
          mov    crc_cr,bx
fly_r1:   sub    ax,ax
          sub    dx,dx
          lodsb
          push   cx
          xor    al,bl
          mov    bx,crc_it
          xor    al,bl
          mov    dl,al
          push   dx
          push   ax
          mov    cl,8
          shl    ax,cl
          mov    ci,3
          shl    dx,cl
```

```
            xor     dx,ax
            pop     ax
            and     al,0f0h
            mov     cl,4
            shr     al,cl
            xor     dx,ax
            mov     bx,dx
            pop     dx
            and     dx,0fh
            xor     bx,dx
            mov     cl,7
            shl     dx,cl
            xor     bx,dx
            mov     cl,5
            shl     dx,cl
            xor     bx,dx
            mov     ax,bx
            mov     bx,crc_cr
            mov     cl,8
            shr     bx,cl
            xor     bx,ax
            mov     crc_cr,bx
            pop     cx
            loop    fly_r1
            recupera
            ret
FLY_REC     ENDP

;   Rutina para presentar la pantalla con el resultado de la
;   deteccion de errores

PANT_DET    PROC  NEAR
            guarda
            borrpant
            ventana 5,19,7,59,norm,3
            print  30,6,mens37,bold
            ventana 10,4,16,75,norm,7
            lin_h  196,4,10,72,norm
            lin_h  196,4,16,72,norm
            lin_v  179,3,11,5,norm
            lin_v  179,76,11,5,norm
            print   3,10,esq_si,norm
            print   3,16,esq_ii,norm
            print   76,10,esq_sd,norm
            print   76,16,esq_id,norm
            print  10,12,mens38,bold
            print  10,14,mens39,bold
            print  24,12,crc_reca,norm
            print  41,14,crc_cra,norm
            recupera
            ret
PANT_DET    ENDP
```

```
;   Rutina para grabar el bloque de mensaje correcto en un archivo
;   con el nombre que se introduzca en el programa y cuya extension
;   es .rcp

RECEP_ARCH PROC  NEAR
           guarda
           mov   al,num_rcp          ;si no es la primera vez que se
           cmp   al,0                ;llega a esta rutina ir a arch_r8
           jne   arch_r8

           abrir arch_rcp,2,hand_rcp,er_abr ;abrir el archivo anterior
           mov   ax,er_abr                  ;verificar posible error
           cmp   ax,0
           jne   arch_r2

arch_r8:   mov   ax,lng_rec
           sub   ax,2
           mov   nb_agr,ax

;   grabar el bloque recibido sin el CRC en el archivo de recepcion

           grabar hand_rcp,nb_agr,buff_rec,nb_gr,er_gr
           mov   ax,er_gr            ;verificar posible error
           cmp   ax,0
           jne   arch_r3
           jmp   arch_r6
arch_r2:   print 22,24,mense7,bold   ;mensajes de error
           jmp   arch_r4
arch_r3:   print 22,24,mense8,bold

arch_r4:   print 60,24,mens35,bold   ;si hay error espera tecla
           poscur 78,24              ;para salir al DOS
arch_r5:
           esp_tec
           mov   ax,paltec
           cmp   al,01bh
           jne   arch_r5
           call  sal_dos
arch_r6:   inc   num_rcp             ;si no hay error incrementa NUM_RCP
           recupera
           ret
RECEP_ARCH ENDP

;   Rutina para presentar en pantalla el mensaje total recibido

LIST_ARCH  PROC  NEAR
           guarda
           limppant
           print 30,1,mens48,bold
           poscur 0,3
           abrir arch_rcp,0,handle,er_lec  ;abrir archivo de recepcion
```

```
            cmp     er_lec,0
            jne     list_22

;leer archivo de recepcion

list_3:     leer    handle,nb_alr,buff_rcp,nb_lec,er_lec
            cmp     er_lec,0
            jne     list_22
            mov     ax,nb_lec           ;AX=numero de bytes leidos
            cmp     ax,0
            je      list_21
            cmp     ax,1000             ;compara AX con 1000
            jle     list_6
            xor     dx,dx               ;si es mayor a 1000
            mov     bx,1000             ;separa en bloques
            div     bx
            mov     lst_blq,ax
            cmp     dx,0
            je      list_10
            inc     lst_blq
            mov     lst_resto,dx
            jmp     list_13
list_10:    mov     lst_resto,0
            jmp     list_13
list_22:    jmp     list_19
list_13:    mov     dx,1000
            jmp     list_9
list_6:     mov     dx,ax
            mov     lst_blq,1
list_9:     lea     si,buff_rcp         ;SI=deplazamiento de BUFF_RCP
            mov     cx,lst_blq          ;CX=numero de bloques
            jmp     list_12
list_21:    jmp     list_4
list_12:    poscur 0,3
list_11:    lodsb                       ;escribe la cadena de bytes
            mov     bh,0                ;de longitud dada en DX desde
            mov     bl,0                ;la posicion senalada para el
            mov     ah,0eh              ;cursor
            int     10h
            dec     dx
            cmp     dx,0
            jne     list_11
            cmp     lst_blq,1           ;comparaciones para determinar
            je      list_8              ;la longitud del siguiente
            cmp     lst_resto,0         ;bloque a presentar
            je      list_17
            cmp     cx,2
            jne     list_17
            mov     dx,lst_resto
            jmp     list_18
list_23:    jmp     list_12
list_17:    mov     dx,1000
```

```
list_18:  cmp    cx,1
          je     list_8
          jmp    list_20
list_19:  jmp    list_1
list_20:  print  62,23,mens52,bold    ;espera tecla para presentar
          poscur 78,23                ;la siguiente pantalla
list_16:  esp_tec
          mov    ax,paltec
          cmp    al,0dh
          jne    list_16
          lmppant
          loop   list_23
list_8:   jmp    list_3
list_1:   call   esc_lec
          jmp    list_5
list_4:   cerr   handle,er_lec        ;cerrar archivo de recepcion
          cmp    er_lec,0
          jne    list_1
list_5:   recupera
          ret
LIST_ARCH  ENDP


;  ***** RUTINAS PARA MANEJO DE ARCHIVOS *****
;  Rutina para colocar en un buffer el nombre del ultimo archivo
;  de mensaje en forma de cadena asciiz
;  Entrada:  SI=desplazamiento del buffer para el nombre

NOM_ARCH  PROC   NEAR
          guarda
          mov    bx,offset asczd      ;BX=puntero del directorio
n_arch1:  mov    al,ds:[bx]
          cmp    al,0
          je     n_arch2
          mov    ds:[si],al           ;graba directorio en [SI]
          inc    si
          inc    bx
          jmp    n_arch1
n_arch2:  cmp    f_esc1,1              ;si se presiono ESC hay solo
          je     n_arch5              ;unidad
          mov    byte ptr ds:[si],'\'
          inc    si
n_arch5:  mov    bx,offset asczar     ;BX=puntero del archivo
n_arch3:  mov    al,ds:[bx]
          cmp    al,0
          je     n_arch4
          mov    ds:[si],al           ;graba archivo en [SI]
          inc    si
          inc    bx
          jmp    n_arch3
n_arch4:  mov    al,0                 ;aumenta byte de ceros
          mov    byte ptr ds:[si],al
```

```
          recupera
          ret
NOM_ARCH  ENDP

;  Rutina para leer un archivo de mensaje cuyo nombre esta en el
;  buffer ARCH_MSG

LEC_MSG   PROC  NEAR
          guarda
          abrir  arch_msg,0,handle,er_lec  ;abrir archivo arch_msg
          cmp    er_lec,0            ;verifica posibles errores
          je     lecmsg1
          call   esc_lec

