11
2ej

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS
PROFESIONALES
A R A G O N

INTEGRACION, PRUEBAS Y MANEJO DE INTERRUPCIONES EN LAS UNIDADES PROCESADORAS DE UNA COMPUTADORA DE PROCESAMIENTO CONCURRENTE

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

P R E S E N T A :

RAFAEL DIAZ BARRIGA SANCHEZ

ASESOR : M. EN I. ESAU VICENTE VIVAS

SAN JUAN DE ARAGON 1996

A mis padres, y a mi hermana.

Este trabajo y lo que representa no habría sido posible sin la participación de quienes ahora agradezco:
Principalmente a Esad Vicente, por su enseñanza y paciencia.
A Gerardo Tornos, Martín López y Víctor Mata, por su ayuda y consejos.
A mis compañeros de la EINSP Aragón y de la sección de automatización del Instituto de Ingeniería por su constante interés y su amistad.
A mi familia y a mis amigos por su apoyo y su ejemplo.
A Max y a todo el equipo de sistemas de Acetlan por el impulso y el consejo.

Gracias a todos.

Rafael Díaz Bravo Sánchez.

ÍNDICE

RESUMEN

En la presente tesis se describe la integración de los circuitos impresos de las unidades procesadoras, así como las pruebas realizadas con las diversas unidades que componen la computadora tolerante a fallas (CTF) desarrollada en el Instituto de Ingeniería de la UNAM.

Las pruebas realizadas incluyen las de ensamblado de circuitos impresos, las modulares al autómata de operación de las unidades de procesamiento, pruebas efectuadas a las tarjetas de aislamiento y las pruebas de integración de la arquitectura completa. La importancia de esta última etapa radica en que de una buena interacción entre las principales unidades que integran la arquitectura de la CTF, como Unidades de Procesamiento (UP) y Unidades de Detección de Fallas y Control (UDFC), dependen las labores de detección de fallas, diagnóstico, aislamiento de fallas, mantenimiento en línea y operación continua.

La tesis incluye el diseño de algoritmos que permitan realizar programas para el manejo de interrupciones, los cuales serán utilizados durante el manejo preventivo o correctivo de cualquiera de las tres unidades procesadoras. Parte de estos procedimientos incluye el respaldo de la información completa de un procesador en buen estado para posteriormente inicializar a un procesador recién instalado en el sistema.

CAPÍTULO

UNO

INTRODUCCIÓN

"Si algo puede fallar, fallará".
Ley de Murphy.
"...fallará en el peor momento".
Corolario.

1.1 Antecedentes

El avance de la civilización ha ido acompañado por una necesidad siempre creciente de cálculos numéricos y el manejo de volúmenes de datos cada vez más grandes. Por consiguiente, conforme pasa el tiempo el ser humano ha inventado sistemas de cálculo más rápidos y capaces de manejar grandes cantidades de información que le ayudan a administrar de manera más eficiente su tiempo.

Con la introducción de los sistemas digitales de cómputo en la vida del hombre, pareceria que se ha llegado muy cerca de la perfección en los sistemas de cálculo. Sin embargo, también es cierto que dicho perfeccionamiento es relativo, pues aunque en la actualidad contamos con poderosas computadoras, que realizan en unos cuantos segundos tareas que tomarían años por otros medios, éstas se pueden equivocar, es decir, existe la posibilidad de que incurran en fallas.

Es por esto que desde la década de los 60s se acentuó el interés en sistemas tolerantes a fallas, y en los 70s aparecen ya equipos sofisticados con los cuales se consolidaron aplicaciones tales como el control de vías férreas, control de tráfico aéreo, monitoreo de reactores nucleares, etc.

Actualmente, podemos hablar de un nuevo y acrecentado auge del área de las computadoras tolerantes a fallas debido a la multiplicidad e importancia

de las aplicaciones informáticas y más aún, debido al papel estratégico de los equipos de cómputo en aplicaciones críticas.

En cuanto a los tipos de sistemas tolerantes a fallas reportados en la literatura, los sistemas para aplicaciones críticas basadas en redundancia de circuitería y el análisis de señales electrónicas en tiempo real, son el tipo de esquema más atractivo para el usuario, porque las tareas de detección, diagnóstico y reconfiguración las efectúa automáticamente la electrónica del equipo.

La programación de aplicaciones se realiza de manera convencional sin que los atributos tolerantes a fallas condicionen modos particulares de trabajo.

Otra de las características más atractivas de estos sistemas radica en que la detección, diagnóstico y recuperación se lleva a cabo en el instante en que ocurre la falla, por lo cual la aplicación continúa desarrollándose como si nada hubiera pasado. Esta forma de trabajo, además, evita la deformación de información o la propagación de errores hacia bases de datos o resultados generados. Esta arquitectura es la que se considera como un verdadero sistema tolerante a fallas.

1.2 El procesamiento concurrente como técnica de tolerancia a fallas

Los sistemas tolerantes a fallas (STF) son aquellos que realizan correctamente su tarea encomendada aún en presencia de anomalías en la electrónica *(hardware)* . Por ejemplo, el efecto de un error en algún módulo electrónico de un STF se sobrelleva de tal forma que no perjudica la operación correcta y continua del sistema.

La tolerancia a fallas (TF) es una filosofía de diseño que integra atributos en un sistema para que este alcance altos índices de desempeño.

Todas las filosofías de diseño requieren el uso de elementos redundantes para lograr su cometido. Los primeros diseños de STF tomaban el concepto de redundancia únicamente a nivel físico, esto es, se empleaban réplicas de componentes o de módulos enteros para lograr sus atributos de tolerancia a fallas.

En nuestros días se tiene un mejor entendimiento del concepto de redundancia y de los diferentes tipos de esta. Una redundancia es simplemente la adición de información, recursos o tiempo necesarios para la operación normal del sistema.

Existen cuatro tipos principales de redundancias: redundancia en el tiempo, redundancia en la información, redundancia por *software* y redundancia por *hardware*.

La redundancia en el tiempo emplea periodos extra en la ejecución de alguna tarea con el objeto de cumplir con las labores de detección y tolerancia a fallas. La redundancia en la información implica la incorporación de información complementaria, excediendo la necesaria para desarrollar una

determinada función; por ejemplo el uso de códigos de detección de errores. La redundancia por *software*, es aquella en que la detección y tolerancia a fallas se puede implementar por programación. En lo que se refiere a la redundancia *por hardware*, se ampliarán detalles en las secciones siguientes, ya que dicho concepto es de mayor trascendencia para los fines de este trabajo.

1.2.1 Redundancia *por hardware*

Este tipo de redundancia se refiere a la adición de componentes complementarios, con el propósito de ayudar en las tareas de detectar o tolerar fallas. Esta repetición física de componentes es tal vez la forma más común de redundancia que se usa en los sistemas digitales, la cual es una práctica usual, ya que los costos de repetición se abaten en cierto grado con la reducción en precios de los semiconductores.

Existen tres formas de llevar a cabo la *redundancia por hardware:* pasiva, activa e híbrida. Las técnicas pasivas se emplean para enmascarar fallas ocultando así la ocurrencia de éstas. Las técnicas activas (o dinámicas) hacen uso intensivo de la detección de fallas, localización de las mismas y recuperación. La combinación de los procedimientos pasivos y activos deriva en la obtención de criterios híbridos.

La idea básica de la redundancia híbrida *por hardware* es el incorporar las bondades de las técnicas activas y pasivas. El enmascaramiento de fallas se aprovecha para evitar que el sistema produzca resultados erróneos, mientras que la detección, ubicación y reemplazo de fallas son útiles para las operaciones de reconfiguración en caso de presentarse alguna falla. Las técnicas de redundancia híbrida se utilizan en aplicaciones que requieren alta integridad en las operaciones.

Dentro de la redundancia híbrida, los módulos redundantes que participan en el procesamiento de la información, son característicos del procesamiento concurrente.

El procesamiento concurrente es identificado como el uso de varias entidades realizando el mismo trabajo. Estas entidades (computadoras) procesan la misma información, leen los mismos datos y efectúan idénticas operaciones.

Algunas de las técnicas de redundancia híbrida *por hardware* que utilizan procesamiento concurrente son: la redundancia de autodepuración, la redundancia modular selectiva y la arquitectura triplex-duplex, que se describen enseguida:

- Redundancia de autodepuración. En el presente modelo, los módulos de repuesto participan con sus resultados en un proceso de voteo. El circuito de voteo recibe las salidas de los N módulos que conforman el sistema, el funcionamiento de cada módulo depende de la salida del circuito de voteo,

ya que de existir alguna discrepancia, el módulo con falla es desconectado del sistema mediante un interruptor asociado.
Es pertinente señalar que el circuito de voteo basa su operación en umbrales ponderados e involucra el uso de elementos analógicos; Sin embargo, este esquema es extremadamente atractivo por sus cualidades de autodepuración que facilitan el mantenimiento y la reparación, puesto que no es necesaria la interrupción de la operación del equipo para el reemplazo de módulos dañados. (Ver figura 1).

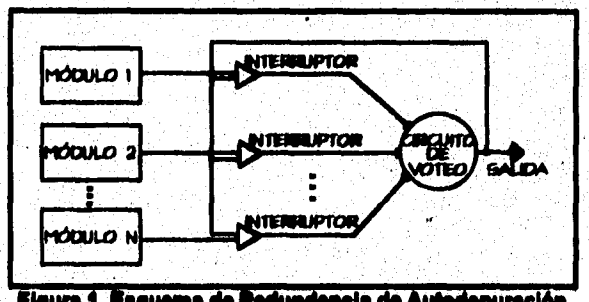


Figura 1. Esquema de Redundancia de Autodepuración.

- Redundancia modular selectiva. En este modelo se tienen N módulos idénticos que componen el sistema junto con tres circuitos especiales denominados: comparadores, detectores y colectores.
  La función del circuito comparador es equiparar la salida de un módulo con las salidas del resto de los módulos, produciendo una salida por cada comparación realizada.
  El detector determina las discrepancias reportadas por el comparador y deshabilita aquella unidad que difiera de la mayoría restante. Este circuito produce una salida por cada módulo del sistema, indicando su condición operativa (falla o buen estado).
  El circuito colector se encarga de generar la salida del sistema, esto lo hará en función de las salidas de cada módulo y de los parámetros proporcionados por el detector indicando los módulos con falla. (Ver figura 2 en la siguiente página).

- Arquitectura triplex - duplex. Esta técnica combina la redundancia modular triple y la duplicación con comparación. El esquema de redundancia modular triple utiliza tres módulos, ya que es el número mínimo requerido para un voteo mayoritario. Este esquema permite enmascarar fallas y la continuidad en la operación logrando con ello el buen desempeño de al menos un módulo sin falla. La técnica de duplicación con comparación

facilita la detección de fallas y la remoción de los módulos con falla que el proceso de voteo indique. (Ver figura 3).

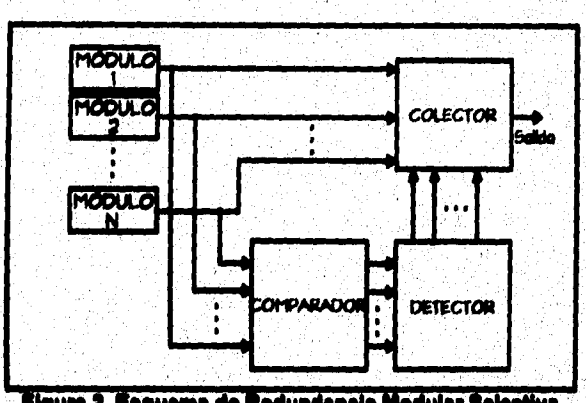


Figura 2. Esquema de Redundancia Modular Selectiva.

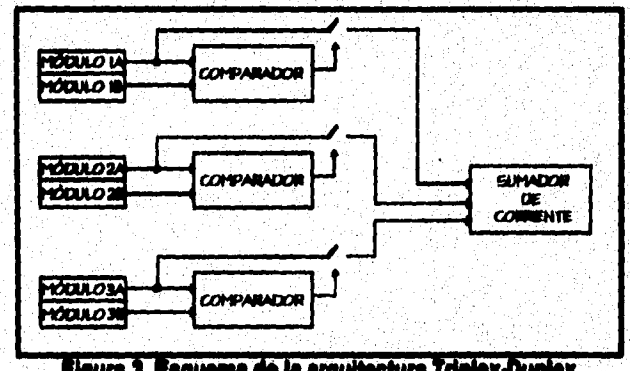


Figura 3. Esquema de la arquitectura Triplex-Duplex.

1.3 Aplicaciones de la computación de procesamiento concurrente

Las aplicaciones de la computación tolerante a fallas pueden dividirse principalmente en aplicaciones de larga vida y aplicaciones de cómputo crítico.

Las aplicaciones de larga vida deben tener una muy alta probabilidad de seguir siendo operativas al final de largos periodos, un ejemplo de estas

aplicaciones son los satélites y sondas espaciales, ya que se encuentran aislados, aunque a veces es posible configurarlos remotamente.

En cuanto al cómputo crítico, es aquel en que los cálculos y la operación continua (segura) son críticos en diversas ramas productivas, para la seguridad humana, el cuidado del medio ambiente o para la seguridad del mismo equipo, un ejemplo de esta aplicación es una red de servicios bancarios, ya que cualquier falla provocaría grandes pérdidas.

Una computadora tolerante a fallas está orientada principalmente para aquellas aplicaciones en donde se requieren servicios continuos y confiables, en donde las implicaciones por desperfectos conllevan grandes pérdidas económicas e incluso, humanas.

Para dar una idea del tipo de aplicaciones posibles , podemos citar: el control de reactores químicos, control de centrales de generación de energía eléctrica, control y supervisión de sistemas de transporte semiautomático, control de señalizaciones y asignación de vías férreas, control de procesos industriales en general, servidores de red, controladores de redes de conmutación telefónica, etcétera.

CAPITULO

DOS

DESCRIPCIÓN GENERAL DE LA ARQUITECTURA DE PROCESAMIENTO CONCURRENTE

2.1 Introducción

En este capítulo se describe la arquitectura electrónica de una Computadora Tolerante a Fallas (CTF), que permite tolerar fallas en sus componentes y sus subsistemas principales. Se da un panorama general y se explican los diferentes módulos que integran una arquitectura de procesamiento concurrente basada en tres procesadores y dos unidades de detección de fallas y control. También, se menciona la necesidad de utilizar tarjetas de aislamiento para permitir el retiro de módulos en mal estado y la adición de tarjetas en buen estado con el fin de lograr la operación continua del equipo. Finalmente, se describe el uso de una unidad de interfaz múltiple integrada en la CTF, para la conexión de los periféricos esenciales del sistema.

En el siguiente capítulo se describe detalladamente la arquitectura de las unidades procesadoras basadas en el microprocesador 386SX.

2.2 Detalles generales de operación de la CTF

La arquitectura de la CTF está formada por seis tarjetas objetivo, cada una vinculada a una tarjeta de aislamiento. Tres de las tarjetas objetivo constituyen tres sistemas completos basados en procesadores 386SX, a los que en adelante se denominará unidades de procesamiento (UPs); otras dos forman las unidades de detección de fallas y control (UDFCs) y la última es la unidad de interfaz múltiple (UIM), que es una tarjeta auxiliar en donde se conecta un pequeño teclado para labores de mantenimiento preventivo-correctivo, así como el teclado principal y la bocina comunes a las tres unidades de procesamiento. (Ver figura 4).