;  leer el archivo

lecmsg1:  leer   handle,nb_alr,buff_msg,nb_lec,er_lec
          cmp    er_lec,0            ;verifica posibles errores
          je     lecmsg2
          call   esc_lec

lecmsg2:  cerr   handle,er_lec       ;leer archivo
          cmp    er_lec,0            ;verifica posibles errores
          je     lecmsg3
          call   esc_lec

lecmsg3:  recupera
          ret
LEC_MSG   ENDP

;  Rutina para leer el archivo de mensaje cuyo nombre esta en el
;  buffer ARCH_ORG  (identico a LEC_MSG)

LEC_ORG   PROC  NEAR
          guarda
          abrir  arch_org,0,handle,er_lec
          cmp    er_lec,0
          je     lecorg1
          call   esc_lec

lecorg1:  leer   handle,nb_alr,buff_org,nb_lec,er_lec
          cmp    er_lec,0
          je     lecorg2
          call   esc_lec

lecorg2:  cerr   handle,er_lec
          cmp    er_lec,0
          je     lecorg3
          call   esc_lec

lecorg3:  recupera
          ret
```

```
LEC_BRG    ENDP

;  Rutina para presentar en pantalla un mensaje cuando hay error en
;  la lectura del archivo

ESC_LEC    PROC   NEAR
           mov    ax,seg mensé
           mov    ds,ax
           mov    dx,offset mensé
           mov    ah,09h
           int    21h
           ret
ESC_LEC    ENDP


;  ¡¡¡¡¡  RUTINAS LLAMADAS POR LOS MACROS   ¡¡¡¡¡


SUB        PROC   NEAR


ESP:       guarda
           mov    ah,0              ;funcion esperar tecla
           int    16h
           mov    paltec,ax         ;PALTEC=tecla ingresada
           recupera
           ret
;

ILET:
           mov    al,28             ;columna y fila de la primera
           mov    x,al              ;opcion
           mov    al,12
           mov    y,al
           cmp    cl,0              ;compara con 0 (primera opcion)
           je     ilet1
ilet2:     inc    y                 ;incrementa fila
           loop   ilet2             ;lazo conforme cl
ilet1:     posdir x,y               ;obtiene direccion de (X,Y)
           mov    si,pddir          ;SI=direccion de (X,Y)
           mov    di,si             ;obtiene caracter escrito en esa
ilet5:     mov    ax,es:[si]        ;direccion
           inc    si
           inc    si
           cmp    al,0              ;compare con fin de linea
           je     ilet3             ;si es igual va a fin
           cmp    al,bl             ;busca el caracter pedido en
           jne    ilet4             ;LETRA y cambia su atributo
           mov    ah,dl             ;volviendolo a escribir en la
ilet4:     mov    es:[di],ax        ;misma posicion
           inc    di
           inc    di
```

```
          jmp   ilet5
ilet3:    ret
;

MACSP:
          xor   si,si               ;SI=0
          mov   di,offset buff_pant ;DI=puntero de BUFF_PANT
          mov   bx,ds               ;mueve la cadena de caracteres
          mov   es,bx               ;de pantalla (caracter y atributo)
          mov   ds,videoseg         ;al buffer BUFF_PANT
          mov   cx,4000
          rep   movsb
          ret
;

MACRP:
          xor   di,di               ;DI=0
          mov   es,videoseg         ;mueve la cadena de caracteres del
          mov   si,offset buff_pant ;BUFF_PANT (caracter y atributo) al
          mov   cx,4000             ;segmento de video
          rep   movsb
          ret
;

MACSTR:
          mov   dl,cl
macstr2:  mov   ah,01h              ;funcion presenta tecla en pantalla
          int   21h                 ;pantalla
          cmp   al,01bh             ;es ESC?
          je    macstr5
          cmp   al,5ch
          jne   macstr3
macstr5:  mov   f_esc,1             ;si es ESC, F_ESC=1
          mov   longin,0            ;LONGIN=0
          jmp   macstr1
macstr3:  cmp   al,0dh              ;si es ENTER ir a fin
          je    macstr9
          cmp   al,08h              ;si es tecla de retroceso
          je    macstr4             ;saltar a macstr4
          lea   si,tab_arch
macstr6:  mov   dl,ds:[si]
          inc   si
          cmp   dl,0ffh
          je    macstr8
          cmp   al,dl
          jne   macstr6
          mov   ds:[bx],al          ;grabar en TEC_STR el caracter
          inc   longin              ;incrementar contador
          inc   bx                  ;incrementar puntero
          dec   cl
          cmp   cl,0                ;si no es 0 saite a esperar
          jne   macstr2             ;tecla
```

```
            cmp     al,0dh
            jne     macstr7
            dec     longin
            jmp     macstr1
macstr9:    cmp     dl,cl
            je      macstr5
            jmp     macstr1
macstr4:    dec     bx                  ;retroceder puntero
            inc     cl
            dec     longin              ;decrementar contador
            push    bx
            mov     al,20h              ;escribir el caracter espacio
            mov     bh,0                ;en la posicion retrocedida
            mov     ah,0ah
            int     10h
            pop     bx
            jmp     macstr2             ;vuelve a esperar tecla
macstr8:    dec     cl
            inc     bx
            inc     longin
macstr7:
            push    bx
            push    cx
            push    dx
            mov     bh,0
            mov     ah,03h
            int     10h
            mov     y_fil,dh
            mov     x_col,dl
            dec     x_col
            poscur  x_col,y_fil
            pop     dx
            pop     cx
            pop     bx
            jmp     macstr4
macstr1:    ret
;

MAC_SA:
            mov     dl,cl
macsa4:     mov     ah,01h
            int     21h
            dec     cl
            cmp     cl,0
            je      macsa1
            cmp     al,0dh
            je      macsa9
            lea     si,t_unid
macsa3:     mov     dl,ds:[si]
            inc     si
            cmp     dl,0ffh
            je      macsa2
```

```
            cmp     al,dl
            jne     macsa3
            cmp     al,'B'
            je      macsa8
            cmp     al,'b'
            jne     macsa10
macsa8:     mov     f_b,1
macsa10:    mov     ds:[bx],al
            jmp     macsa4
macsa9:     mov     f_esc,1
            jmp     macsa5
macsa1:     cmp     al,08h
            je      macsa6
            cmp     al,0dh
            je      macsa5
macsa2:     inc     cl
            push    bx
            push    cx
            push    dx
            mov     bh,0
            mov     ah,03h
            int     10h
            mov     y_fil,dh
            mov     x_col,dl
            dec     x_col
            poscur  x_col,y_fil
            pop     dx
            pop     cx
            pop     bx
            jmp     macsa7
macsa6:     inc     cl
            inc     cl
macsa7:     push    bx
            mov     bh,0
            mov     al,20h
            mov     ah,0ah
            int     10h
            pop     bx
            jmp     macsa4
macsa5:     ret
;

MLINH:
mlinh1:     print   x,y,lin,atrib       ;escribe caracter en (X,Y)
            inc     x                   ;incrementa columna
            loop    mlinh1              ;lazo hasta que CX=0
            ret
;

MLINV:
mlinv1:     print   x,y,lin,atrib       ;escribe caracter en (X,Y)
            inc     y                   ;incrementa fila
```

```
            loop    mlinv1              ;lazo hasta que CX=0
            ret
;
RHA:
            push    ax                  ;salva registros
            push    cx
            push    bx
            mov     bx,offset tabha     ;BX=puntero de TABHA
            mov     dl,al
            and     al,0f0h             ;nibble mas significativo
            mov     ah,0
            mov     cl,16               ;se pasa a la parte menos signif.
            div     cl
            xlat    tabha               ;traduccion a ascii
            mov     ch,al
            mov     al,dl
            and     al,0fh              ;nibble menos significativo
            xlat    tabha               ;traduccion a ascii
            mov     cl,al
            mov     dx,cx
            pop     bx
            pop     cx
            pop     ax
            ret
;
RPD:        push    ax
            push    cx
            mov     al,160              ;direccion de video es:
            mul     ch                  ;160*Y + 2*X
            add     cl,cl
            mov     ch,0
            add     ax,cx
            mov     pddir,ax            ;PDDIR=direccion de video
            pop     cx
            pop     ax
            ret
;
RPRT:       push    di
            mov     di,pddir            ;DI=direccion de video
rprt1:      lodsb                       ;escribe en pantalla hasta
            cmp     al,0                ;que haya un cero
            je      rprt2
            stosw
            jmp     rprt1
rprt2:      pop     di
            ret