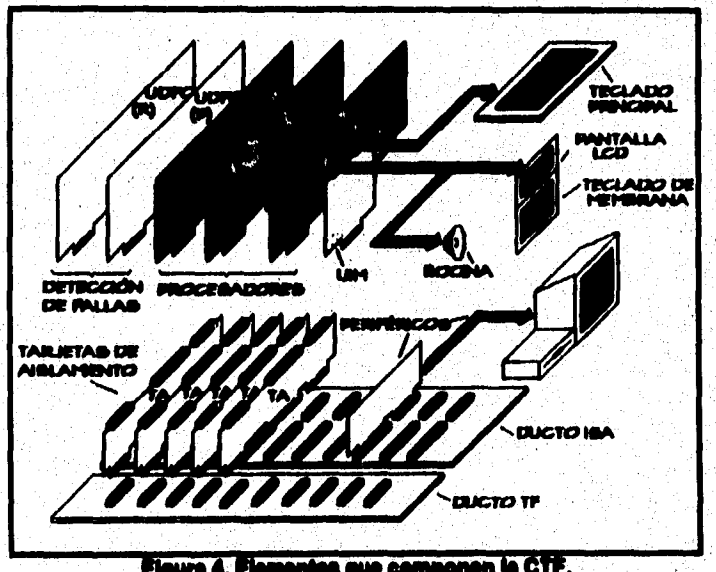


Figura 4. Elementos que componen la CTF.

Las UPs trabajarán concurrentemente, es decir, procesarán la misma información, leerán los mismos datos y efectuarán idénticas operaciones. Para evitar colisiones en los ductos, debido a la operación concurrente, sólo una de las UPs tiene el derecho de escribir y enviar datos a los periféricos instalados. Durante el arranque, la UDFC principal establece cual de las UPs será la principal y cuales las redundantes. En otras palabras los datos generados por los procesadores redundantes se utilizan, junto con los del

principal, para comparar información en tiempo real que permita detectar cualquier tipo de error generado en las tres tarjetas.

En caso de fallas en cualquiera de los componentes de la UP principal se utiliza alguna de las UPs redundantes para continuar la carga de trabajo encomendada sin interrupciones y sin pérdida de información. Si la falla ocurre en alguna de las tarjetas redundantes, se envía un reporte al usuario para que proceda a reemplazar inmediatamente la tarjeta averiada.

La razón de utilizar tres sistemas idénticos en la arquitectura es contar con un número mínimo adecuado de elementos para efectuar un voteo electrónico cuyo resultado sea apoyado por mayoría y sin lugar a equivocaciones. Para esto se subraya el hecho de que al diseño se basa en la suposición de que ocurre una sola falla a la vez, principio que es utilizado en otros sistemas disponibles.

La detección de fallas se efectuará por comparación de los datos generados por los procesadores mediante circuitos electrónicos de voteo. De esta forma, será posible detectar fallas en líneas de control, datos y direcciones o bien, por fallas en la memoria o en el mismo procesador, pues en cualquiera de las anomalías anteriores se propagan errores en los datos procesados que van desde la ausencia de ellos, hasta la parcial o completa disparidad de los mismos.

2.3 Unidades de procesamiento

Cada tarjeta cuenta con un procesador 386SX (16 bits), puede contener desde 1 Mb hasta 16 Mb de memoria DRAM, EPROM BIOS, un circuito integrado de aplicación específica que auxilia al procesador para interactuar con la memoria dinámica, para multiplexar los ductos de datos internos y para aceptar y generar las señales del ducto estándar conocido como ISA *(Industry Standard Architecture)*.

Las tarjetas cuentan con peines de conexión totalmente compatibles con las PC-AT y adicionalmente, tienen peines que se conectan a un ducto propietario en el que se tienen líneas de control y datos necesarios para poder realizar la supervisión de fallas, su diagnóstico y la reconfiguración de los procesadores.

Cada UP cuenta además, con un autómata electrónico totalmente independiente del microprocesador, cuya función es complementar el control de la arquitectura tolerante a fallas.

El objetivo de esta tesis se centra en las UPs, por lo que su arquitectura y funcionamiento se detallan en el capítulo siguiente.

2.3.1 Unidades de aislamiento para UPs

Cada tarjeta de procesamiento se conecta a una tarjeta de aislamiento mediante la cual se controla dinámicamente el tráfico de información de los procesadores. En operación normal, estas tarjetas permiten el semi-aislamiento de los procesadores redundantes para que puedan aceptar la misma información que el principal, pero impedirán que más de una UP envíe señales de datos y control al ducto PC-AT durante escrituras, evitando así posibles colisiones.

El control de la tarjeta se realiza mediante lógica electrónica que toma decisiones en forma dinámica para adoptar un modo de operación acorde con el tipo de anomalías detectadas.

En presencia de alguna falla, las tarjetas de aislamiento permiten desconectar totalmente las líneas de datos, direcciones, control y alimentación de la tarjeta del procesador con falla.

Por medio de las tarjetas de aislamiento se permitirá al usuario intercambiar las tarjetas objetivo dañadas sin necesidad de apagar el equipo, sin temor de ocasionar cortos circuitos en los diferentes puntos de los peines y por tanto, sin detener el programa de aplicación, de esta forma se logra el atributo de disponibilidad buscado, aun ante la presencia de fallas en cualquiera de las partes de las tarjetas de procesamiento.

Las tarjetas de aislamiento cuentan con peines de conexión, tanto al ducto PC-AT como al ducto propietario (para actividades de tolerancia a fallas), de tal forma que el aislamiento y la protección que realizan es para ambos ductos.

2.4 Unidad de detección de fallas y control (UDFC)

La UDFC está compuesta por dos tarjetas que tienen el mismo circuito impreso: una es la principal y la otra actúa como respaldo en caso de fallas en la principal.

Cada tarjeta cuenta con una sección de comparadores electrónicos para realizar la detección de errores transitorios y permanentes de las UPs. Además, cuenta con dos autómatas electrónicos de alta velocidad con los que se controla la captura y multiplexaje de datos, así como el diagnóstico del estado de operación de las UPs y de los mismos comparadores. Adicionalmente, tiene incorporado un microcontrolador para realizar labores de control complicadas o interactivas junto con una serie de registros para efectuar el control de la arquitectura.

Debemos subrayar que la UDFC es la parte más importante de la arquitectura, de tal forma que su diseño, desarrollo y la explicación a detalle

de su arquitectura, así como las pruebas de su funcionamiento son motivo de tesis complementarias. Entre las tareas que realiza se encuentran:

A) Inicialización de cada procesador con el sistema operativo.
B) Arranque sincronizado de los tres procesadores.
C) Sostenimiento de la concurrencia sincronizada.
D) Asignación de la categoría a cada procesador (principal o redundante).
E) Comparación de los resultados generados por cada UP.
F) Detección de fallas e identificación de la tarjeta dañada.
G) Aislamiento de cualquier UP dañada y activación de un procesador redundante para que el sistema continúe operando.

2.4.1 Unidades de aislamiento para UDFCs

Cada UDFC se conecta a una unidad de aislamiento, la cual tiene por objetivo permitir la conmutación de la UDFC principal a la redundante, en caso de falla en la primera, la lógica electrónica envía una interrupción a la tarjeta redundante, indicándole la necesidad de que tome el control y la supervisión de la arquitectura. Ante esta situación la nueva UDFC desactiva a su contraparte y además, envía mensajes de alarma al usuario por medio de la Unidad de Interfaz Múltiple (UIM). La señal de desactivación llega a la unidad de aislamiento conectada a la tarjeta con falla provocando la desconexión completa de la tarjeta objetivo.

Como se comprenderá, la UDFC utiliza más el ducto tolerante a fallas (TF) que el PC-AT, por lo que las arquitecturas de las tarjetas de aislamiento para UPs y UDFCs son diferentes, aunque el principio de operación es similar.

2.5 Unidad de Interfaz Múltiple (UIM)

Para facilitar el manejo del teclado y la bocina compartidos por las UPs, así como conectar una pantalla de despliegue alfanumérico y un teclado de membrana (utilizados para que el usuario controle la arquitectura TF durante las labores de mantenimiento correctivo o preventivo), se decidió colocar todos los conectores en una tarjeta. Esta unidad periférica es un apéndice de la arquitectura, por lo cual su estado de operación no es crítica para el sistema y por tanto, no es necesario que opere con un esquema tolerante a fallas.

CAPÍTULO
TRES

ARQUITECTURA DE LAS UNIDADES PROCESADORAS

3.1 Introducción

La arquitectura de cada unidad de procesamiento cuenta con características tales como la electrónica necesaria para la sincronización con las otras dos UPs y la electrónica para compartir un teclado común. En las secciones siguientes se describen éstas y otras características, asimismo, las dos formas de utilización de cada UP que son: como uniprocesador sin redundancias y como procesador parte de un esquema de tolerancia a fallas. Finalmente, se explica el proceso de desarrollo de las tarjetas electrónicas que forman cada unidad de procesamiento.

En los capítulos subsecuentes se describen las pruebas realizadas a las diversas partes que integran cada unidad de procesamiento y las pruebas entre UPs y otras unidades de la CTF.

3.2 La arquitectura de las UPs

Cada UP está formada por dos secciones; la primer sección es donde ocurre el procesamiento de la información mientras que en la segunda sección se realizan todas las tareas relacionadas con la operación concurrente como son: la sincronización y la detección de operaciones.

La primer sección está constituida por:

- El microprocesador 386SX, que es un CPU de 32 bits de arquitectura interna, tiene 16 bits en el ducto de datos externo y 24 bits en el ducto de direcciones externo, cuenta con un reloj de 20 Mhz y tecnología de alta velocidad CHMOS. Además, tiene un empaque de 100 terminales el cual es de bajo costo, lo que lo hace accesible a sistemas pequeños.
- Un circuito integrado VLSI *(Very Large Scale Integration)*, que incorpora en un solo componente controladores de DMA 8237, controladores de interrupción 8259, manejadores de tiempo y contadores 8254, reloj de tiempo real y controladores de memoria, en adición a muchas otras funciones lógicas TTL y de interfaz, además de ofrecer total compatibilidad con la arquitectura estándar ISA PC-AT.
- Cuatro bases que forman un banco de memoria dinámico tipo SIMM con las cuales se pueden formar bancos de 1 a 16 Mbytes.
- BIOS *(Basic Input Output System)* en EPROM de 64 Kb expandible a 128 ó a 256 Kb para aplicaciones de tipo uniprocesador autosostenido.
- Un controlador de teclado 8742 para manejar un teclado convencional de PC.

La segunda sección se encuentra formada por:

- Un autómata electrónico totalmente independiente del microprocesador, cuya función es complementar el control de la arquitectura tolerante a fallas. Con él se frena y se supervisa el estado de operación del microprocesador para hacer posible la sincronización y concurrencia de las tres UPs.
- Dos circuitos integrados de registro tipo 74HC374, utilizados para la captura de datos; uno maneja los ocho bits más bajos y el otro los ocho bits más altos.
- Un interruptor analógico 74HC4066 para que el sistema triplex maneje el mismo teclado y la misma bocina.
- Un peine para las señales del ducto PC-AT y un ducto para las señales del ducto propietario TF.

En el diagrama de bloques de la figura 5 se muestran tanto la sección de procesamiento como la de sincronización y detección de operaciones, ambas integran cada unidad de procesamiento.

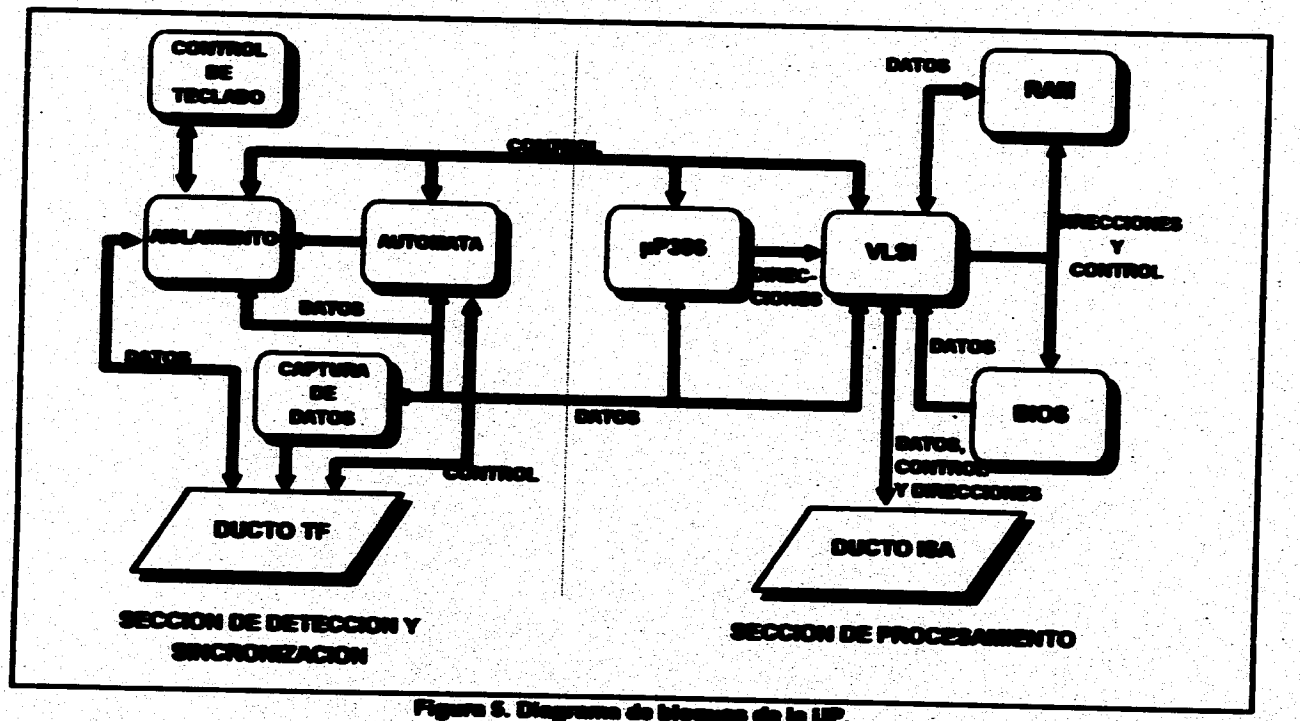


Figura 5. Diagrama de bloques de la UP.

3.3 Electrónica para la sincronización de las UPs

La operación concurrente de la arquitectura triplex de la CTF se basa en la posibilidad de frenar a los procesadores cuando se necesite hacerlo. En cuanto el sistema operativo ha sido instalado en cada procesador comienza la detección de fallas en tiempo real, es decir, se detectan fallas a la misma velocidad con que el microprocesador ejecuta una instrucción. A partir de ese momento un autómata electrónico detecta aquellas operaciones realizadas por el procesador que leen o escriben a memoria o hacia algún puerto. Una vez que se ha detectado de esta forma una operación, el dato generado es capturado en circuitos de registro para ser transferido a la UDFC cuando ésta lo solicite.

Estas operaciones se ejecutan concurrentemente en cada una de las UPs, lo que implica que desde la detección de operaciones hasta la captura de datos ocurran en tiempos muy semejantes en cada UP, pero no necesariamente iguales, debido a las diferencias de comportamiento eléctrico de los circuitos integrados. Esto aunado a que la UDFC requiere de cierto tiempo para realizar la lectura multiplexada de los datos capturados en cada UP redunda en la necesidad de detener a todas las UPs instaladas por tiempos muy cortos sin que éstas pierdan su estado de operación. En este tiempo la UDFC lee, guarda y compara los datos de cada UP. La manera de frenar a los procesadores se detalla en el capítulo cuatro.

3.4 Modos de funcionamiento de una UP

Cada UP puede funcionar de dos modos: uno en que funciona como uniprocesador sin redundancias, lo cual implica que cada UP cuenta con todas las características necesarias para trabajar como una computadora común. En este modo de funcionamiento se encuentran inhibidos todos los atributos de tolerancia a fallas. Una segunda forma de operación es cuando funciona como unidad procesadora para propósitos de tolerancia a fallas. La selección entre uno y otro modo de funcionamiento la realiza el usuario mediante un interruptor miniatura ubicado en el circuito impreso de cada UP.

En cada modo de funcionamiento existen algunas diferencias con respecto al uso del teclado principal y la bocina; en modo uniprocesador se utiliza la tarjeta de la unidad procesadora junto con periféricos adicionales para conformar la arquitectura de una PC industrial convencional en la que deben existir conectores directos para bocina y teclado mientras que para la arquitectura tolerante a fallas se requiere que un máximo de tres UPs puedan compartir el mismo teclado y la misma bocina.