SUB         ENDP
```

```
;MAC_PANT.ASM

;   ‡‡  MACROS PARA ESCRITURA DE TEXTOS EN PANTALLA  ‡‡

MAC_PANT   MACRO

;   ‡   Macro para borrar la pantalla con un caracter

BORRPANT   MACRO
           push   cx                ;salva registros
           push   di
           push   ax
           mov    cx,2000           ;numero de caracteres
           mov    di,0              ;DI= area de video
           mov    ax,07b0h          ;AX= caracter y atributo
           rep    stosw
           pop    ax                ;recupera registros
           pop    di
           pop    cx
           ENDM

;   ‡   Macro para borrar un numero de caracteres a partir de cierta
;       posicion
;       Entrada:  X= columna
;                 Y= fila
;                 NUM-CAR= numero de caracteres

BORRVENT   MACRO   x,y,num_car
           push    ax              ;salva registros
           push    cx
           push    di
           posdir  x,y             ;posiciona el cursor en (X,Y)
           mov     cx,num_car      ;numero de caracteres a borrar
           mov     di,pddir        ;DI= direccion del cursor
           mov     ax,07b0h        ;AX= caracter y atributo
           rep     stosw           ;escritura en pantalla
           pop     di
           pop     cx              ;recupera registros
           pop     ax
           ENDM

;   ‡   Macro para esperar el ingreso de una cadena de caracteres
;       Entrada:   LONG= maximo numero de caracteres esperados
;       SALIDA:    TEC_STR= cadena de caracteres ingresados
;                  LONGIN= numero de carateres ingresados

ESP_STR    MACRO  tec_str,long
           guarda
           mov    bx,offset tec_str    ;BX=puntero de TEC_STR
           mov    cl,long              ;CL=LONG
           mov    longin,0             ;inicio del contador
           call   macstr               ;llama a macstr
```

```
          recupera
          ENDM

;   *   Macro para esperar el ingreso de una cadena de caracteres

ESP_SA    MACRO  tec_sa,long_sa
          guarda
          mov    bx,offset tec_sa     ;BX=puntero de TEC_SA
          mov    cl,long_sa           ;CL=LONG_SA
          mov    longin,0             ;inicio del contador
          call   mac_sa               ;llama a macstr
          recupera
          ENDM

;   *   Macro para esperar el ingreso de una tecla

ESP_TEC   MACRO
          call   esp                  ;llama a esp
          mov    ax,paltec            ;AX=PALTEC tecla presionada
          ENDM

;   *   Macro para guardar pantalla actual en memoria

GUARPANT  MACRO
          guarda
          call   macgp                ;llama a macgp
          recupera
          ENDM

;   *   Macro para convertir un numero hexadecimal en ascii

HEASC     MACRO
          call   rha                  ;llama a rha
          ENDM

;   *   Macro para resaltar la letra de acceso a una opcion
;       Entrada:   N_OPC=numero de opcion
;                  LETRA=caracter ascii en decimal
;                  ATR=atributo

ILUMLET   MACRO  n_opc,letra,atr
          guarda
          mov    dl,atr               ;recoge parametros de inicio
          mov    cl,n_opc
          mov    bl,letra
          call   ilet                 ;llama a ilet
          recupera
          ENDM
```

```
LIN_H     MACRO  caracter,col,fil,num_car,in_atrib
          push   ax                  ;salva registros
          push   bx
          push   cx
          push   dx
          push   si
          lea    si,lin              ;SI=desplazamiento de LIN
          mov    al,caracter         ;coloca caracter a dibujar en
          mov    ds:[si],al          ;memoria con un byte final de ceros
          inc    si
          mov    al,0
          mov    ds:[si],al
          mov    x,col               ;recoge parametros de inicio
          mov    y,fil
          mov    cx,num_car
          mov    dl,in_atrib
          mov    atrib,dl            ;ATRIB=atributo
          call   mlinh               ;llama a mlinh
          pop    si                  ;recupera registros
          pop    dx
          pop    cx
          pop    bx
          pop    ax
          ENDM

;    Macro para dibujar una linea vertical
;    Entrada:   CARACTER=caracter ascii a dibujar
;               COL=columna inicial
;               FIL=fila inicial
;               NUM_CAR=numero de caracteres (largo de la linea)
;               IN_ATRIB=atributo del caracter

LIN_V     MACRO  caracter,col,fil,num_car,in_atrib
          push   ax                  ;salva registros
          push   bx
          push   cx
          push   dx
          push   si
          lea    si,lin              ;SI=desplazamiento de LIN
          mov    al,caracter         ;coloca caracter a dibujar en
          mov    ds:[si],al          ;memoria con un byte final de
          inc    si                  ;ceros
          mov    al,0
          mov    ds:[si],al
          mov    x,col               ;recoge parametros iniciales
          mov    y,fil
          mov    cx,num_car
          mov    dl,in_atrib
          mov    atrib,dl            ;ATRIB=atributo
          call   mlinv               ;llama a mlinv
          pop    si                  ;recupera registros
          pop    dx
```

```
          pop   cx
          pop   bx
          pop   ax
          ENDM

;   ł   Macro para limpiar la pantalla con atributo normal

LMPPANT   MACRO
          push  cx                ;salva registros
          push  di
          push  ax
          mov   cx,2000           ;contador
          mov   di,0              ;DI=0 inicio
          mov   ax,0720h          ;AX=caracter y atributo
          rep   stosw             ;escribir en la pantalla
          pop   ax                ;recupera registros
          pop   di
          pop   cx
          ENDM

;   ł   Macro para posicionar el cursor en (X,Y)
;       Entrada:   X=columna
;                  Y=fila

POSCUR    MACRO  x,y
          push  ax                ;salva registros
          push  bx
          push  dx
          mov   dh,y              ;DH=fila
          mov   dl,x              ;DL=columna
          mov   bh,0
          mov   ah,2              ;funcion posicionar cursor
          int   10h
          pop   dx
          pop   bx
          pop   ax
          ENDM

;   ł   Macro para obtener la direccion PDDIR correspondiente a (X,Y)
;       para acceso directo a memoria de video
;       Entrada:  X=columna
;                 Y=fila

POSDIR    MACRO  x,y
          push  cx
          mov   cl,x              ;recoge parametros iniciales
          mov   ch,y
          call  rpd               ;llama a rpd
          pop   cx
          ENDM

;   ł   Macro para escribir en pantalla un texto en memoria
```

```
;       que termina con 0
;       Entrada:  X=columna
;                 Y=fila
;                .DIR=buffer de texto
;                 ATRIB=atributo

PRINT       MACRO  x,y,dir,atrib
            posdir x,y                  ;direccion de video
            push   si
            push   ax
            mov    si,offset dir        ;SI=puntero de texto
            mov    ah,atrib
            call   rprt                 ;llama a rprt
            pop    ax
            pop    si
            ENDM

;   ‡   Macro para recuperar de memoria una pantalla

RECPANT     MACRO
            guarda
            call   macrp                ;llama a macrp
            recupera
            ENDM

;   ‡   Macro para crear una ventana en pantalla
;       Entrada:   FSI=fila superior izquierda
;                  CSI=columna superior izquierda
;                  FID=fila inferior derecha
;                  CID=columna inferior derecha
;                  ATR=atributo
;                  N_linea=numero de lineas