Debido a que la comunicación del teclado con el controlador ubicado en cada una de las UPs es bidireccional, (es decir, una comunicación tipo serie, pero con retroalimentación) no permite que las tres UPs se conecten directamente al mismo teclado, ya que podrían provocarse colisiones durante el proceso de retroalimentación.

Para solucionar lo anterior, se permite que todas las señales provenientes del teclado sean leídas directamente por todas las UPs y se utiliza un interruptor analógico que establece la ruta de retroalimentación únicamente para la UP declarada como principal.

La solución para la bocina es similar a la del teclado por lo cual sólo la UP principal será la encargada de enviar los pulsos eléctricos hacia la bocina. El control de la salida para esta señal lo realiza el interruptor analógico.

En modo uniprocesador las conexiones son directas a través de los conectores.

3.6 Proceso de desarrollo de las UPs

Como se mencionó al principio de este capítulo, cada UP está formada por dos partes: una en la que se realiza el procesamiento y otra donde se ubica la electrónica de detección de operaciones y sincronización. En la figura 6 se muestran las dos secciones y el proceso de acoplamiento para integrar una sola tarjeta de procesamiento.

Estas dos secciones se desarrollaron por separado para finalmente, integrarse como una sola UP.

La sección de procesamiento se realizó mediante la adaptación de una tarjeta madre *(motherboard)* comercial, basada en el 386SX.

La determinación de adaptar una tarjeta comercial se tomó en base a la experiencia obtenida de una primera versión de UP desarrollada desde el diseño del circuito impreso de dos capas hasta el montaje de componentes de montado superficial para el caso del microprocesador (Intel 80386SX) y de su *"chip set"* (serie TACT83000 de Texas Instruments). El diseño de este prototipo de UP presentaba un problema en cuanto a la sustitución de algunos de estos componentes dañados, ya que la serie TACT83000 fue descontinuada, lo que provocó que sólo fuera posible conseguir dichos componentes mediante un pedido especial para un número mínimo de unidades muy alto, periodos de entrega prolongados y un costo fuera del presupuesto de este proyecto.

Por otro lado, el gran volumen de comercialización de las tarjetas comerciales reduce notablemente su costo, además de que sus reducidas dimensiones facilitan su adaptación e integración dentro del sistema.

La tarjeta comercial debía reunir ciertas características necesarias para integrarse dentro de la UP. Primeramente debía estar basada en un microprocesador 386SX, contener un controlador de teclado compatible con el 8042 de Intel, contar con bases para memoria RAM, que su *"chip set"* fuese

compatible con sistemas PC-AT, que incluyese ranuras de expansión del tipo ISA-AT y principalmente, que su circuito impreso fuese de dos capas, ya que un circuito multicapa habría hecho imposible la intercepción de ciertas señales necesarias para la sincronización y la detección de operaciones.

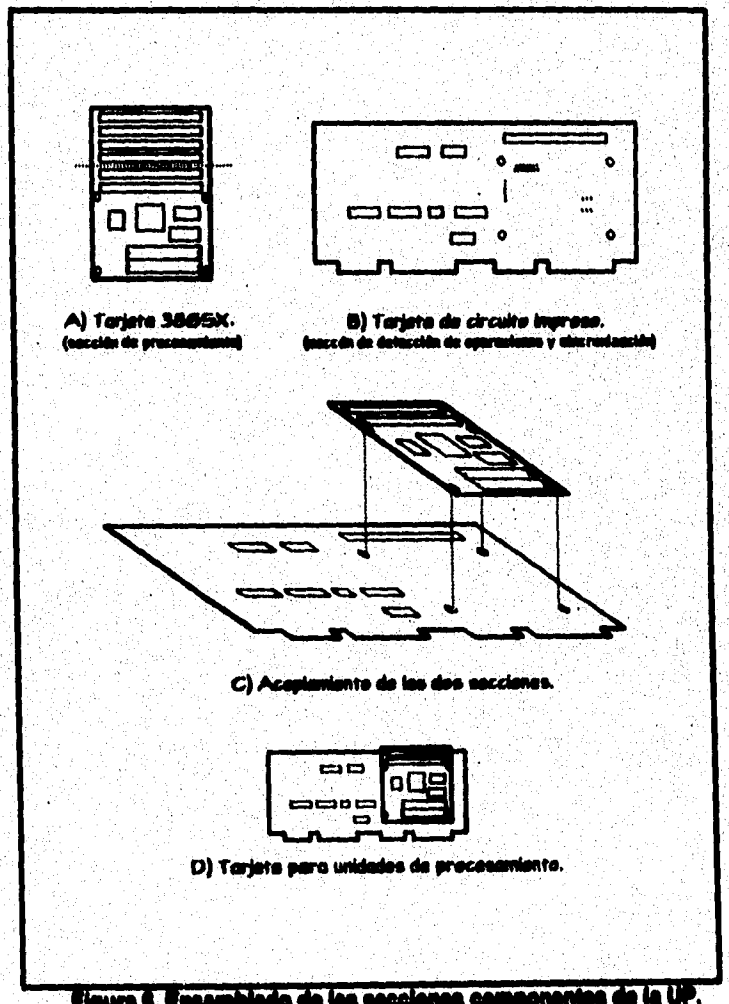


Figura 6. Ensamblado de las secciones componentes de la UP.

La tarjeta comercial utilizada es de amplio uso en computadoras personales y entre las modificaciones hechas estuvo la de reducir sus dimensiones, ya que contaba con seis ranuras de expansión de las cuales sólo dos eran fundamentales para poder adaptarse a la UP, esto se logró haciendo un corte que eliminaba cuatro ranuras de expansión que representaban aproximadamente una tercera parte de la tarjeta.

En cuanto a la sección de detección de fallas y sincronización se diseño una tarjeta de circuito impreso de dos capas donde se incluyó toda la electrónica necesaria para la detección de operaciones válidas para sincronizar a los tres procesadores, para la captura de datos y la electrónica necesaria para compartir el teclado y la bocina. Además, cada tarjeta cuenta con dos peines; uno es el ducto estándar PC-AT y el otro para el ducto propietario denominado tolerante a fallas (TF).

El diseño de estas tarjetas se realizó con la ayuda de un paquete de diseño de circuitos asistido por computadora, el TANGO PCB PLUS, cuyas características se detallan en el apéndice A.

La integración de ambas secciones se logró acoplando la tarjeta comercial (de dimensiones reducidas) sobre la tarjeta de circuito impreso diseñada, conectando las 98 líneas de señales del ducto PC-AT mediante un conector adaptado a una de las dos ranuras de expansión en la tarjeta comercial. (Ver figura 7).

Figura 7. Unidad de Procesamiento ensamblada

Las aproximadamente 30 líneas necesarias para las labores de sincronización y detección de operaciones se interceptaron con cables delgados y se llevaron a puntos de interconexión en la tarjeta diseñada (de

dimensiones mayores para contener a la tarjeta comercial). Entre estas líneas se encuentran los 16 bits de datos, señales de control del microprocesador y algunas señales del controlador de teclado. En la tabla siguiente se presentan todas estas señales:

LINEAS INTERCEPTADAS	
NOMBRE	**DESCRIPCION**
CLK2	*Clock 2*, pin 15 del 386, señal de reloj que oscila al doble de velocidad del reloj principal.
D/C#	*Data/Control*, pin 24 del 386, distingue entre ciclos de datos o ciclos de control.
ADS	*Address status*, pin 16 del 386, indica cuando un ciclo de bus es válido.
HLDA	*Bus hold acknowledge*, pin 3 del 386, indica que el microprocesador otorga control del bus a otro dispositivo.
RESET	*Reset*, suspende cualquier operación en progreso.
IOCHRDY	*I/O chanel ready*, pin A10 del bus AT, utilizada por periféricos para solicitar tiempo adicional.
D0-D15	Datos del microprocesador.
KBCLK	*Keyboard clock*, reloj del teclado.
KBDATA	*Keyboard data*, datos del teclado.
SPEAKER	Señal de tono en la bocina.
L2-6042(39), L1-6042(1), 7405(2), 7405(12).	Líneas de comunicación del teclado con el controlador de teclado (8042).

Tabla I. Señales que fueron interceptadas.

CAPÍTULO

CUATRO

ENSAMBLADO Y PRUEBAS REALIZADAS A LAS UNIDADES DE PROCESAMIENTO

4.1 Introducción

En este capítulo se detallan las pruebas realizadas a las unidades de procesamiento, éstas abarcan desde las aplicadas a la tarjeta de circuito impreso hasta las pruebas de operación realizadas a las dos secciones que conforman una UP. Asimismo se explican las secuencias de prueba e inicialización seguidas por el BIOS y algunas técnicas para evitar y solucionar problemas.

En lo que se refiere al autómata electrónico ubicado en la UP, se analiza su funcionamiento y se describen las pruebas efectuadas para lograr el frenado momentáneo de un procesador. Es debido precisar que el funcionamiento de la CTF está basado en el procesamiento concurrente y que para que éste sea eficaz es necesario tener la posibilidad de detener cualquier procesador para lograr la sincronización, de ahí la importancia de estas pruebas.

En el siguiente capítulo se explican las pruebas de interacción entre las UPs y otros módulos de la CTF.

4.2 Pruebas a la tarjeta de circuito impreso

Antes de que la tarjeta de circuito impreso se hubiera fabricado fue necesario realizar la revisión del diseño verificando exhaustivamente que todas las líneas, conexiones y componentes ubicados en el diseño de la tarjeta de circuito impreso *(pcb)* correspondieran correctamente con los marcados en el diagrama esquemático. Esta revisión se logra gracias a la ayuda de una herramienta del paquete de diseño TANGO PCB PLUS, la cual permite iluminar cada rama del circuito a la vez, facilitando el seguimiento de la ruta y conexiones que dicha señal tenga. Una vez que esta operación se ha realizado tantas veces como sea necesario, la tarjeta de circuito impreso puede ser enviada a fabricación.

Cuando se desarrolla un nuevo sistema es muy probable que éste no funcione inmediatamente como se espera, esto se debe a que existen algunos problemas básicos asociados con el diseño de tarjetas electrónicas para prototipos; Estos pueden ser: defectos de fabricación, líneas en cortocircuito, líneas sin conexión o errores en el montaje de componentes. En las pruebas aplicadas a las UPs fue posible evitar o reducir estos problemas mediante la aplicación de un método sistemático. Muchos de los pasos realizados pudieran parecer obvios o tediosos, pero son importantes cuando se trabaja con sistemas complejos. A continuación se explican estos pasos .

4.2.1 Inspección visual de la tarjeta

Antes de aplicar energía a la tarjeta fue necesario realizar una inspección visual, la cual consistió primero en verificar que cada parte y componente tuvieran la orientación correcta, es decir, que la terminal marcada con el número uno estuviere ubicada en su sitio.

El segundo paso fue inspeccionar que los componentes no hubieran sufrido ningún daño durante su inserción, cerciorando que cada terminal esté bien conectada y soldada. El uso de un microscopio óptico fue de gran ayuda para detectar conexiones débiles o soldadas en frío.

Cuando se trabaja con prototipos es importante el uso de bases para los circuitos integrados.

Finalmente, se verificó que los componentes fueran los correctos o, en el caso de remplazo, que fueran el equivalente adecuado.

4.2.2 Medición de continuidad

Antes de revisar la continuidad debe mencionarse que no es recomendable usar multímetros con señal sonora de continuidad, ya que estos aplican un pequeño voltaje que podría dañar a ciertos componentes. El primer paso fue verificar que no existiera un corto circuito directo entre voltaje de

alimentación (Vcc) y tierra (GND). Ésta medición con la tarjeta desnuda, sin componentes, indicó circuito abierto, pero en la tarjeta ya ensamblada con componentes no se obtuvo una lectura de resistencia del orden de megaohms, sino de unos cuantos cientos de ohms entre planos, debido a la resistencia interna de los componentes, puesto que se polarizan entre Vcc y GND. Por último, fue necesario verificar que cada terminal estuviera conectada en la ruta correcta y en las terminales de los componentes adecuados.

Figura 8. Pruebas a la tarjeta de circuito impreso

4.2.3 Verificación de los niveles del sistema

Es necesario revisar la magnitud y el nivel de ruido de la fuente de alimentación, si la señal de alimentación tiene más de 50mV de ruido los capacitores de desacoplo (entre Vcc y GND) pueden arreglar la situación colocándolos cerca de las entradas de alimentación, sobre la tarjeta. Finalmente, se requiere verificar que la frecuencia y la calidad de las señales de reloj sean las correctas.

4.2.4 Resultados de las pruebas a la tarjeta de circuito impreso

Después de haber realizado las pruebas anteriores se tuvo la seguridad de que el circuito impreso fabricado correspondía exactamente con el diseño esquemático original y, además toda posibilidad de cortocircuito o malfuncionamiento por defectos de fabricación o del montaje de componentes, fue eliminada.

Una vez superada esta primera etapa de pruebas básicas, el siguiente paso fue comprobar el correcto funcionamiento de las UPs, primero como uniprocesador y posteriormente, con sus características especiales para tolerancia a fallas.

4.3 Pruebas de encendido

Después de controlar los niveles esenciales para el sistema, lo siguiente fue aplicar alimentación monitoreando que no existe un aumento en el consumo de corriente y que los componentes no tuvieran un calentamiento excesivo.

Las pruebas de encendido pretendían determinar si cada UP era capaz de arrancar y cargar el sistema operativo como lo haría cualquier computadora PC comercial.

4.3.1 Secuencias de prueba e inicialización

Cada vez que un sistema (compatible con IBM-PC) se enciende o una señal de reinicio (RESET) ocurre, el BIOS ejecuta rutinas de prueba e inicialización a los circuitos en la tarjeta madre *(motherboard)* y en las tarjetas de periféricos, éstas rutinas se denominan POST por sus siglas en inglés *Power- On System Test*. El orden y la manera en que se realizan las pruebas varía con las diferentes versiones de BIOS, una secuencia usual es:

A) Probar algunos registros en el CPU.
B) Inicializar el 8253/8254 para el refrescamiento de RAM.
C) Inicializar el controlador de DMA para refrescamiento de memoria en el canal 0.
D) Verificar que el refrescamiento esté operando.
E) Probar la parte baja de la memoria RAM.
F) Cargar los vectores de interrupción en la parte baja de RAM.
G) Inicializar video y teclado.
H) Probar y medir la restante RAM.
I) Inicializar los puertos paralelo y serie.
J) Inicializar las unidades de disco duro y de disco flexible.
K) Llamar la rutina de servicio de arranque.

Las máquinas que emplean la arquitectura estándar industrial (ISA) emiten códigos POST a través del puerto 80, estos códigos muestran en hexadecimal, el procedimiento que está siendo aplicado por el BIOS. Los mismos códigos deberían cambiar con cada procedimiento de prueba o de inicialización que ocurra, de esta manera, un código que no cambia al siguiente tiende a un área general de error o falla.

4.3.2 Puesta en marcha de las UPs

En el caso de la CTF se utilizó una tarjeta decodificadora del puerto 80 para monitorear los códigos de error emitidos por el BIOS de la UP trabajando en modo uniprocesador (sin redundancias). El monitoreo de los códigos de error mostró en dos pantallas de siete segmentos, el código de la inicialización o de la prueba que el BIOS aplicaba en ese momento. Cada

secuencia de prueba o de inicialización dura escasos segundos y este código cambia al código de la siguiente prueba, solo en caso de que exista un mal funcionamiento, de la parte del sistema sometido a prueba, el código no cambia lo que nos refiere a un componente o a un grupo de componentes involucrados en dicho proceso.

La sustitución de componentes sirvió para determinar si un componente con defecto era el causante de un problema; Sin embargo cuando esto no era así, se encontró que muchos de los problemas detectados de esta manera, finalmente eran causados por las características eléctricas de las entradas o la carencia de conexión en las entradas no utilizadas (entradas flotantes).

Las entradas flotantes son entradas de dispositivos que no reciben estímulos de voltaje o corriente desde una salida, éstas pueden causar problemas funcionales o paramétricos dentro de un sistema, por ejemplo: alta corriente, aumento en el ruido o una inesperada llegada de señales de control que resulte en funcionamientos no deseados. De aquí que todas las entradas fueron revisadas y sólo las que contaban con resistencias de levantamiento *(pull up)* o de bajada *(pull down)* fueron dejadas como circuito abierto.

También, fueron verificadas las salidas que manejan entradas con posibles problemas, ya que salidas activas llevan la entrada de una carga hacia un voltaje de entrada bajo máximo (Vil máx) o hacia un voltaje de entrada alto mínimo (Vih mín). Las salidas de tres estados manejan las entradas de una carga en cualquier Vil máx o Vih mín o en un tercer estado de alta impedancia; además, las salidas de colector abierto sólo llevan la entrada de una carga a Vil máx cuando se activan y, mientras están desactivadas se encuentran en alta impedancia. (Ver figura 9).

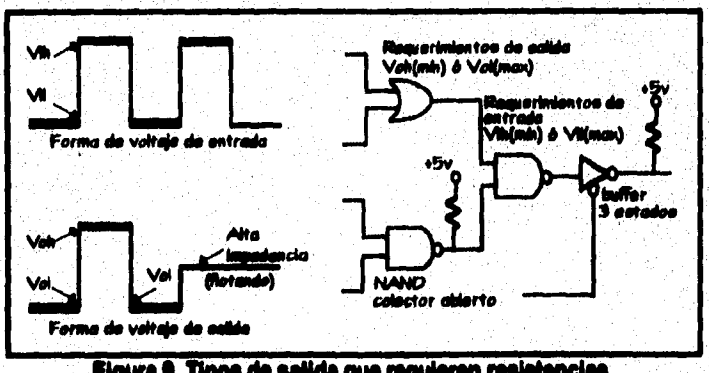


Figura 9. Tipos de salida que requieren resistencias.

Las salidas de tres estados y de colector abierto requieren de resistencias para manejar entradas correctamente, sin ellas la salida hace que

la entrada se encuentre "flotando" en el momento en que la salida se encuentra en alta impedancia. En la siguiente tabla se presentan las salidas en las que fue necesaria la inserción de resistencias:

Ducto PC-AT					
SEÑAL	SALIDA	VALOR	SEÑAL	SALIDA	VALOR
smemw#	3 estados	4.7 kohm	refresh#	c. abierto	330 ohm
smemr#	3 estados	4.7 kohm	memcs16#	c. abierto	330 ohm
iow#	3 estados	4.7 kohm	iocs16#	c. abierto	330 ohm
ior#	3 estados	4.7 kohm	iochrdy	c. abierto	1 kohm
sd0-sd7	3 estados	4.7 kohm	memr	3 estados	10 kohm
iochchk#	c. abierto	330 ohm	memw	3 estados	10 kohm
reset	3 estados	330 ohm	master#	c. abierto	330 ohm
ardy#	c. abierto	330 ohm			

Tabla II. Señales del ducto PC-AT con resistencias.

Ducto TF					
SEÑAL	SALIDA	VALOR	SEÑAL	SALIDA	VALOR
hldat	3 estados	10 kohm	fincmp	3 estados	10 kohm
iado	3 estados	10 kohm	cti	3 estados	10 kohms
e2tf	3 estados	10 kohm			

Tabla III. Señales del ducto TF con resistencias.

4.3.3 Resultados de las pruebas de encendido

Con las pruebas de encendido se encontró que cada UP podía funcionar como cualquier computadora compatible, cargando el sistema operativo, manejando un monitor, unidades de disco y un teclado. (Ver figura 10).

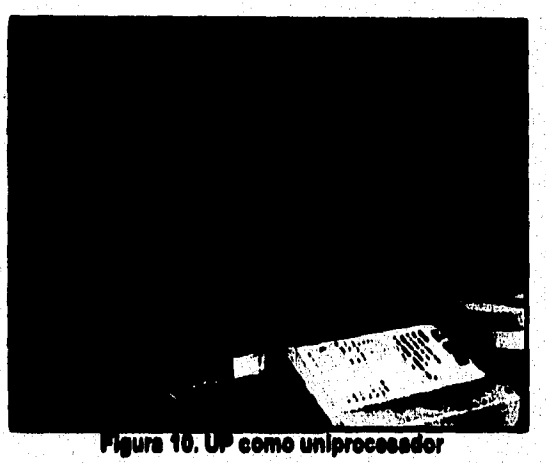


Figura 10. UP como uniprocesador

Sin embargo, hasta esta prueba la parte de cada UP dedicada a la operación concurrente había permanecido inhibida, por lo que el siguiente paso a seguir fue probar las características de la operación concurrente como son: la sincronización y la detección de operaciones.

4.4 Pruebas de sincronización

Como ya se explicó en el capítulo anterior, el principio en el cual se basa la computadora de procesamiento concurrente es la sincronización. En la CTF la sincronización se logra frenando los procesadores cuando es necesario.

El frenado en computadoras de arquitectura estándar industrial se puede lograr mediante el control de una de las señales del ducto AT, empleadas para expansión o interacción con periféricos.

La señal IOCHRDY *(I/O chanel ready)* permite a los recursos tales como memoria y dispositivos de entrada/salida (I/O), indicar al procesador que se requiere tiempo adicional para completar el ciclo actual. Cuando IOCHRDY no se desactiva, se ejecuta un ciclo estándar, es decir, sin estados de espera. Si IOCHRDY se desactiva, el microprocesador sostiene sus líneas de direcciones y control hasta que ha transcurrido el tiempo necesario para que el recurso esté en condiciones de recibir o de entregar un dato, al terminar este proceso se activa nuevamente la señal IOCHRDY para indicar al procesador que el periférico está en condiciones de continuar y que por lo tanto, el procesador es liberado. (Ver diagrama de tiempo en la figura 11). Cabe mencionar que la señal IOCHRDY no debe permanecer inactiva durante accesos o ciclos de transferencia mayores de 15.6μseg. De otra manera un ciclo de refrescamiento de memoria podría no ocurrir en el momento adecuado y los datos de la DRAM se perderían.

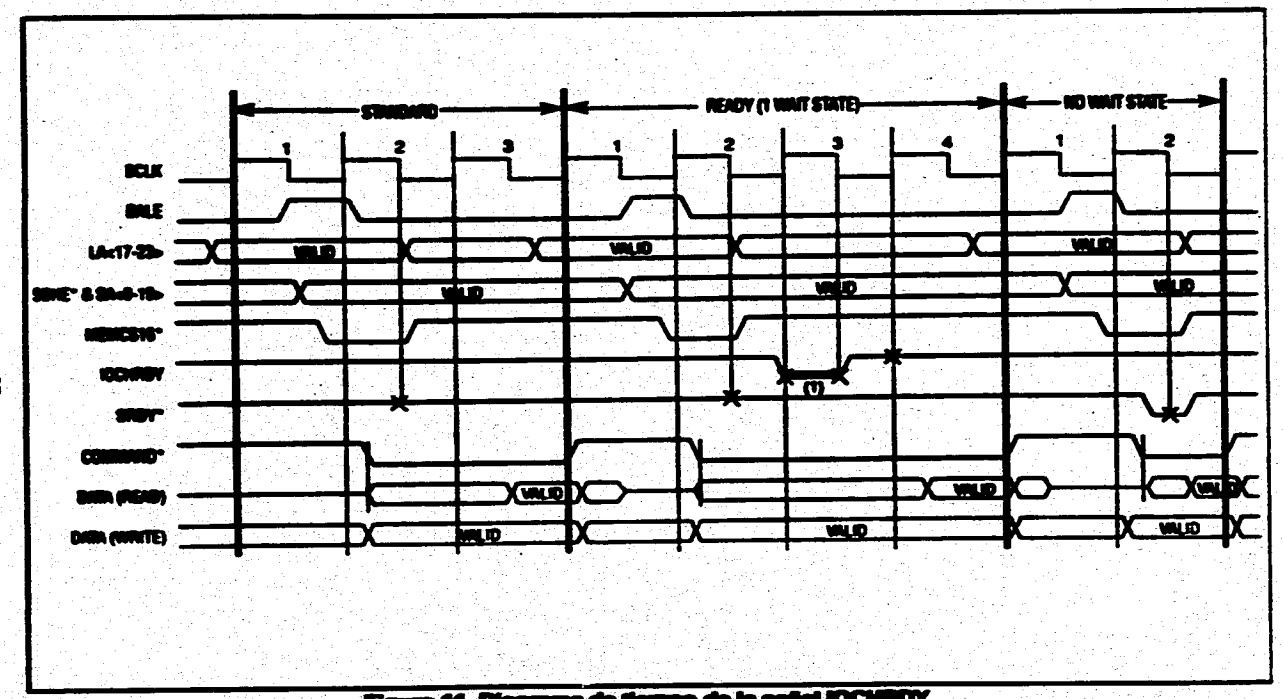


Figura 11. Diagrama de tiempo de la señal IOCHRDY.

La prueba más importante por lo que representa, consistía en frenar el procesador por un tiempo semejante al máximo tiempo requerido por la UDFC para leer y comparar los datos generados en cada UP, más un margen para garantizar que dicho proceso se realice. Durante este tiempo el procesador debe conservar su estado de operación.

Con este fin se diseñó un autómata electrónico de prueba que al detectar una operación válida frena el procesador. La detección de una operación válida se realiza con las señales ADS *(Adress status)* y D/C# *(Data/Control#)*, ambas señales del microprocesador. ADS indica un ciclo con direcciones válidas y D/C# indica si se manejan señales de datos o de control.

Una vez que una operación es detectada el autómata desactiva la señal IOCHRDY para detener el procesador durante aproximadamente 30 ciclos de reloj, ya que el tiempo requerido por la UDFC para realizar sus tareas no es mayor de 20 ciclos de reloj. Por último, se activa nuevamente la señal IOCHRDY liberando al microprocesador, el cual debe recuperar su operación normal. (Ver figura 12).

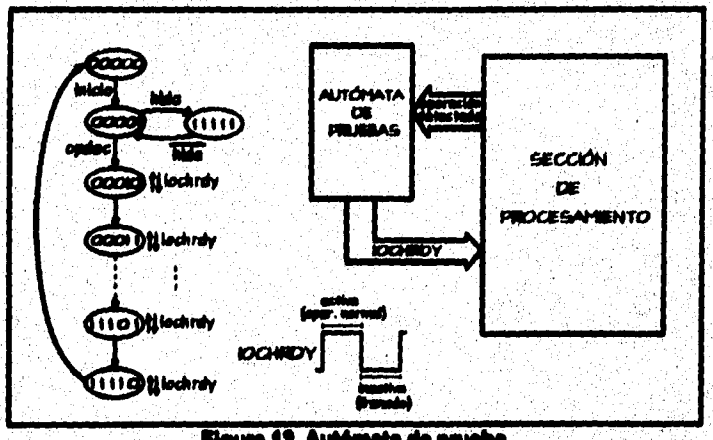


Figura 12. Autómata de prueba.

En el autómata se previó la detección de procesos DMA utilizados por el 386 para realizar accesos a disco, durante estos eventos se impedía que se frenara el procesador. Para detectar un proceso DMA se utilizó la señal HLDA *(hold acknowledge)*.

4.4.1 Resultados de las pruebas de sincronización

En la siguiente gráfica (obtenida por medio de un osciloscopio digital de 300MHz) aparecen dos señales correspondientes al ducto ISA-AT. La primera es BALE, *Bus Latch Enable*, que al ser válida, es decir cuando vale 5V (1 lógico) indica que en ese momento la líneas: SA *system address*, LA *latchable address*, AEN *address enable* y SBHE# *system byte high enable*, son válidas, por lo tanto las direcciones también lo son. La segunda señal es IOCHRDY, la cual es desactivada (puesta en 0V) por el autómata de prueba antes descrito.

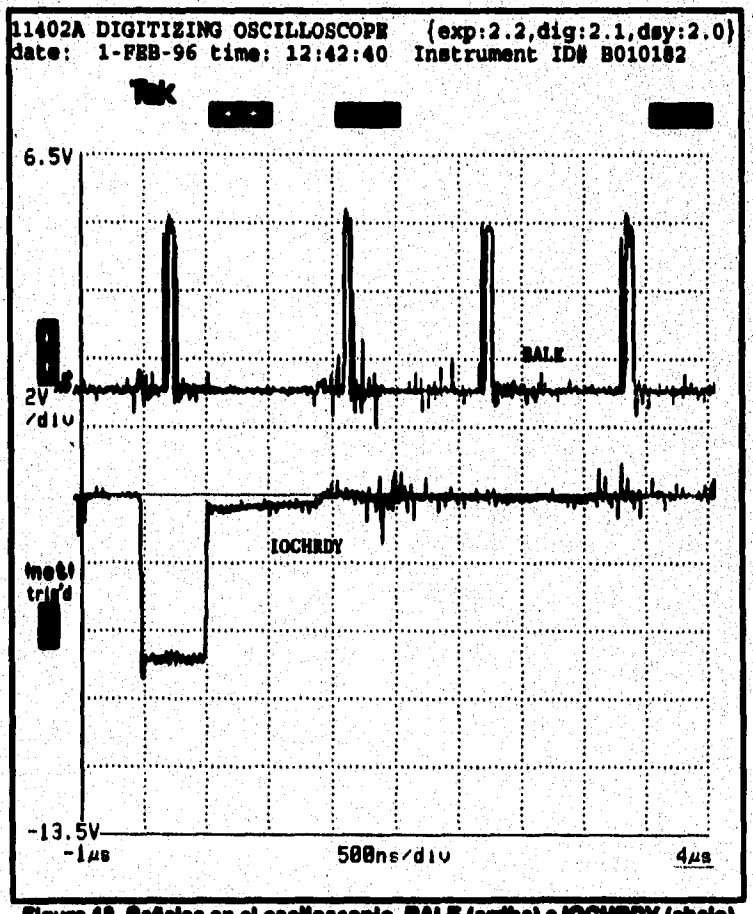


Figura 13. Señales en el osciloscopio, BALE (arriba) e IOCHRDY (abajo)

De la gráfica anterior se observa que el primer pulso de la señal BALE ocurre al mismo tiempo que la señal IOCHRDY se vuelve baja (0 lógico), lo que provoca que el siguiente pulso de BALE se presente 1500nseg después, es decir, 3 divisiones a 500nseg por división (aprox. 660kHz), esto debido a la inserción de estados de espera provocada por la desactivación de IOCHRDY; En los siguientes ciclos, cuando IOCHRDY no se desactiva, los pulsos de BALE ocurren periodicamente a 1000nseg (aprox. 1MHz), ya que IOCHRDY es activa y se ejecuta un ciclo estandar sin estados de espera.
Con esta prueba se pudo comprobar la posibilidad de frenar el procesador por tiempo suficiente para realizar la detección de fallas.

4.4.2 El autómata de detección de operaciones y sincronización

Como ya se ha mencionado cada UP contiene electrónica dedicada para detectar aquellas operaciones realizadas por el 386 que involucran accesos a memoria o a puertos. Para realizar estas tareas se desarrolló un autómata electrónico. (Ver figura 14).