VENTANA     MACRO  fsi,csi,fid,cid,atr,n_linea
            push   ax
            push   bx
            push   cx
            push   dx
            mov    ch,fsi               ;recoge caracteres de
            mov    cl,csi               ;inicio
            mov    dh,fid
            mov    dl,cid
            mov    bh,atr
            mov    al,n_linea
            mov    ah,07h               ;desplazamiento hacia abajo
            int    10h                  ;de la pagina activa
            pop    dx
            pop    cx
            pop    bx
            pop    ax
            ENDM
            ENDM
```

```
;MAC_ARCH.ASM

;   ‡‡  MACROS PARA MANEJO DE ARCHIVOS  ‡‡

MAC_ARCH   MACRO

;   ‡   Macro para crear un archivo
;       Entrada:   FICHERO= cadena asciiz con el nombre del archivo
;                  ATRIB= atributo del archivo. (1 palabra)
;                  0 - lectura/escritura
;       Salida:    HANDLE= identificador del archivo  (1 palabra)
;                  ERROR= numero del error (1 palabra)

CREAR      MACRO fichero,atrib,handle,error
           LOCAL bien,fin

           push   ax                  ;salva registros
           push   cx
           push   dx

           lea    dx,fichero          ;recoge parametros de entrada
           mov    cx,atrib

           mov    ah,3ch              ;crear el archivo
           int    21h

           jnc    bien                ;si cf=0 bifurcar

           mov    error,ax            ;codigo del error
           jmp    fin

bien:      mov    handle,ax           ;file handle
           mov    error,0             ;error=0

fin:       pop    dx                  ;restaura registros afectados
           pop    cx
           pop    ax

           ENDM

;   ‡   Macro para abrir un archivo ya existente
;       Entrada:   FICHERO= cadena asciiz con el nombre del archivo
;                  ACCESO= codigo del acceso (1 palabra)
;                  0 - solo lectura     1 - solo escritura
;                  2 - lectura y escritura
;       Salida:    HANDLE= identificador del archivo (1 palabra)
;                  ERROR= numero del error (1 palabra)

ABRIR      MACRO fichero,acceso,handle,error
           LOCAL bien,fin

           push   ax                  ;salva registros afectados
```

```
            push    dx

            lea     dx,fichero          ;recoge parametros de entrada
            mov     al,acceso

            mov     ah,3dh              ;funcion abrir archivo
            int     21h

            jnc     bien                ;si cf=0 bifurcar

            mov     error,ax            ;codigo del error
            jmp     fin

bien:       mov     handle,ax           ;file handle
            mov     error,0             ;error=0

fin:        pop     dx                  ;restaura registros afectados
            pop     ax

            ENDM

;   *   Macro para leer un cierto numero de bytes de un archivo
;       Entrada:    HANDLE= identificador del fichero (1 palabra)
;                   NBYTES1= numero de bytes a leer (1 palabra)
;       Salida:     AREA= area de lectura (variable)
;                   NBYTES2= numero de bytes leidos (1 palabra variable)
;                   ERROR= numero del error (1 palabra)

LEER        MACRO handle,nbytes1,area,nbytes2,error
            LOCAL bien,fin

            push    ax                  ; salva registros
            push    bx
            push    cx
            push    dx

            mov     bx,handle           ;recoge parametros de entrada
            mov     cx,nbytes1
            lea     dx,area

            mov     ah,03fh             ;funcion leer registro
            int     21h

            jnc     bien                ;si cf=0 bifurca
            mov     error,ax            ;codigo del error
            jmp     fin

bien:       mov     nbytes2,ax          ;numero de bytes leidos
            mov     error,0             ;error = 0

fin:        pop     dx                  ;restaura registros afectados
            pop     cx
```

```
        pop     bx
        pop     ax

        ENDM

;   *   Macro para grabar un registro en un archivo
;       Entrada:    HANDLE= identificador del fichero (1 palabra)
;                   NBYTES1= numero de bytes a grabar (1 palabra)
;                   AREA= area a grabar (variable)
;       Salida:     NBYTES2= numero de bytes grabados (1 palabra variable)
;                   ERROR= numero del error (1 palabra)

GRABAR  MACRO handle,nbytes1,area,nbytes2,error
        LOCAL bien,fin

        push    ax                  ;salva registros
        push    bx
        push    cx
        push    dx

        mov     bx,handle           ;recoge parametros de entrada
        mov     cx,nbytes1
        lea     dx,area

        mov     ah,40h              ;funcion grabar registro
        int     21h

        jnc     bien                ;si cf=0 bifurcar
        mov     error,ax            ;codigo del error
        jmp     fin

bien:   mov     nbytes2,ax          ;numero de bytes grabados
        mov     error,0             ;error=0

fin:    pop     dx                  ;restaura registros
        pop     cx
        pop     bx
        pop     ax
        ENDM

;   *   Macro para cerrar un archivo
;       Entrada:    HANDLE= identificador del fichero (1 palabra)
;       Salida:     ERROR= numero del error (1 palabra)

CERR    MACRO handle,error
        LOCAL bien,fin

        push    ax                  ;salva registros
        push    bx

        mov     bx,handle           ;recoge parametro de entrada
```

```
;BIT.ASM

;   ;; OBTENCION DEL CRC CON EL ALGORITMO DE CALCULO BIT A BIT ;;

    SSEG SEGMENT STACK
    DB 256 DUP (?)
    SSEG ENDS

;   Constantes a utilizar en el programa
SHFT        EQU         8
CRC_INIC    EQU         0000H

    DSEG SEGMENT
;   Datos para el calculo del CRC bit a bit
CRC_M1      DW          0               ;palabra para almacenar CRC
BUFF_MSG    DB          'ALGORITMO DE CALCULO DEL CRC' ;buffer de mensaje
BINARIO     DB           8 DUP (?)      ;memoria para conversion a binario
PERMUT      DB          28 DUP (?)      ;bytes de mensaje permutados
LONG        DW          28

    DSEG ENDS

    CSEG SEGMENT
    ASSUME CS:CSEG,DS:DSEG,SS:SSEG,ES:NOTHING

    START PROC FAR
;ETIQUETA  OPC    OPER                 COMENTARIOS
          mov    ax,dseg              ;inicializa segmento de datos
          mov    ds,ax
          mov    es,ax
          call   permutar
          mov    ax,crc_inic          ;AX=CRC inicial
          mov    crc_m1,ax            ;CRC_M1=AX
          mov    dx,long              ;DX=LONG
          lea    si,permut            ;SI=desplazamiento de PERMUT
salto2:   lodsb
          mov    bl,al                ;BL=byte de mensaje permutado
          mov    cx,shft              ;CX=numero de desp. ciclicos
          push   dx                   ;salva registro DX
          mov    dl,80h               ;DL=10000000
salto1:   test   bl,dl                ;ver si MSB es 1 o 0
          push   bx                   ;salva registro BX
          jnz    uno
          mov    bx,0                 ;si es cero BX=0
          jmp    salto3
uno:      mov    bx,1                 ;si es uno BX=1
salto3:   mov    ax,crc_m1            ;AX=CRC_M1
          push   ax                   ;salva registro AX
          and    ax,0001h             ;separar el LSB del CRC
          xor    ax,bx                ;AX=AX xor BX
          test   ax,1                 ;bit en AX es 1?
          jnz    nocero
```