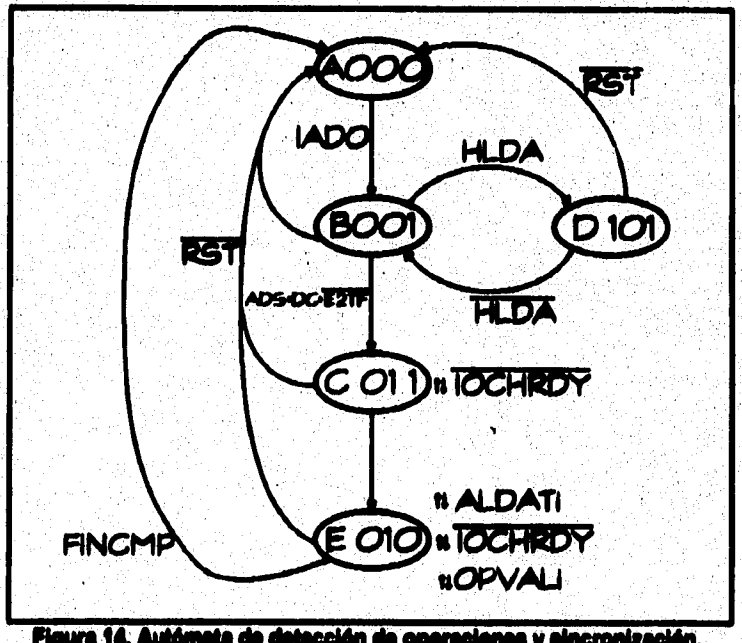


Figura 14. Autómata de detección de operaciones y sincronización.

Para iniciar la detección de operaciones, el autómata espera que se active la señal IADO (inicia autómata de detección de operaciones), la cual es generada por la UDFC principal. Posteriormente, el autómata pregunta por el inicio de una operación DMA, en cuyo caso se acude y se espera en el estado D hasta que dicho proceso haya terminado.

Adicionalmente, en el estado B, el autómata permite frenar a la UP respectiva siempre y cuando se decodifique una operación válida y que además, no existan dos tarjetas desactivadas en el sistema (E2TF). De cumplirse esta condición se desactiva la señal IOCHRDY provocando la generación de estados de espera en la UP respectiva. En el estado E se activa la señal OPVALI (operación válida) que avisa a la UDFC que se ha detectado una operación válida.

Además, se activa la señal ALDATi (activa *latch* de datos i) para retener el dato por comparar en un *latch* 74LS374, y en este estado se permanece hasta que se activa la señal FINCMP (fin de comparaciones), generada por los autómatas contenidos en la UDFC principal. La generación de la señal anterior indica que la UDFC ha recabado los datos generados en cada UP, los ha comparado entre sí y ha tomado una de las dos decisiones siguientes:

A) Generar la reconfiguración necesaria para resolver algún tipo de falla detectada para posteriormente, liberar al (los) procesador(es) restante(s).

B) Terminar el proceso de comparación al no detectar anomalías, y en este caso, liberar a los procesadores instalados.

Para lograr la reducción del número de componentes electrónicos utilizados por el autómata se recurrió a un circuito integrado tipo GAL *(Generic Array Logic)*, que es un dispositivo lógico de alta velocidad (12 nanosegundos de respuesta). (Ver figura 15).

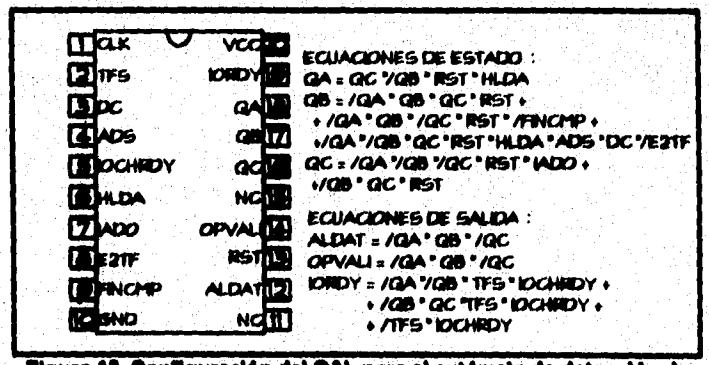


Figura 15. Configuración del GAL para el autómata de detección de operaciones y sincronización.

4.4.3 Pruebas de detección de operaciones

Para realizar esta prueba se sustituyó la EPROM BIOS por un EPROM que contenía un programa escrito en el lenguaje ensamblador del 386, la función de dicho programa era contar en hexadecimal desde 0000 hasta FFFF continuamente; De esta manera en el osciloscopio se observó la señal ALDATi (Activa Latch de Datos i) y el dato contenido en los circuitos integrados 74HC374. En la gráfica siguiente se muestran estas señales.

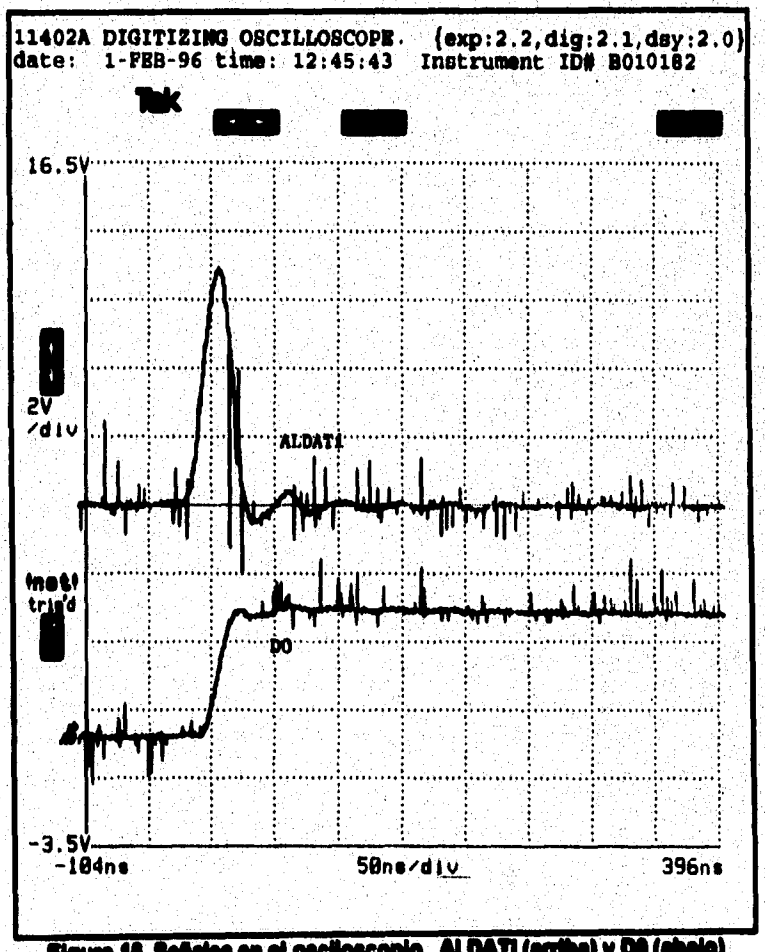


Figura 18. Señales en el osciloscopio, ALDATI (arriba) y D0 (abajo)

En la gráfica anterior se puede ver como es que se captura un dato válido; Cuando la señal ALDATi se activa (1 lógico), los datos existentes en el bus de datos del microprocesador son retenidos por los circuitos *latch* 74HC374, en este caso se muestra como el bit menos significativo D0, cambia de 0V a 5V en la salida del *latch* , en el momento en que la señal ALDATi es activada, esto se debe a que en el programa de conteo el bit menos significativo cambia constantemente de bajo (0V) a alto (5V).
Con esto se comprobó que el autómata de detección de operaciones es capaz de detectar los datos que corresponden a una operación válida.

CAPITULO

CINCO

PRUEBAS DE INTEGRACIÓN DE LA ARQUITECTURA COMPLETA

5.1 Introducción

Debido al diseño modular de la CTF es necesario verificar el correcto funcionamiento de las UPs al interactuar con otros módulos de la computadora, es por eso que en las siguientes secciones se explican las pruebas realizadas primero, entre cada unidad de procesamiento y su tarjeta de aislamiento (TA) asociada, después, las pruebas del funcionamiento de las UPs entre sí y, por último, las pruebas de interacción entre unidades de procesamiento y unidades de detección de fallas y control.

En el siguiente capítulo se dan detalles sobre la necesidad del uso de interrupciones para fines de mantenimiento correctivo y preventivo en la Computadora Tolerante a Fallas.

5.2 Pruebas entre una UP y su Tarjeta de Aislamiento (TA)

Las tarjetas de aislamiento constituyen un medio que hace posible la operación concurrente de las tres unidades de procesamiento conectadas a un mismo ducto y además, permiten desconectar parcial o totalmente una tarjeta objetivo (UP).

En las UPs la desconexión parcial impide la colisión en líneas de direcciones y datos en el ducto AT y, la desconexión total de las tarjetas objetivo se utiliza para aislar tarjetas que han sido identificadas por el sistema como anómalas.

Cuando la TA se encuentra vinculada a un procesador principal su sección de interfaz con el ducto AT operará de tal forma que todas las señales de salida generadas por el procesador principal se verán reflejadas en el ducto AT.

Si la TA está conectada a un procesador redundante, impedirá la llegada de direcciones y datos del procesador redundante al ducto AT; sin embargo, las líneas de entrada se habilitan para que los procesadores acepten la misma información. (Ver figura 17).

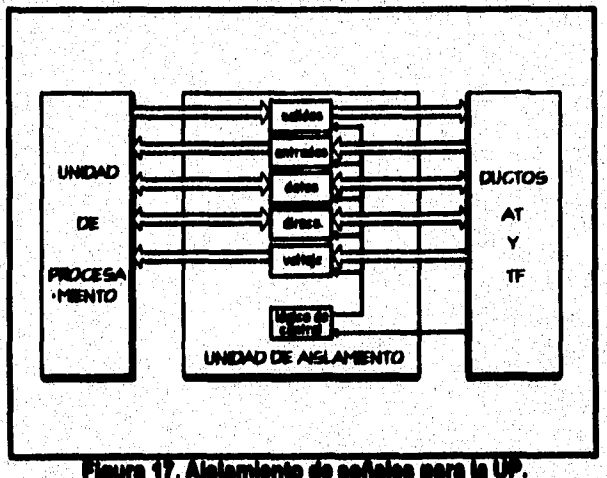


Figura 17. Aislamiento de señales para la UP.

El permitir o no el paso de señales digitales en cualquier sentido (del ducto a la UP o viceversa) se logra mediante el uso de interruptores

analógicos de tipo semiconductor (MM74HC4066), estos interruptores son bidireccionales y son controlados digitalmente.

El control para los diversos grupos de aislamiento como son: datos, direcciones, control, etcétera, se logró con lógica programada en circuitos tipo GAL. Las ecuaciones de lógica combinacional para el control se muestran a continuación:

/HENT = /ERCAT * ET
/HSAL = /ERCAT * ET * CT
/HDIR = /ERCAT * ET *CT
/HVCC = ET
/HDAT = /ERCAT * ET * /IOR + /ERCAT * ET * /CT * /IOW + /ERCAT * ET */MEMR * + /ERCAT * ET * /SMEMR + /ERCAT * ET * /CT /MEMW + /ERCAT * ET * /CT * /SMEMW

Donde:

HENT - Habilitador de entradas.	ERCAT - Error de categoría.
HSAL - Habilitador de salidas.	ET - Estado (enc./apag.).
HDIR - Habilitador de direcciones.	CT - Categoría (princ./redun).
HVCC- Habilitador de alimentación.	MEMR - Lectura de memoria.
HDAT - Habilitador de datos.	MEMW - Escritura a memoria.

Las pruebas realizadas fueron:

A) Programando los GALs con las ecuaciones de control se variaron las entradas y se verificó que las salidas fueran las esperadas.
B) Dentro de la tarjeta de aislamiento se comprobó que las salidas del GAL fueran capaces de controlar todos los interruptores por sección de aislamiento.
C) Se conectó cada UP a su respectiva tarjeta de aislamiento para que pudiera trabajar como procesador principal o como redundante.

De estas pruebas se determinó que la unidad de procesamiento funciona correctamente conectada a la tarjeta de aislamiento y que a su vez, ésta es capaz de aislar a la UP parcial o totalmente.

Figura 18. Unidad de Procesamiento y su tarjeta de aislamiento

5.3 Interacción entre unidades de procesamiento.

La interacción entre UPs sólo ocurre en dos formas: cuando sucede un acceso directo a memoria (DMA), y cuando se ha detectado una operación válide.

En el primer caso, cada UP al detectar un DMA activa su señal HLDAi de reconocimiento de DMA. Esta señal es llevada por el ducto TF hasta las tarjetas de aislamiento de las tres UPs, es decir, a cada tarjeta de aislamiento llegan las señales HLDA de cada UP (HLDA1, HLDA2 y HLDA3), donde la lógica electrónica genera HLDAT (hlda total) a partir de la ecuación:

$$HLDAT = HLDA1 * HLDA2 * HLDA3$$

De aquí se infiere que HLDAT se genera siempre y cuando se activen HLDA1, HLDA2 y HLDA3.

La señal HLDAT generada regresa a todas las UP para indicar al autómata que existe un ciclo DMA y evitar que dentro de este ciclo se frene al procesador. Las UPs continuan operaciones solo cuando los ciclos DMA de las tres tarjetas ha culminado.

En el segundo caso, cuando se ha detectado una operación válida, el autómata de cada UP genera la señal OPVALi, de igual manera, esta señal llega a cada TA junto con las OPVALi de las otras UPs (OPVAL1, OPVAL2 y OPVAL3), para generar mediante la lógica,OPVALT (operación válida total) con base en:

$$OPVALT = OPVAL1*OPVAL2 + OPVAL2*OPVAL3 + OPVAL1*OPVAL3$$

Donde OPVALT se puede generar tan sólo con la presencia de dos de las tres señales OPVALi de las UPs.

Por otra parte, el funcionamiento de ambas ecuaciones se corroboró variando las entradas y observando las salidas.

5.4 Pruebas de funcionamiento entre UPs y UDFC

Las pruebas que dan validez al funcionamiento de la CTF son las que verifican la correcta interacción entre las UPs y la UDFC. Estas pruebas consisten en generar errores intencionales en alguna de las UPs perfectamente identificada y comprobar que la UDFC detecte dicha UP con falla y la reporte oportunamente. En caso de que se trate de la UP principal, deberá originar el cambio de categoría en alguna de las UPs redundantes para convertirla en principal.

Los errores se originan alterando una de las líneas, ya sea de datos o de direcciones, manteniéndola en un nivel lógico fijo y provocando una lectura o escritura.

Debido al tiempo limitado para la elaboración de esta tesis y a que los trabajos con la UDFC no han concluido, se recurrió a simular la acción de reconfiguración ordenada por la UDFC al detectar la falla intencional.

Al detectarse una falla o al solicitarse una reconfiguración por parte del usuario, la UDFC envía una señal de interrupción a las UPs y éstas a su vez, responden leyendo un registro en la UDFC que indica la acción de reconfiguración que se deberá realizar.

La simulación de la UDFC consistió en proporcionar la señal de interrupción y el contenido de los registros a leer mediante el teclado. Con esto se pudo probar que en efecto el procesador respondía adecuadamente a la interrupción solicitada externamente, y ejecutaba la acción indicada en la rutina de atención a interrupciones.

En el capítulo siguiente, se detallan cuales son las acciones de reconfiguración y cuales son los algoritmos y protocolos para el manejo de interrupciones en la UP.

CAPITULO

SEIS

ALGORITMOS Y MANEJO DE INTERRUPCIONES

6.1 Introducción

En este capítulo se describe la necesidad de interacción entre UPs y UDFC para permitir actualizar datos y variables, se muestra cómo dicha interacción se logra basándose principalmente, en el uso de interrupciones y cómo es que dichas interrupciones son manejadas dentro del sistema de la CTF. Asimismo se explica de qué manera se conserva la sincronización de los procesadores cuando ocurre una petición de servicio a la interrupción por parte de la UDFC.

En el siguiente capítulo se exponen las conclusiones que emanan del presente trabajo, así como algunas recomendaciones que pueden ser de utilidad para el funcionamiento y mejora de la CTF.

6.2 Las interrupciones en sistemas PC-AT

El diseño de sistemas de microcomputadoras requiere que dispositivos de entrada y salida tales como; teclados, monitores, sensores y otros periféricos, reciban atención de una manera eficiente, de tal forma que las tareas totales del sistema puedan continuar sin efecto en la secuencia de operaciones (programa).

Con el uso de interrupciones el microprocesador al estar ejecutando su programa principal sólo se detiene para dar servicio a dispositivos periféricos, cuando ésto se requiera.

Las señales de petición de interrupción (IRQ0-15) son entradas externas, asíncronas, y se encuentran en el ducto PC-AT. Estas entradas informan al microprocesador que debe completar cualquier instrucción que esté siendo ejecutada en ese momento y direccionar una nueva rutina que atenderá el dispositivo que requiere atención.

Una vez que el servicio se complete, el procesador regresará exactamente al punto en que dejó el programa principal.

Como ya se dijo, el circuito VLSI de las UPs incluye un controlador de interrupciones compatible con el 8259 de Intel. En los sitemas PC-AT, los puertos de configuración del controlador de interrupciones son:

El puerto 20, usado para reconocimiento e inicialización y, el puerto 21, usado para habilitar o deshabilitar peticiones de interrupción especificas.

Cada dispositivo periférico, generalmente, tiene un programa especial o rutina que está asociada con su requerimiento específico, a esto se le denomina rutina de servicio.

6.3 Comunicación entre UPs y UDFC

El protocolo básico de comunicación entre UPs y UDFC consiste en el envio de una señal de interrupción desde el microcontrolador (en la UDFC), hacia los procesadores (en las UPs) instalados, para llevarlos a todos ellos a una misma rutina de interrupción. Esta rutina consiste en leer un registro de acción (en la UDFC principal), en tal registro la UDFC coloca previamente información que define el tipo de acción solicitada, así como a la(s) UP(s) a quien(es) va dirigida.

Las acciones posibles son las siguientes:

- Acción 1- Arranque sincrono de los tres procesadores.
- Acción 2- Indica recuperar la información guardada en disco duro para actualizarla en la memoria RAM.
- Acción 3- Indica respaldar toda la información contenida en RAM del pocesador principal en el disco duro.
- Acción 4- Conmutación de UDFC. Las UPs permanecen frenadas hasta que la nueva UDFC ordene el arrenque.

- Acción 5- Se indica a las UPs un cambio en estado (encendido o apagado) o en la categoría (principal o redundante) de cualquiera de las UPs instaladas, ya sea por haber ocurrido una falla o por cambios realizados por el usuario.
- Acción 10- Identificación de cada UP con un número para diferenciar acciones hacia UPs específicas.

Además, las UPs tienen acceso a la UDFC mediante cuatro puertos:

A) RDRE# -(Dirección 0301H). Lectura del registro de estado. Con este puerto las UPs actualizan su información acerca del estado actual de la arquitectura.

B) LECRA# -(Dirección 0300H). Lectura del registro de acción. En este puerto se indica el tipo de acción solicitada cuando la UDFC envía una interrupción.

C) LDRER# -(Dirección 0303H). Carga registro de estado redundante. Siempre que existe un proceso de reconfiguración de la arquitectura TF, la UP pasa la información del estado del sistema de la UDFC principal a la redundante que es un respaldo de la información del registro de estado

D) FINITA# -(Dirección 0302H). Fin de iniciación de tarjeta. En este puerto cada UP envía (una a la vez) una señal indicando la finalización del arranque. Durante la operación del sistema, este puerto es utilizado por cada UP para indicar que el procesador se encuentra detenido*(halt)*, es decir, que ha terminado su tarea encomendada. De este estado, cada procesador sólo puede salir por un reinicio (RESET) o por una petición de servicio de interrupción.

Es importante aclarar que cuando finaliza la rutina de servicio de interrupción de cada microprocesador es frenado mediante la instrucción*halt* para mantener la sincronización de los tres procesadores en los siguientes casos:

1) Cuando se requiere dar de alta a un nuevo procesador.
2) Después de conmutar de UDFC.
3) Luego de que las UPs realizan lecturas a los registros de estado o de acción.

De otro modo, el frenado producido por la desactivación de IOCHRDY, que sólo ocurre en caso de detectarse una operación válida, solamente es realizada por el autómata de detección de operaciones para conservar la sincronía de trabajo de las tres UPs.

A continuación se explican aspectos de la comunicación entre UDFC y UPs mostrando la inicialización y las rutinas de atención a interrupciones.

Cuando se enciende el sistema, el primer programa que se ejecuta es el autoejecutable (AUTOEXEC.BAT). Este archivo está localizado en el

directorio raíz del disco de inicio (generalmente la unidad C). Los comandos del archivo AUTOEXEC.BAT establecen las características de los dispositivos, personalizan información que MS-DOS presenta e inician los programas residentes en memoria y otras aplicaciones. Es aquí después de estos procedimientos convencionales, que se llama al programa INICIOCTF, para la inicialización de la Computadora Tolerante a Fallas.

Dentro del programa INICIOCTF se inicializa el vector de interrupción para apuntar a la rutina de atención a interrupciones propia, posteriormente, se lee el número de procesador asignado por la UDFC principal y se responde con una señal para indicarle a ésta que el procesador ha concluido su inicialización, finalmente el procesador ejecuta la instrucción *halt*. De este estado sólo puede salir con una señal de RESET o mediante una orden de arranque síncrono enviada por la UDFC (una interrupción).

Para sacar al procesador del estado de *halt* en que se encuentra, la UDFC principal envía una interrupción que obliga al procesador a saltar a una rutina de atención a interrupciones donde la UDFC principal, mediante un puerto, informa al procesador cuál de las cinco diferentes acciones es la que éste debe llevar a cabo, dichas acciones ya se han mencionado anteriormente en esta sección.

En la mayoría de estas acciones (excepto en la 3) el primer paso es llamar a la subrutina DETECT-UDFC, la cual evita que existan dos UDFCs declaradas como principales o como redundantes, lo que generaría, una señal de error.

En la acción 1, arranque síncrono, se habilita a los procesadores para que puedan arrancar síncronizadamente después de actualizar ciertos registros. Al final de cualquiera de las cuatro acciones restantes se coloca a los procesadores en estado de *halt*, ya que cualquiera de estas acciones que se ejecute, requiere de acciones adicionales o simplemente, requiere de la ejecución de la acción 1.

En lo referente a las demás acciones, la acción 3 resulta de gran importancia ya que en ella se almacena en disco duro toda la información contenida en RAM, que en este caso es de 1Mbyte, junto con algunos registros del microprocesador, los cuales son:

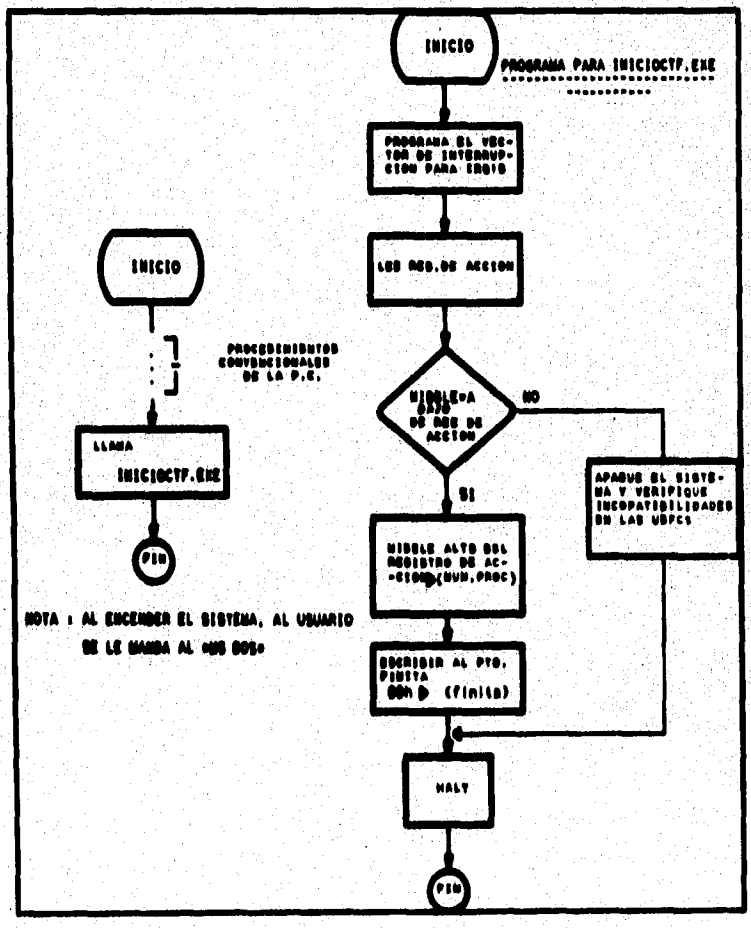
De propósito general (32 bits)		De segmento (16 bits)		Banderas (32bits)	Apuntador (32bits)
eax	ebp	gs	ss	eflags	eip
ebx	esp	fs	es		
ecx	esi	cs			
edx	edi	ds			

Tabla IV. Registros del 386SX almacenados.

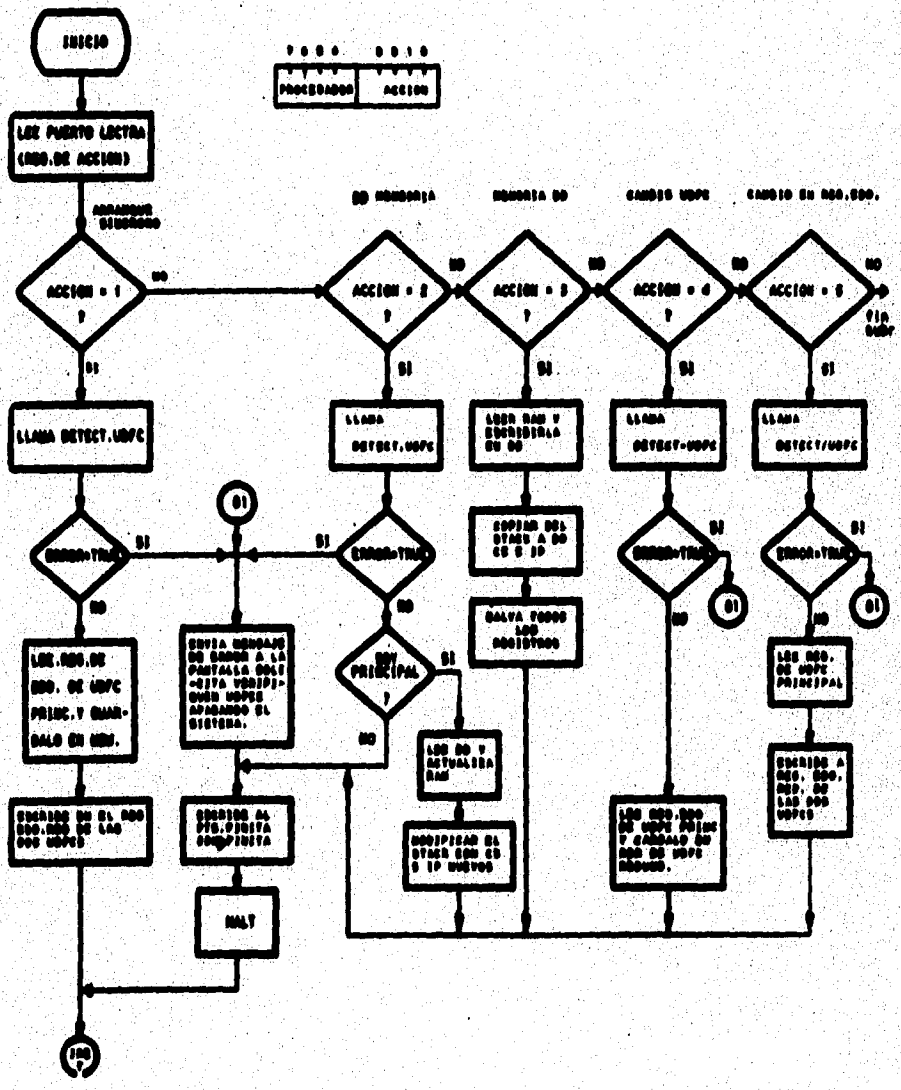
Esta acción es usada cuando una UP será dada de baja para conservar la situación actual.

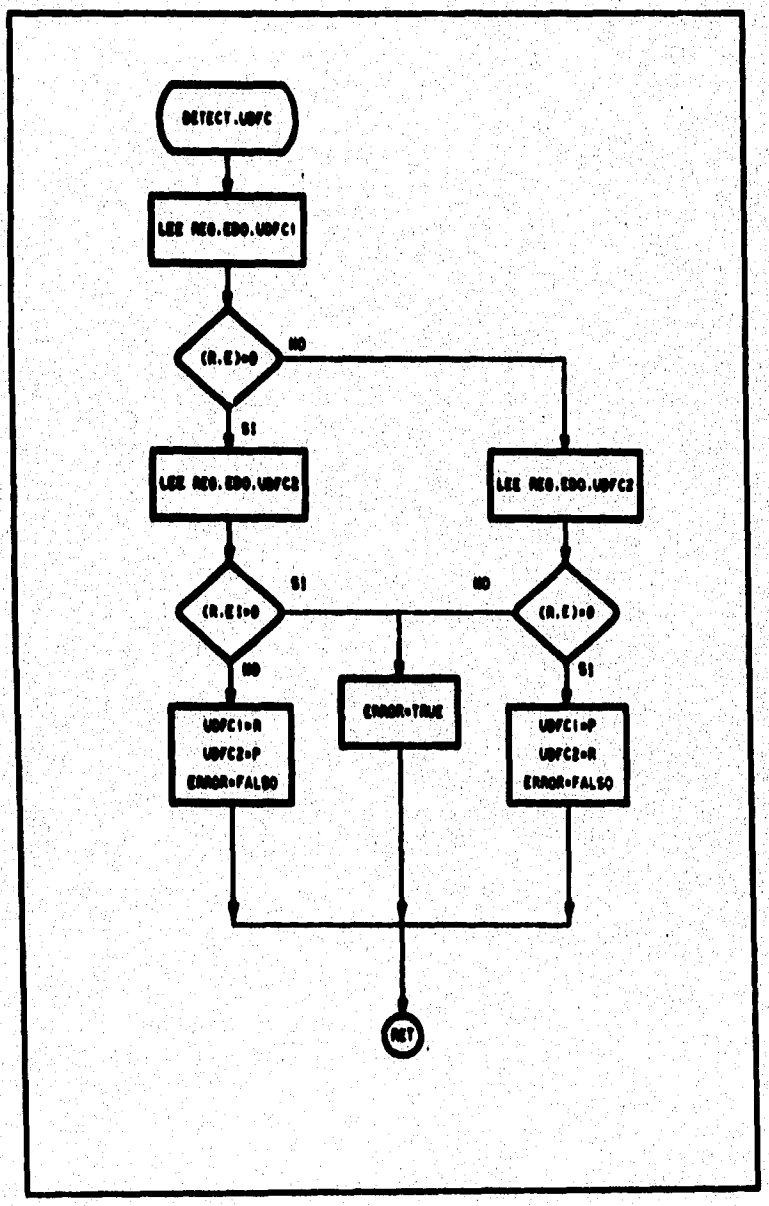
En la acción 2 ocurre el proceso inverso, es decir, recuperar de disco duro la información par actualizar la memoria RAM y los registros del microprocesador, esta acción es tomada cuando se ha dado de alta una nueva UP y se requiere restablecer las condiciones anteriores al cambio de UP.

Enseguida se muestran todos los procedimientos mediante diagramas de flujo. Además, en el apéndice C se presenta el listado de un programa en lenguaje C que realiza estos procedimientos, incluyendo una rutina en ensamblador del 386 para el manejo de la información entre disco duro y memoria.