```
            mov     bx,0000h            ;si es cero BX=0000
            jmp     desp
nocero:     mov     bx,8408h            ;si es uno BX=8408h
desp:       pop     ax                  ;recupera registro AX (CRC)
            shr     ax,1                ;AX=AX===>1 lugar
            xor     ax,bx               ;AX=AX xor BX
            mov     crc_w1,ax           ;CRC_W1=AX (CRC temporal)
            shr     dl,1                ;DL=DL===>1 lugar
            pop     bx                  ;recupera registro DX
            loop    salto1              ;lazo hasta que CX=0
            pop     dx                  ;recupera registro DX
            dec     dx                  ;DX=DX-1
            loopnz  salto2              ;lazo hasta que DX=0
            mov     ah,4ch
            int     21h
            RET
START ENDP
;
;   Rutina para permutar los bytes del mensaje, es decir cambiar el
;   orden de los bits, de manera que el bit mas significativo quede
;   como el menos significativo
;   Entrada:    Buffer con el mensaje (BUFF_MSG)
;               Buffer para la conversion a binario (BINARIO)
;               Longitud del bloque (LONG)
;   Salida:     Buffer con los bytes de mensaje permutados (PERMUT)

PERMUTAR    PROC    NEAR
            mov     cx,long             ;CX=LONG
            cld
            lea     si,buff_msg         ;SI=desplazamiento de BUFF_MSG
            lea     di,binario          ;DI=desplazamiento de BINARIO
            lea     bx,permut           ;BX=apunta a PERMUT
perm1:      push    cx                  ;salva registros
            push    si
            push    bx
            add     di,7                ;DI=DI+7
            mov     ch,0
            mov     cl,8                ;CX=contador de bits
            lodsb                       ;AL=byte del mensaje
            mov     bl,80h              ;BL=10000000
perm2:      test    al,bl               ;el bit en AL es 1?
            jnz     perm3
            mov     dl,0                ;si es cero DL=0
            jmp     perm4
perm3:      mov     dl,1                ;si es uno DL=1
perm4:      mov     [di],dl             ;[DI]<=== DL
            dec     di                  ;decrementa DI
            shr     bl,1                ;BL=BL ===> 1 lugar
            loop    perm2               ;lazo hasta CX=0
            inc     di
            pop     bx                  ;recupera registro BX
            call    bin_hex
```

```
          pop     si                  ;recupera registro SI
          inc     si
          pop     cx                  ;recupera registro CX
          loop    permi               ;lazo hasta que CX=0
          ret
PERMUTAR    ENDP
;
;   Rutina para convertir un numero en binario en hexadecimal
;   Entrada:   Buffer de datos binarios (BINARIO)
;   Salida:    Graba en buffer de datos permutados el byte

BIN_HEX     PROC   NEAR

          push    bx                  ;salva registro BX
          mov     ch,0
          mov     cl,7                ;CX=contador de bits
          cld
          lea     si,binario          ;SI=desplazamiento de BINARIO
          lodsb                       ;AL=primer bit
b_hex1:   shl     al,1                ;AL=AL <=== 1 lugar
          mov     bl,al               ;BL=AL
          lodsb                       ;AL=siguiente bit
          xor     al,bl               ;AL=AL xor BL
          loop    b_hex1              ;lazo hasta que CX=0
          pop     bx                  ;recupera registro BX
          mov     [bx],al             ;[BX] <=== AL
          inc     bx
          ret
BIN_HEX     ENDP
CSEG ENDS
END START
```

```
;TABLA.ASM
;   ** CALCULO DEL CRC POR EL METODO DE BUSQUEDA DEL CRC DE UN BYTE **
;   ** EN UNA TABLA **
;
    SSEG SEGMENT STACK
    DB 256 DUP (?)
    SSEG ENDS

;   Constantes para el programa
LONG_T     EQU      256             ;longitud de la tabla
CRC_INIC   EQU      0000H           ;CRC inicial
CRC_IT     EQU      0000H
    DSEG SEGMENT
;   Datos para el calculo con busqueda en tabla
CRC_M2     DW       0               ;palabra de memoria para el CRC
BUFF_MSG   DB       'ALGORITMO DE CALCULO DEL CRC'  ;buffer de mensaje
TABLA_1    DW       256 DUP (?)     ;area para almacenar la tabla
LONG       DW       28              ;longitud del mensaje
    DSEG ENDS
;
    CSEG SEGMENT
    ASSUME CS:CSEG,DS:DSEG,SS:SSEG,ES:NOTHING
START PROC FAR
;ETIQUETA  OPC     OPER      COMENTARIOS
          mov    ax,dseg
          mov    ds,ax
          mov    es,ax
          call   crcxi
          mov    cx,long            ;CX=LONG
          mov    ax,crc_inic        ;AX=CRC inicial
          mov    crc_m2,ax          ;CRC_M2=AX
          lea    bx,buff_msg        ;BX=desplazamiento de BUFF_MSG
b_tabl:   push   cx                 ;guarda registro CX
          sub    ax,ax
          mov    al,[bx]            ;AL=byte del mensaje
          push   bx                 ;salva registro BX
          mov    bx,crc_m2          ;BX=CRC_M2
          xor    al,bl              ;AL=AL xor BL
          mov    cx,ax              ;CX=AX
          lea    si,tabla_1         ;SI=desplazamiento de TABLA_1
          add    si,cx              ;CX=indice de la tabla
          add    si,cx
          lodsw                     ;AX=CRC de la tabla
          mov    bx,crc_m2          ;BX=CRC_M2
          mov    ci,8
          shr    bx,cl              ;BX= BX===>8 lugares
          xor    ax,bx              ;AX=AX xor BX
          mov    crc_m2,ax          ;CRC_M2=AX
          pop    bx                 ;recupera registro BX
          inc    bx                 ;incrementa BX
          pop    cx                 ;recupera registro CX
          loop   b_tabl             ;lazo hasta que CX=0
```

```
          mov   ah,4cr
          int   21h
          RET
START ENDP
;
CRCXI     PROC  NEAR
          cld
          lea   di,tabla_1          ;DI=desplazamiento de TABLA_1
          mov   bl,00h              ;BL=00h valor inicial
          mov   cx,long_t           ;CX=LONG_T
crcxii:   mov   ax,crc_it           ;AX=CRC inicial
          push  cx                  ;salva CX
          push  bx                  ;salva BX
          sub   dx,dx
          xor   al,bl               ;AL=AL xor BL (AL=VECTOR X)
          mov   ah,0
          mov   bl,al
;   la siguiente es la combinacion de los componentes del vector
;   X que dan el CRC de un byte. indicada en la TABLA 3.5
          push  dx                  ;salvar DX (vector X)
          push  ax                  ;salvar AX (vector X)
          mov   cl,8
          shl   ax,cl               ;AX=AX <=== 8 lugares (AX=b0-b7)
          mov   cl,3
          shl   dx,cl               ;DX=DX <=== 3 lugares (DX=b5-b12)
          xor   dx,ax               ;DX=DX xor AX (DX=b0-b12)
          pop   ax                  ;recupera registro AX
          and   al,0f0h             ;AL=nibble mas significativo de X
          mov   ci,4
          shr   al,cl               ;AL=AL ===> 4 lugares (AL=b12-b15)
          xor   dx,ax               ;DX=DX xor AX (b0-b15)
          mov   bx,dx               ;BX=DX
          pop   dx                  ;recupera registro DX
          and   dx,0fh              ;DX=nibble menos significativo de X
          xor   bx,dx               ;BX=BX xor DX
          mov   cl,7
          shl   dx,cl               ;DX=DX <=== 7 lugares (DX=b3-b8)
          xor   bx,dx               ;BX=BX xor DX
          mov   cl,5
          shl   dx,cl               ;DX=DX <=== 5 lugares (DX=b0-b3)
          xor   bx,dx               ;BX=BX xor DX
;
          mov   ax,bx               ;AX=BX (CRC de un byte)
          stosw                     ;graba el CRC en TABLA_1
          pop   bx                  ;recupera registro BX
          inc   bx                  ;incrementa BX
          pop   cx                  ;recupera registro CX
          loop  crcxii              ;lazo hasta que CX=0
          ret
CRCXI     ENDP
   CSEG ENDS
END START
```