RUTINA DE ATENCION A INTERRUPCION (INT15)(EN CADA UP)




DETECT.UDFC
LEE REG.EDO.UDFC1
(R.E)=0
NO
SI
LEE REG.EDO.UDFC2
LEE REG.EDO.UDFC2
(R.E)=0
SI
NO
(R.E)=0
NO
SI
UDFC1=R
UDFC2=P
ERROR=FALSO
ERROR=TRUE
UDFC1=P
UDFC2=R
ERROR=FALSO
RET

CAPÍTULO

SIETE

CONCLUSIONES Y RECOMENDACIONES

A lo largo del presente trabajo se han descrito las características de la arquitectura y del ensamblado de las unidades procesadoras de una Computadora Tolerante a Fallas. De igual manera, se han descrito las pruebas realizadas a cada Unidad de Procesamiento, UP, en las dos secciones que las componen. Así como las pruebas de interacción entre las distintas unidades que componen la CTF. También, se incluyó la forma en que son manejadas las interrupciones para labores de mantenimiento dentro del sistema.

Entonces, en este capítulo, se exponen las conclusiones obtenidas de las pruebas realizadas a las UPs por separado e integradas dentro del sistema. Además, se incluyen algunas recomendaciones.

7.1 Conclusiones

1) Se desarrolló una segunda versión de unidad de procesamiento, la cual cuenta con mayores posibilidades de mantenimiento y reparación a nivel componente que la primera versión, esto debido al bajo costo y a la forma de ensamblado de los componentes.

2) Se adaptó exitosamente, una tarjeta comercial *motherboard* 386 para poder funcionar dentro del esquema de tolerancia a fallas propuesto. Esta tarjeta representa la sección de procesamiento de la UP e incluye, al procesador 386SX, 1Mb de RAM, EPROM BIOS, *chip VLSI* y controlador de teclado.

3) Se diseñó el circuito impreso de la sección de detección de operaciones y sincronización que sirve de base al ensamblado de la UP. Este circuito se realizó en dos caras. Es menester subrayar que este circuito impreso fue fabricado en la industria nacional, que a pesar de sus limitaciones logró un resultado satisfactorio.

4) Cada UP es capaz de funcionar dentro de un esquema de procesamiento concurrente para fines de tolerancia a fallas. Además, cada UP, también, puede funcionar como un procesador independiente. En ambos casos, las UPs son totalmente compatibles con paquetes y periféricos comerciales.

5) La operación concurrente de los tres procesadores, basada en la sincronización, se logró mediante el frenado de cada procesador valiéndose para ello, de la desactivación de la señal IOCHRDY siempre que sea necesario. Como se mencionó en la sección 4.4 el periodo de prueba para la desactivación de IOCHRDY fue de aproximadamente 30 ciclos de reloj, que a una frecuencia de 40MHz corresponden a 0.7µseg, este tiempo es menor al tiempo máximo permitido de 15,6µseg pero suficientemente amplio para realizar las labores de detección de fallas

6) El autómata de detección de operaciones y sincronización es capaz de detectar accesos a puertos o a memoria y de capturar datos válidos.

7) La tarjeta de aislamiento asociada a cada UP, puede aislar parcialmente a una UP redundante o totalmente a una UP reportada con falla. Así como permitir el funcionamiento sin aislamiento de la UP principal.

8) Se recurrió al uso de interrupciones para actualizar cualquier procedimiento de reconfiguración ordenado por la UDFC principal sin que esto repercuta de manera importante en el flujo del programa de aplicación.

7.2 Recomendaciones

1) Para continuar y concluir la validación de la CTF es necesario la implementación de las pruebas entre UPs y UDFC como se detallan en la sección 5.4; Ya que como se dijo, estas fueron realizadas simulando la acción de la UDFC mediante datos adquiridos atravez del teclado.

2) En cuanto a la electrónica de detección de operaciones es importante realizar pruebas con componentes de mayor velocidad, con el fin de actualizar las unidades procesadoras para funcionar con sistemas 486 a 25 ó 33 Mhz.

3) En el caso de nuevos diseños de circuitos impresos multicapa para reducir las dimensiones de las tarjetas empleadas o simplemente, circuitos impresos para otras aplicaciones, es importante considerar y guardar una estrecha relación con la industria nacional, ya que esto traerá beneficios en ambos sentidos.

APÉNDICE A

TANGO PCB PLUS PARA EL DISEÑO DE CIRCUITOS IMPRESOS

Tango PCB PLUS es un paquete de software para diseñar y producir tarjetas de circuito impreso *(PCB- Print Circuit Board)*; con este paquete es posible dibujar y editar un diseño en la computadora permitiendo así, generar archivos de tipo *Gerber, Postcript y Epostcript*, los cuales son formatos estándar para impresoras láser que ofrecen salidas en película tipo positivo o negativo.

Este paquete permite el diseño de impresos constituidos de hasta 23 capas incluyendo las capas de componentes, de soldadura, ocho capas intermedias, planos de alimentación y tierras, máscaras de antisoldado para el lado de componentes y de soldadura, capa de referencias de componentes y de soldadura, capa de perforaciones, etc.

Asimismo, para definir trazos entre componentes permite editar líneas, arcos, círculos, polígonos para el trazo de planos, etc., todos ellos de dimensiones variables.

Además, faculta la escritura de textos utilizados como referencias particulares, el establecimiento de bloques para copiado y rotación. De forma

tal , que este paquete posibilita la construcción de impresos de hasta 32X32 pulgadas.

Uno de los detalles más importantes del paquete Tango es la generación de salidas denominadas *nido de ratas*, producidas con la información de los circuitos esquemáticos generados por el paquete Orcad. Este tipo de salida permite visualizar a todos los componentes que forman el diseño de manera conjunta con el total de las interconexiones declaradas. La importancia de esta opción radica en que el usuario puede modificar la ubicación de cualquiera de los componentes y en tiempo real (graficación animada) observar los cambios producidos en las interconexiones.

En cuanto al diseño de la UP, debe hacerse énfasis en que se escogió el arreglo que implicara el menor número de cruces entre las líneas de interconexión.

Otro punto importante es que el impreso de las UPs se elaboró en dos caras únicamente, lo que aumentó la complejidad del circuito impreso diseñado, ya que se contó con menos espacio para realizar el trazado de líneas. Se trató de dejar amplios espacios para planos de tierra, por lo demás, necesarios para disminuir problemas de ruido ocasionado por el tránsito de señales de alta frecuencia.

No obstante que el diseño involucra impresos de únicamente dos caras, para generar impresos de alta calidad es necesario generar al menos seis capas por circuito impreso, las que son:

- Capa de componentes, en donde se montan los circuitos integrados.
- Capa de soldadura, en la cual se sueldan los componentes.
- Capa de referencias, donde se incluyen textos y gráficos que aparecerán señalados en la tarjeta de circuito impreso.
- Máscara de antisoldado del lado de componentes que impide que la soldadura se propague en lugares indeseados e impide corto circuitos.
- Máscara de antisoldado del lado de soldadura.
- Capa de perforaciones, ésta indica los lugares en que el impreso deberá ser perforado y por donde se insertarán los componentes para su soldado.

A continuación se muestran algunas de las capas de la tarjeta de circuito impreso de la Unidad de Procesamiento diseñada.

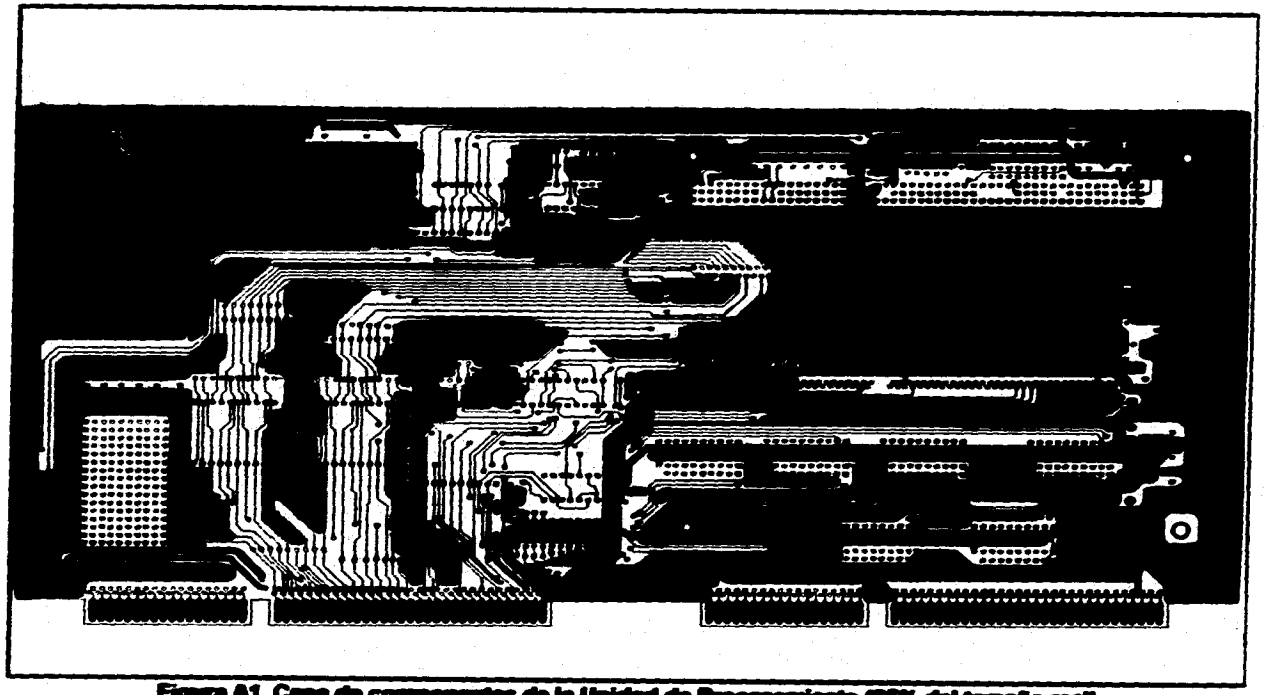


Figura A1. Capa de componentes de la Unidad de Procesamiento (65% del tamaño real)

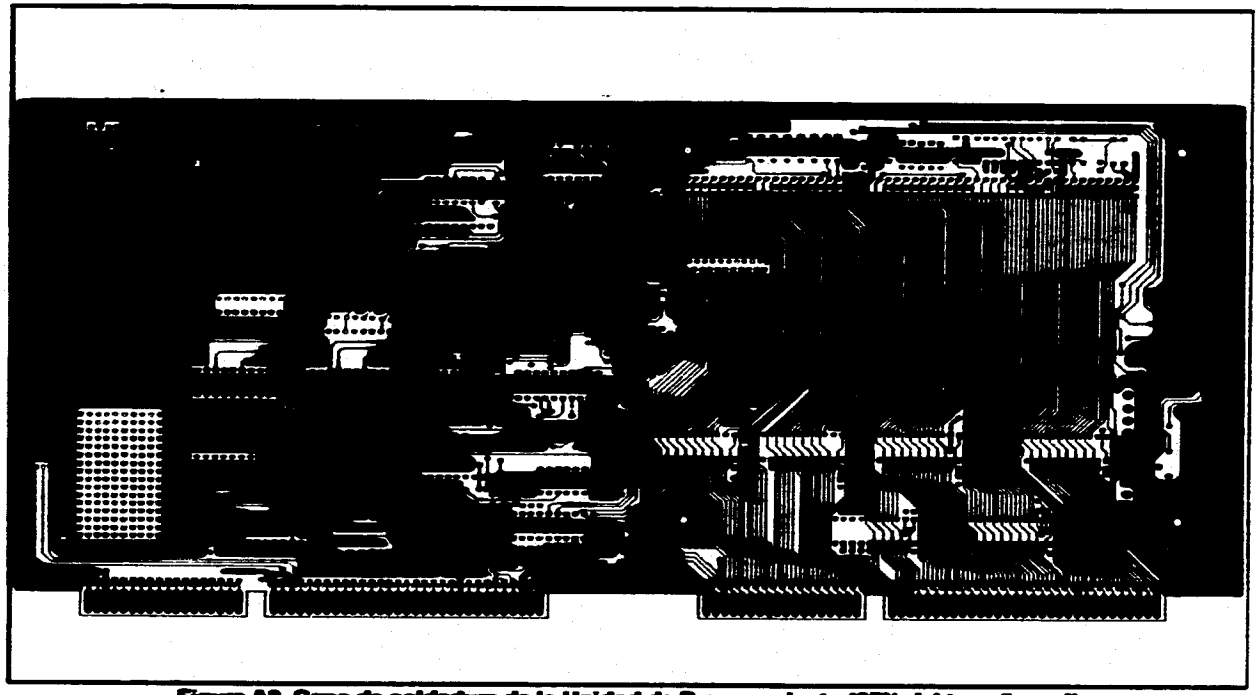


Figura A2. Capa de soldadura de la Unidad de Procesamiento (65% del tamaño real)

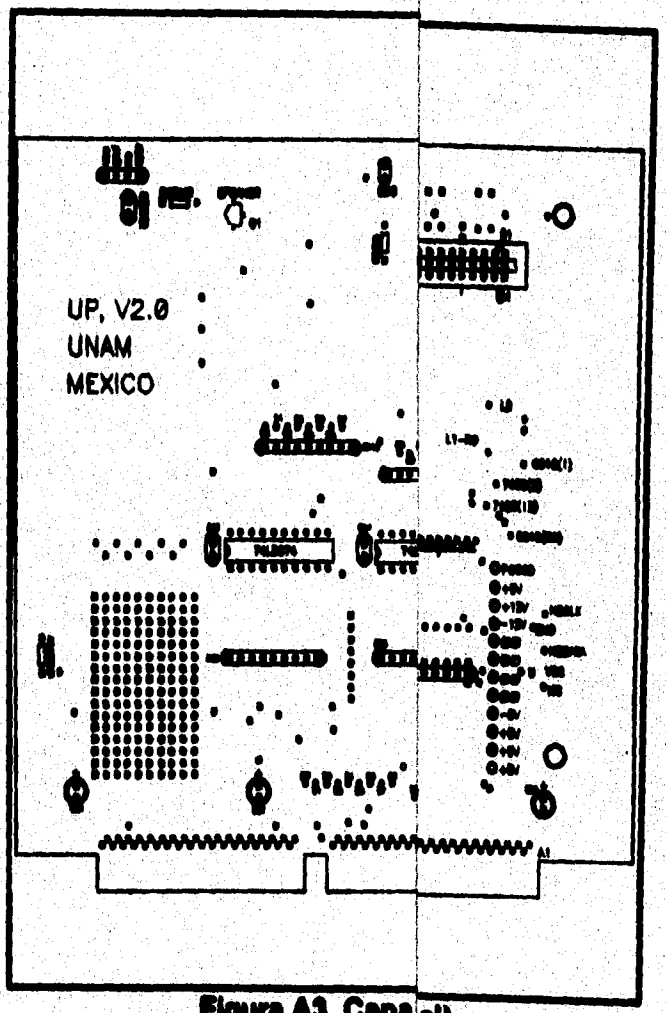


Figura A3. Capa a)