```
;FLY.ASM
;   ** CALCULO DEL CRC-CCITT POR EL METODO 'ON THE FLY' **
;
    SSEG SEGMENT STACK
    DB 256 DUP (?)
    SSEG ENDS

;   Constantes para el programa
CRC_INIC   EQU      0000H           ;CRC inicial
CRC_IT     EQU      0000H
;
    DSEG SEGMENT
;   Datos para el calculo del CRC con el metodo "on the fly"
CRC_M3     DW       0               ;palabra para grabar el CRC
BUFF_MSG   DB       'ALGORITMO DE CALCULO DEL CRC'  ;buffer de mensaje
LONG       DW       28              ;longitud del mensaje
    DSEG ENDS

    CSEG SEGMENT
    ASSUME CS:CSEG,DS:DSEG,SS:SSEG,ES:NOTHING

START PROC FAR
;ETIQUETA  OPC     OPER      COMENTARIOS
          mov    ax,dseg
          mov    ds,ax
          mov    es,ax

          lea    si,buff_msg         ;SI=desplazamiento de BUFF_MSG
          mov    cx,long             ;CX=LONG
          mov    bx,crc_inic         ;BX=CRC inicial
          mov    crc_m3,bx           ;CRC_M3=BX
b_fly1:   sub    ax,ax
          sub    dx,dx
          lodsb                      ;AL=byte del mensaje
          push   cx                  ;salva registro CX
          xor    al,bl               ;AL=AL xor BL
          mov    bx,crc_it           ;BX=CRC_IT
          xor    al,bl               ;AL=AL xor BL
          mov    dl,al               ;DL=AL (VECTOR X)

;   la siguiente es la combinacion de los componentes del vector
;   X que dan el CRC de un byte, indicada en la TABLA 3.5

          push   dx                  ;salvar DX (vector X)
          push   ax                  ;salvar AX (vector X)
          mov    cl,8
          shl    ax,cl               ;AX=AX (=== 8 lugares (AX=b0-b7)
          mov    cl,3
          shl    dx,cl               ;DX=DX (=== 3 lugares (DX=b5-b12)
          xor    ax,ax               ;DX=DX xor AX (DX=b0-b12)
          pop    ax                  ;recupera registro AX
          and    al,0f0h             ;AL=nibble mas significativo de )
```

```
            mov     cl,4
            shr     al,cl               ;AL=AL ===> 4 lugares (AL=b12-b15)
            xor     dx,ax               ;DX=DX xor AX (b0-b15)
            mov     bx,dx               ;BX=DX
            pop     dx                  ;recupera registro D)
            and     dx,0fh              ;DX=nibble menos significativo de X
            xor     bx,dx               ;BX=BX xor DX
            mov     cl,7
            shl     dx,cl               ;DX=DX <=== 7 lugares (DX=b5-b8)
            xor     bx,dx               ;BX=BX xor DX
            mov     cl,5
            shl     dx,cl               ;DX=DX <=== 5 lugares (DX=b0-b3)
            xor     bx,dx               ;BX=BX xor DX
            mov     ax,bx               ;AX=BX
            mov     bx,crc_m3           ;BX=CRC_M3
            mov     cl,8
            shr     bx,cl               ;BX=BX ===> 8 lugares (BX=c0-c7)
            xor     bx,ax               ;BX=BX xor AX
            mov     crc_m3,bx           ;CRC_m3=BX (CRC temporal)
            pop     cx                  ;recupera CX
            loop    p_fly1              ;lazo hasta que CX=0

            mov     ah,4ch
            int     21h
            RET
START ENDP
    CSEG ENDS
END START
```

```
;LN6_TM1.ASM
;
;   ;; PROGRAMA PARA CALCULAR EL TIEMPO REQUERIDO POR LOS ;;
;   ;; DIFERENTES ALGORITMOS PARA CALCULAR EL CRC DE UN   ;;
;   ;; BUFFER DE MENSAJE DE LONGITUD VARIABLE ;;

;   AREA DEL STACK
    SSEG SEGMENT STACK
    DB 256 DUP (?)
    SSEG ENDS

;SIMBOLOS
CR          EQU         13
LF          EQU         10

; Datos para lectura de archivos
NB          EQU         1024

; Datos para calculo del CRC
CRC_INIC    EQU         0000H
SHFT        EQU         8
CRC_IT      EQU         0000H
LONG_T      EQU         256

; Datos para el tiempo de calculo
CNT_REP     EQU         1               ;numero de repeticiones
L_INIC      EQU         1024            ;longitud inicial del buffer
N_DAT       EQU         10              ;numero de datos a obtener

;   AREA DE DATOS
    DSEG SEGMENT
;   Datos para salida en pantalla
VIDEOSEG    DW          0
NORM        DB          7
BOLD        DB          0FH
REVERS      DB          70H
PDDIR       DW          0
BUFF_PANT   DB          4000 DUP (0)
LONGIN      DB          0
PALTEC      DW          ?
X           DB          0
Y           DB          0

;   Datos para manejo de archivo
ARCH_MSG    DB          'A:\CAP_11.MSG',0  ;archivo de mensaje a leer
HANDLE      DW          ?
NB_ALR      DW          NB
NB_LEC      DW          ?
ER_LEC      DW          ?
BUFF_MSG    DB          NB DUP (?)
MENSE6      DB          'Error en Lectura del Archivo',0
```

```
;   Datos para calculo del CRC
LONG        DW          0
CRC_M1      DW          0
CRC_M2      DW          0
CRC_M3      DW          0
TABLA_1     DW          512 DUP (?)
BINARIO     DB          8 DUP (?)
PERMUT      DB          NB DUP (?)

MENS51      DB          'CALCULO DEL CRC-CCITT',0
MENS52      DB          'TIEMPO DE EJECUCION VS'
            DB           ' LONGITUD DEL MENSAJE',0
MENS53      DB          'BIT A BIT',0
MENS54      DB          'BYTE CON TABLA',0
MENS55      DB          'BYTE ON THE FLY',0
MENS56      DB          'LONGITUD VS TIEMPO',0
MENS57      DB          'LONGITUD (Bytes)',0
MENS58      DB          'ESC Para Salir',0
MENS59      DB          'FORMATO DE TIEMPO: mm:ss:dcm',0

;   Datos para el tiempo de calculo del CRC
INT1CH      DD          ?
CONTS       DW          0
CONTI       DW          0
TIME_I      DW          0
TIME_S      DW          0
HOR         DB          0
MIN         DB          0
SGDS        DB          0
DEC         DB          0
CEN         DB          0
MIL         DB          0
HORA        DB          10 DUP ('?')
HOR1_1      DB          N_DAT DUP (10 DUP ('?'))
HOR2_1      DB          N_DAT DUP (10 DUP ('?'))
HOR3_1      DB          N_DAT DUP (10 DUP ('?'))
DAT_1       DB          N_DAT DUP (6 DUP (0))

    DSEG ENDS
;
IF1
   include mac_crc.asm
   include mac_pant.asm
   include mac_arch.asm
ENDIF

;   AREA DE CODIGO
    CSEG SEGMENT
    ASSUME CS:CSEG,DS:DSEG,SS:SSEG,ES:NOTHING


START PROC FAR
```

```
:ETIQUETA  OPC    OPER              COMENTARIO:
           pusn   ds                :guarda direccion de retorno
           sub    ax,ax              :al DOS
           pusn   ax

           push   ds

           mov    ax,dseg            ;inicializa segmento de datos
           mov    ds,ax

           cli                       :direcciona el vector de
           mov    ax,351ch           :interrupcion 1ch con el offset
           int    21h                :de la rutina CNT_INT
           mov    word ptr int1ch,bx
           mov    word ptr [int1ch+2],es

           assume ds:nothing
           push   ds
           mov    ax,cs
           mov    ds,ax
           mov    ax,251ch
           mov    dx,offset cnt_int
           int    21h
           pop    ds
           assume ds:dseg
           push   es

           mov    ax,40h             :obtiene segmento de video
           mov    es,ax
           mov    ax,es:[63h]
           mov    dx,0b000h
           cmp    ax,3b4h
           je     dirvid
           add    dx,800h
dirvid:    mov    videoseg,dx
           mov    es,dx

           mac_crc
           mac_pant
           mac_arch

           borrpant
           ventana 21,20,24,58,norm,3
           print   29,22,mens56,bold

           call   lec_msg

           mov    ax,nb_lec
           sub    ax,1               :AX=longitud del mensaje
           mov    long,ax
```

```
          call   permutar
          call   crcxi

          mov    ax,i_inic          ;AL=longitud inicial de mensaje
          mov    long,ax
          mov    bx,offset hor1_1
          mov    cx,n_dat           ;CX=numero de datos a obtener
l_tmo1:   mov    ax,0
          mov    conti,ax           ;calculo del CRC y tiempo de
          mov    conts,ax           ;ejecucion con el metodo
          sti                       ;BIT a BIT
          push   cx
          mov    cx,cnt_rep
l_tmp2:   call   bit
          loop   l_tmp2

          cli
          call   time
          lea    si,hora
          call   prt_time

          mov    ax,long
          mov    cx,2
          mul    cx
          mov    long,ax
          pop    cx
          loop   l_tmo1

          mov    ax,i_inic          ;calculo del CRC y tiempo de
          mov    long,ax            ;ejecucion con el metodo por
          mov    bx,offset hor2_1   ;byte con TABLA
          mov    cx,n_dat
l_tmp3:   mov    ax,0
          mov    conti,ax
          mov    conts,ax
          sti
          push   cx
          mov    cx,cnt_rep
l_tmp4:   call   byte_tab
          loop   l_tmp4

          cli
          call   time
          lea    si,hora
          call   prt_time

          mov    ax,long
          mov    cx,2
          mul    cx
          mov    long,ax
          pop    cx
          loop   l_tmo3
```

```
        mov   ax,l_inic          ;calculo del CRC y tiempo de
        mov   long,ax            ;ejecucion para el metodo por
        mov   bx,offset hor3_1   ;byte "on the fly"
        mov   cx,n_dat
l_tmp5: mov   ax,0
        mov   conti,ax
        mov   conts,ax
        sti
        push  cx
        mov   cx,cnt_rep
l_tmp6: call  byte_fly
        loop  l_tmp6

        cli
        call  time
        lea   si,hora
        call  prt_time

        mov   ax,long
        mov   cx,2
        mul   cx
        mov   long,ax
        pop   cx
        loop  l_tmp5

        mov   cx,n_dat           ;salida en pantalla de los
        mov   ax,l_inic          ;resultados
        mov   bx,0
l_tmp8: push  cx
        push  ax
        mov   cx,5
l_tmp7: mov   byte ptr dat_1[bx],' '
        inc   bx
        loop  l_tmp7

        push  bx
        mov   si,10

l_tmp9: sub   dx,dx
        div   si
        add   dl,'0'
        dec   bx
        mov   byte ptr dat_1[bx],dl
        or    ax,ax
        jnz   l_tmp9

        pop   bx
        mov   dl,0
        mov   byte ptr dat_1[bx],dl
        inc   bx
        mov   cx,2
```

```
            pop     ax
            sui     cx
            pop     cx
            loop    l_tmp8

            call    pantl_tmp
            print   46,23,mens58,bold
            noscur 62,23
l_tmp10:    esp_tec
            mov     ax,paltec
            cmp     al,01bh
            jne     l_tmp10

            sti
            pop     es
            mov     ax,251ch
            lds     dx,[int1ch]
            int     21h

            mov     ah,4ch
            int     21h

RET
START ENDP

; Procedimientos adicionales
;
;   ****   PANTALLA DE PRESENTACION   ****
PANTL_TMP  PROC   NEAR
           guarda
           borrpant
           ventana 2,10,6,70,norm,4
           print  30,3,mens51,bold
           print  18,4,mens52,norm
           print  4,8,mens57,bold
           print  26,8,mens53,bold
           print  42,8,mens54,bold
           print  61,8,mens55,bold
           print  3,23,mens59,norm
           ventana 10,2,21,77,norm,12
           mov    x,10
           mov    y,11
           print_1 x,y,dat_1,norm,n_dat
           mov    x,26
           mov    y,11
           print_1 x,y,hor1_1,norm,n_dat
           mov    x,45
           mov    y,11
           print_1 x,y,hor2_1,norm,n_dat
           mov    x,64
           mov    y,11
           print_1 x,y,hor3_1,norm,n_dat
```

```
          recubera
          ret
PANTL_TMP ENDP
;
;   ¡¡¡¡¡   OBTIENE EL TIEMPO REQUERIDO POR CADA ALGORITMO PARA
;           EL CALCULO DEL CRC   ¡¡¡¡¡
TIME      PROC   NEAR
          guarda
          mov    ax,conti           ;inicializa contadores
          mov    time_i,ax
          mov    ax,conts
          mov    time_s,ax
          mov    ax,time_i
          mov    dx,time_s

          mov    ax,32771           ;obtiene el numero de horas
          div    ax
          shr    ax,1
          mov    hor,al

          mov    ax,dx              ;obtiene los minutos transcurridos
          mov    dx,0
          mov    ax,1092
          div    bx
          mov    min,al

          mov    ax,dx              ;obtiene los segundos transcurridos
          mov    dx,0
          mov    ax,18
          div    bx
          mov    sgds,al

          mov    ax,dx              ;obtiene las decimas de segundo
          mov    bx,10
          mul    bx
          mov    dx,0
          mov    bx,18
          div    bx
          mov    dec,al

          mov    ax,dx              ;obtiene las centesimas de segundo
          mov    bx,10
          mul    bx
          mov    dx,0
          mov    bx,18
          div    bx
          mov    cen,al

          mov    ax,dx              ;obtiene las milesimas de segundo
          mov    bx,10
          mul    bx
          mov    dx,0
```

```
                mov     bx,10
                div     bx
                mov     mil,al

                mov     bh,10               ;graba en memoria los minutos
                mov     ah,0
                mov     al,min
                div     bh
                add     ax,3030h
                mov     word ptr hora,ax
                mov     hora+2,':'

                mov     ah,0                ;graba en memoria los segundos
                mov     al,segs
                div     bh
                add     ax,3030h
                mov     word ptr hora+3,ax
                mov     hora+5,':'

                mov     ah,0                ;graba en memoria las decimas de
                mov     al,dec              ;segundo
                div     bh
                add     ah,30h
                mov     byte ptr hora+6,ah

                mov     ah,0                ;graba en memoria las centesimas
                mov     al,cen              ;de segundo
                div     bh
                add     ah,30h
                mov     byte ptr hora+7,ah

                mov     ah,0                ;graba en memoria las milesimas
                mov     al,mil              ;de segundo
                div     bh
                add     ah,30h
                mov     byte ptr hora+8,ah
                mov     al,0
                mov     ds:[hora+9],al
                recupera
                ret
TIME            ENDP

;   !!!!!   NOTA: LAS OTRAS RUTINAS LLAMADAS SON LAS MISMAS DEL PROGRAMA
;   !!!!!   CRC.EXE POR ESTA RAZON NO SE VAN A REPETIR EN EL PRESENTE
;   !!!!!   PROGRAMA

    CSEG ENDS

END START
;Fin del programa indicando punto de entrada.
```

MANUAL DEL USUARIO

Archivos .-

Se requieren los siguientes archivos:

CRC.EXE: Programa principal que contiene la secuencia de la transmisión y el control de errores con el CRC. Puede encontrarse en cualquier directorio o subdirectorio.

EDITOR.COM: Programa para la edición del mensaje a transmitir. El NORTON EDITOR es el utilizado en la tesis. Siempre debe estar en la unidad de disco A.

Además puede disponerse de archivos de mensaje editados previamente, y grabados con la extensión **.msg** .

Ingreso al Programa.-

Para invocar al programa debe ingresarse:

CRC

Escoger una Opción en el Menú.-

Para cada nivel de la secuencia de transmisión se presenta el menú, se escoge una opción presionando la tecla correspondiente a la letra resaltada y subrayada, mayúscula o minúscula.

Modos de Operación.-

Los modos de operación son:

- Modo de Transmisión, y
- Modo de Recepción

El sistema de comunicación es simplex, por tanto, el computador que va a enviar el mensaje debe estar en modo de transmisión, y el otro en modo de recepción.

MODO DE TRANSMISION.-

En este manual cada nivel en la secuencia se indica con la numeración del menú correspondiente. Se comienza con el menú 1.

Al ingresar al programa se presenta una pantalla con el menú inicial, en el que debe escogerse el modo de operación.

Para trabajar en <u>modo de transmisión</u> hay que seguir el procedimiento indicado a continuación:

Menú 1.- De las cuatro opciones escoger las siguientes:

Al ingresar al editor se carga el archivo nombre.MSG. Donde nombre es el introducido en el paso anterior. Si es un archivo nuevo está listo para empezar su escritura.

Los mensajes pueden tener una extensión máxima de 8 Kbytes.

Del Norton Editor.-

Para consultar las opciones del editor presionar la tecla F1, que es la ayuda.

Cuando se ha terminado de escribir el mensaje salir del editor presionando:

F3 E (Función Grabar y Salir)

Menú 3.- Escoger la opción:

Cálculo del CRC: Presenta el CRC calculado con los tres algoritmos y el tiempo requerido por cada uno de ellos para obtener el CRC del bloque número uno de mensaje.
Además calcula los dos bytes del CRC para cada bloque del mensaje y almacena estos valores en memoria.

El tiempo se presenta en milisegundos.

Nota: Si el computador tiene una velocidad muy baja y el tiempo de cálculo excede a los 600 milisegundos se presenta una pantalla que indica en un formato de minutos, segundos y milésimas de segundo el tiempo total requerido para repetir el cálculo del CRC un número de veces que se señala en **Núm. Repeticiones.**

La longitud del bloque de mensaje sobre el que se ha realizado el cálculo se indica en **LONG.**

Menú 4.- Resalta las opciones:

Transmisión del mensaje: Envía los bloques del mensaje original (sin errores), con el respectivo CRC al computador remoto.

Después de recibir la aceptación (ACK) del último bloque de mensaje se envía el carácter de fin de transmisión y se retorna al menú 6 que permite salir del programa al DOS o continuar trabajando tanto en el mismo modo como cambiar al de recepción.

Introducción de errores: Permite modificar el mensaje original, simulando la ocurrencia de errores en la línea de transmisión.

Para introducir errores el programa principal entrega el control al editor cargando el archivo de mensaje original nombre.MSG.

Se puede cambiar el texto sin añadirlo, en cualquier lugar utilizando la función:

F6 INS (Sobreescribir)

Al terminar de introducir errores utilizar la siguiente función del editor:

F3 E (Grabar y salir)

Archivo de Mensaje con Errores.-

El mensaje modificado se graba automáticamente en el archivo denominado: nombre-E.MSG .

Cuando se presiona la tecla de Introducción de Errores, a continuación se presenta el menú 5.

Menú 5.- Resalta la opción:

Transmisión del mensaje: Envía los bloques del archivo de mensaje con errores y el CRC respectivo.
Utiliza el método de retransmisión para y espera.

Retransmisión Para y Espera.- La retransmisión para y espera consiste en enviar un bloque de información con el CRC

respectivo y esperar el acuse de recibo del computador remoto. Si es positivo (ACK) se envía el siguiente bloque; si es negativo (NACK) se retransmite el mismo bloque.

Al terminar de transmitir todos los bloques el programa envía el carácter de fin de transmisión y retorna al menú 6.

Menú 6.- Resalta las siguientes opciones:

Fin

Transmisión del mensaje

Cambio de Directorio

Cambio de Archivo

Introducción de errores

Modo de recepción

Opción FIN.-

En todos los niveles del programa se resalta la opción:

Fin: Permite salir del programa al DOS.

MODO DE RECEPCION.-

Para trabajar en <u>modo de recepción</u> deben seguirse los siguientes pasos:

Menú 1.- Junto con las opciones señaladas para el modo de transmisión se resalta la siguiente:

Archivar el mensaje: Graba el bloque de mensaje en el archivo NOMBRE.RCP.

El programa envía seguidamente el acuse de recibo positivo (ACK) y se pone en espera de la información.

Si el mensaje tiene errores únicamente se transmite al computador emisor el acuse de recibo negativo (NACK) y se pone en espera de un bloque.

Fin de la Recepción.-

Cuando se recibe el carácter de fin de transmisión se indica el directorio y nombre del archivo en el cual se ha grabado el mensaje y se solicita presionar la tecla "L o l" para presentar en la pantalla el listado del mismo.

Listado del mensaje: Presenta en pantalla el mensaje recibido para facilitar su lectura.

Si el mensaje ocupa más de una página por medio de la tecla ENTER puede pasarse a la siguiente, así hasta el final que es cuando presionando ESC se retorna al menú 1 que es el inicial del programa CRC.

Nota: El computador que va a recibir el mensaje debe inicializarse y encontrarse con la pantalla que contiene una ventana de **RECEPCION** antes de que el remoto le envíe la información.

BIBLIOGRAFIA

- Alabau,Antonio, Teleinformática y Redes de Computadores, Marcombo Boixareu Editores, Barcelona, 1984.

- Albertengo, Guido, Parallel CRC Generation , IEEE MICRO, Volumen 10, Número 5, Octubre 1990, Pgs. 63-71.

- CCITT, Libro Rojo del CCITT, Fasc. VIII.1, Rec. V.41.

- Coates, R., Modern Communication Systems, Macmillan, London, 1981.

- DEC, Introduction to Data Communication Conceps, Digital Equipment Corporation, Maynard, 1975.

- DEC, VAX Architecture Handbook, Digital Equipment Corporation, Massachusetts, 1981.

- Egas, Carlos, Codificadores y Decodificadores para Códigos de Bloque Lineales, EPN, 1987.

- Gallager, Robert, Information Theory and Reliable Communications, John Wiley, New York, 1968.

- Gofton, Peter, Mastering Serial Communications, SYBEX, Berkeley, 1986.

- Hamming, Richard, Coding and Information Theory, Prentice Hall, New Jersey, 1980.

- IETEL, Códigos Detectores y Correctores de Errores, Marzo 1985.

- Kruglinski, David, Guía a las Comunicaciones del IBM PC, Mc Graw-Hill, México, 1985.

- Perez, Aram, Byte-wise CRC Calculations, IEEE MICRO, Volumen 3, Número 3, Junio 1983, Pgs. 40-50.

- Prosise, Jeff , Lab Notes, PC MAGAZINE, Septiembre 1989, Pgs. 307-319.

- Robert, Mc. Eliece, The Theory of Information and Coding,

- Rodriguez Miguel Angel, Programación Ensamblador en Entorno MS DOS 8088-8086/8087, ANAYA MULTIMEDIA, Madrid, 1987.

- Shu Lin y Costello, Error Control Coding: Fundamentals and Aplications, Prentice-Hall, New Jersey, 1983.

- Shouse, D.V. , "On the Fly" CRC-16 Byte-wise Calculation for 8088-based Computers , IEEE MICRO, Volumen 5, Número 2, Abril 1985, Pgs. 67-75.