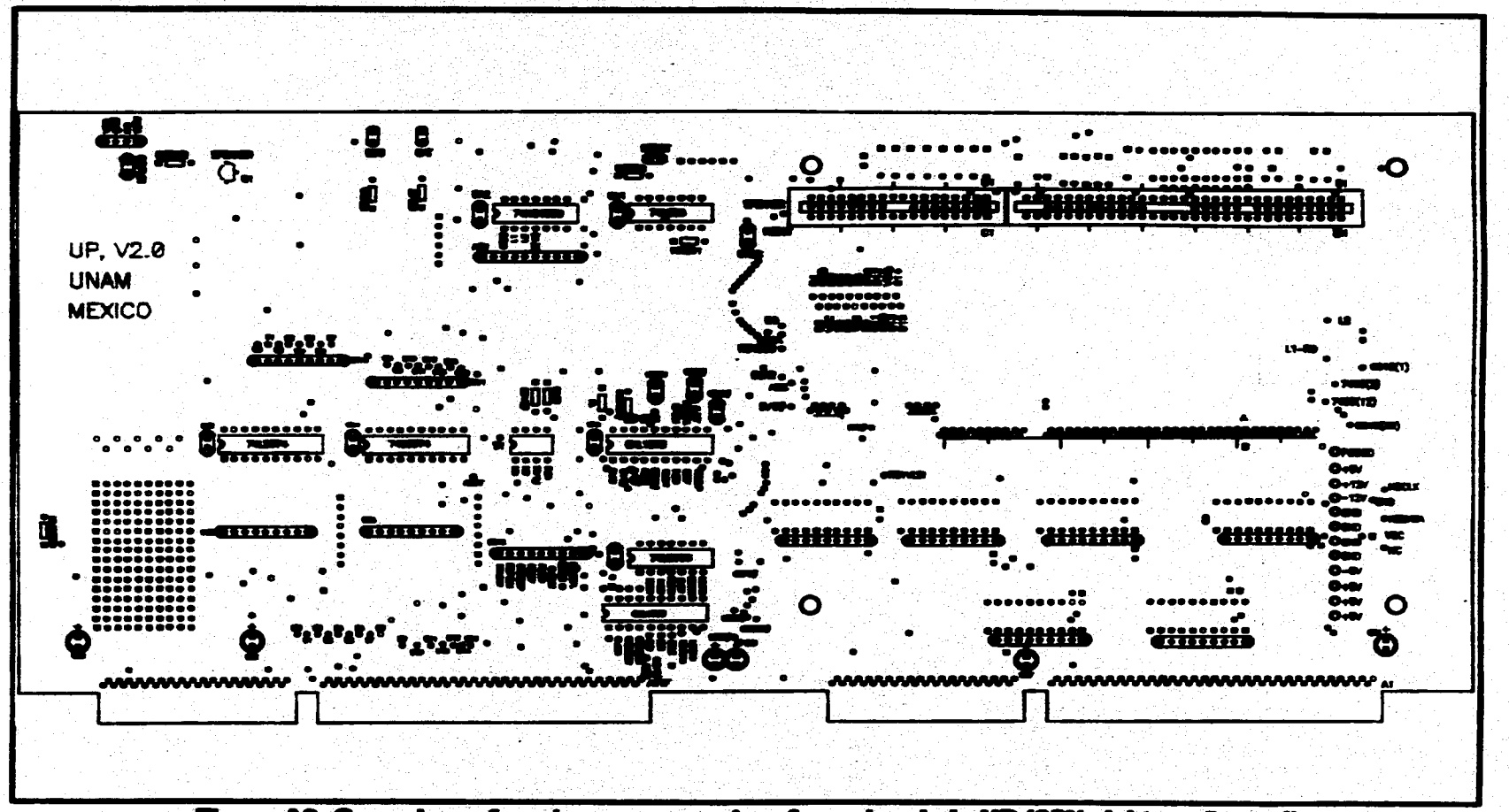
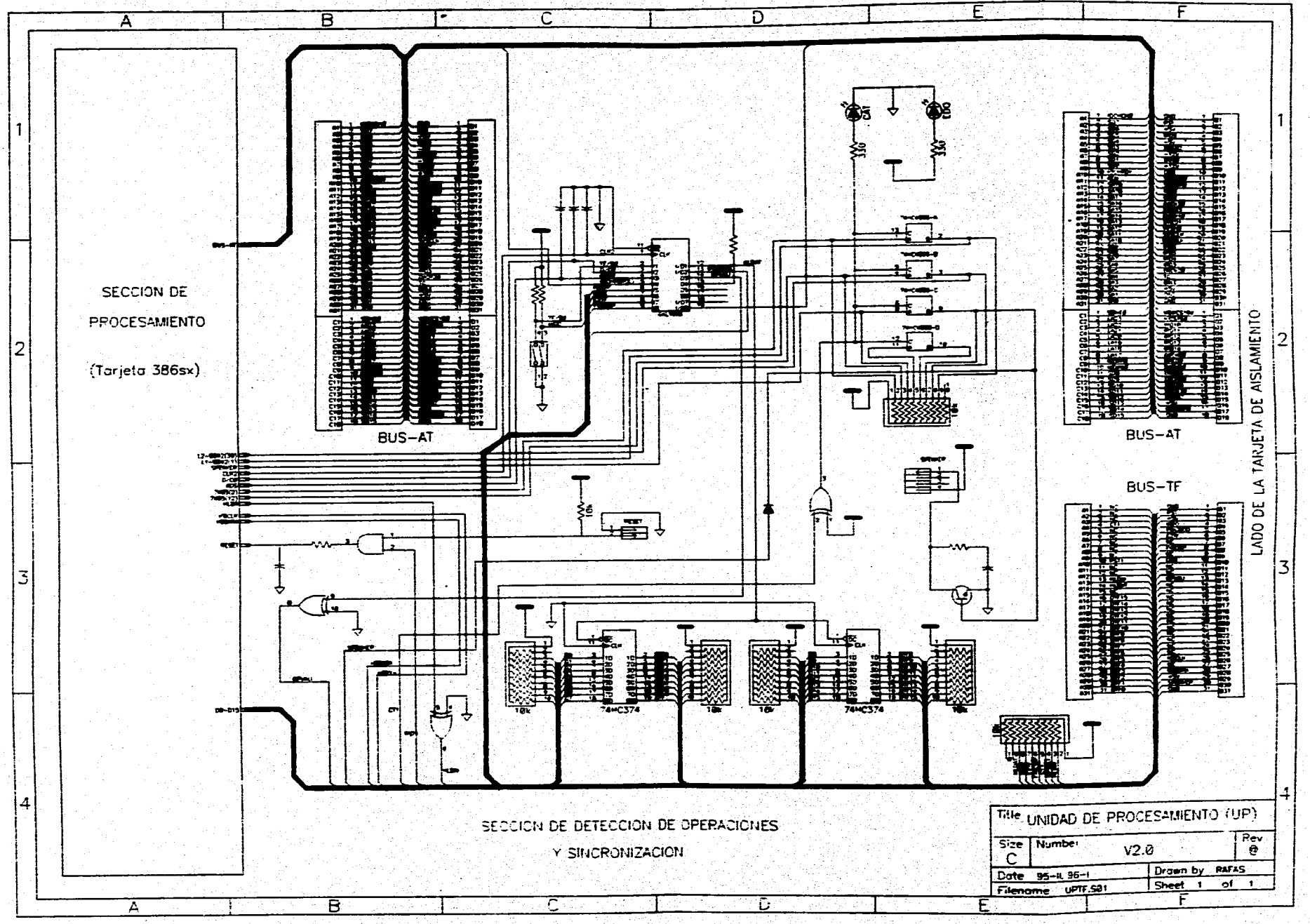


Figura A3. Capa de perforaciones y capa de referencias de la UP (65% del tamaño real)

APÉNDICE B

DIAGRAMA ESQUEMÁTICO DE LAS UNIDADES DE PROCESAMIENTO


SECCION DE
PROCESAMIENTO
(Tarjeta 386sx)
BUS-AT
BUS-AT
BUS-TF
74HC374
74HC374
LADO DE LA TARJETA DE AISLAMIENTO
SECCION DE DETECCION DE OPERACIONES
Y SINCRONIZACION
Title UNIDAD DE PROCESAMIENTO (UP)
Size C
Number V2.0
Rev @
Date 95-IL 96-I
Drawn by RAFAS
Filename UPTF.SCH
Sheet 1 of 1

APÉNDICE C

LISTADO DEL PROGRAMA DE INICIALIZACIÓN

```
/* Programa de inicializacion de la Computadora Tolerante a Fallas,
       y su rutina de servicio de interrupcion. Se simula la accion de
       la UDFC mediante el teclado. */

#include<stdio.h>                    /* Librerias */
#include<dos.h>
#include<conio.h>

void inicia(void);                                  /* Inicializa 8259 */
void interrupt get_out();                                   /* Prototipo de RSI nueva */
void interrupt (*oldfunc)();                        /* Puntero a RSI anterior*/

/* Funciones de accion de la interrupcion */
void armino(void), diram(void), ramdd(void), camudic(void), camrede(void);

void ctlvid();                     /*Subrutina en ensamblador para guardar RAM en disco duro o actualizar RAM con
disco duro*/

int categoria(void), detector(void);                          /* Otras funciones */
void alto1(void), alto2(void);

/* Variables */
unsigned int acc, nibaj, prec, reac, error, princ;

/* Programa principal */
main (void)
{
    inicia();
    printf("INTRODUCE EL CONTENIDO DEL REGISTRO DE ACCION\n");
    scanf("%x", &acc);
    nibaj = (acc & 0x0f);
    acc = acc>>4;
    if (nibaj == 0xa){
           prec = (acc & 0x07);
           puts("FINITA es 60");
    }
    else puts("ERROR - APAGUE EL SISTEMA Y VERIFIQUE UDFCs");
    asm hlt;                                          /*HALT en ensamblador 386*/
    puts("386 en HALT");
    setvect(0x16, oldfunc);
    puts("De regreso a MS-DOS");
    return 0;
}                                                       /* Fin del principal */
void inicia(void)
{
```

```
    oldfunc = getvect(0x14);                    /* Salva rutina anterior */
        setvect(0x14, get_out);                 /* Pone nueva rutina */
        outportb(0x21,0x20);                    /* Habilitando IRQ 3 */
}

/* Rutina de Servicio de Interrupcion, RSI */
void interrupt get_out()
{
    disable();
                printf("CUAL EL CONTENIDO DEL REGISTRO DE ACCION?\n");
    scanf("%x", &reac);
    reac = (reac & 0x0f);
    switch(reac){
            case 1 : arrsinc(); /* Acción 1 */
                        break;
            case 2 : dsram(); /* Acción 2 */
                        break;
            case 3 : ramdd(); /* Acción 3 */
                        break;
            case 4 : camudfc();/* Acción 4 */
                        break;
            case 5 : camredo();/* Acción 5 */
                        break;
            default:;}
                    outportb(0x20,0x63);        /* Borra IRQ 3 */
    enable();
}

void arrsinc()
{
    puts("EJECUTANDO ARRANQUE SINCRONO\n");
    int rest;
    detector();
    if (error) alto1();
    else{
        puts(" CONTENIDO DEL REGISTRO DE ESTADO\n");
        }
    return;
}

void dsram()
{
            puts("ACTUALIZANDO RAM CON INFORMACION DE DISCO DURO\n");
    detector();
    categoria();
        if (error) alto1();
        else if (princ){
        puts("actualizando RAM\n");
       puts("modificando stack con IP y CS");
       asm mov al, '2'
       cifutil();
            alto2();
            }
         else alto2();
}

void ramdd()
{
    puts("ALMACENANDO CONTENIDO DE RAM EN DISCO DURO\n");
     puts("salvando el stack y registros en disco duro");
     asm mov al,'1'
     cifutil();
    alto2();
}

void camudfc()
```

```
{
    puts("CAMBIO DE UDFC\n");
    detector();
    if (error) alto1();
    else {
            puts("Los registro de estado de la UDFC principal \n"); puts("y lo carga en la UDFC redundante\n");
            alto3();
            }
}

void camredo()
{
    int red;
    detector();
    if (error) alto1();
    else{
            puts(" CONTENIDO DEL REGISTRO DE ESTADO\n");

            alto3();
            }
}

detector()
{
    int udfc1, udfc2, redo1, redo2;
                puts("CUAL EL CONTENIDO DEL REGISTRO DE ESTADO DE UDFC1?\n");
    scanf("%x",&redo1);
    if (redo1 == 0){
        puts("AHORA DE UDFC2\n"); scanf("%x",&redo2);
        if (redo2 == 0) error =1;
        else{
                        puts("UDFC1 ES REDUNDANTE Y UDFC2 ES PRINCIPAL\n");
            error = 0;
            }
    }
    else{
            puts("CUAL EL CONTENIDO DEL REGISTRO DE ESTADO DE UDFC2?\n"); scanf("%x",&redo2);
            if (redo2 == 0) {
            puts("UDFC1 ES PRINCIPAL Y UDFC2 ES REDUNDANTE\n");
            error = 0;
            }
            else error = 1;
    }
return;
}

int categoria()
{
    int redo;
    puts("CUAL ES EL REGISTRO DE ESTADO?\n");
    scanf("%x",&redo);          princ = redo && 0x01;

    redo = redo>>(proc+2);
    return;
}

void alto1()
{
                        puts("ERROR - APAGUE EL SISTEMA Y VERIFIQUE UDFCs\n");
    puts("FINITA es 00\n");
    alto2();
}

void alto2()
{
    enable();
```

```
    puts("HALT");
    asm hlt;
    puts("386 en HALT");
}


void ctfutil()
{
asm {
/*
Programa para guardar el contenido de la memoria
RAM
con todos los registros del microprocesador
en disco.*/


dosseg
.model small
.stack 100h
.386

.data

cr          equ   13
lf          equ   10
menu        db '                ',cr,lf
                  db '                ',cr,lf
                  db '                ',cr,lf
                  db '                ',cr,lf
                  db '                ',cr,lf
                  db '                ',cr,lf
                  db '                ',cr,lf
                  db '                ',cr,lf
                  db '                MENU CTF
',cr,lf
                  db '                ',cr,lf
                  db '                ',cr,lf
                  db '                1) Guardar en disco
',cr,lf
                  db '                2) Cargar memoria ',cr,lf
                  db '                3) Regresar a DOS
',cr,lf
l_menu      equ $-menu
numero      dw 1

buffer      dd 1 dup(0),cr,lf,'$'
buffen      dw 4
manejador   dw ?
archivo     db 'C:\GUARDA.DOC',0
disco_lleno db 'Disco lleno. '
            db 'Cerrando Archivo . '
contador    dd ?
gax         dd ?
gbx         dd ?
gcx         dd ?
gdx         dd ?
gsp         dd ?
gbp         dd ?
gsi         dd ?
gdi         dd ?
gge         dd ?
gfs         dd ?
ggs         dd ?
gds         dd ?
ges         dd ?
ges         dd ?
gip         dw ?
gflag       dd ?
gcp         dd ?

doserror    db 'Error ocurrido en DOS: $'

.code

main proc

        call clrscr   /*LIMPIAMOS LA PANTALLA
        mov bx,de     IMPRIMIMOS EL MENU EN LA
                      PANTALLA*/
        mov ax,@data
        mov ds,ax
inicio:
        call clrscr
        call sigue1
        call clrscr
        mov bx,0001h
        lea dx,menu
        mov cx,l_menu
        mov ah,40h
        int 21h
        pusha

displ:
        lea si,numero /*PEDIMOS LA OPCION DEL
                      MENU*/
        mov bh,0
        mov bl,7
        mov ah,0

        cmp al,'1'
        je muestra
        cmp al,'2'
        je muestra1
        cmp al,'3'
        je salida
        jmp displ

muestra:      /* LLAMADO A RUTINA DE GUARDAR
              MEMORIA*/
        mov [si],al   /* Y REGISTROS EN DISCO
DURO*/
        mov ah,0eh
        int 10h
        popa
        call guarda
        jmp inicio

muestra1:   /* LLAMADO A RUTINA DE RECUPERAR
            MEMORIA*/
        mov [si],al   /* Y REGISTROS DEL DISCO
DURO*/
        mov ah,0eh
        int 10h
        popa
        call recupera
        jmp inicio
```

```
salida:
        mov dx,offset archivo  /* rutina para borrar
archivo*/
        mov ah,41h
        int 21h

        pops
        call cirscr
        mov ax,4C00h
        int 21h
        ret
main endp

sigue1 proc
        pop ax
        mov [gip],ax
        push ax
        ret
endp sigue1

guarda proc
        /*pop ax*/
        /*mov [gip],ax*/
        mov [gax],eax
        mov [gbx],ebx
        mov [gcx],ecx
        mov [gdx],edx
        mov [gsp],ebp
        mov [gbp],esp
        mov [gsi],edi
        mov [gdi],esi
        mov [gcs],cs
        mov [gds],ds
        mov [ges],es
        mov [gss],ss
        pushf
        pop eax
        mov [gflag],eax
        push ax
        popf
        mov ax,[gip]        /*Devolver a la pila la*/
                            /* direccion de regreso*/
        push ax         /* (IP) para este
procedimiento*/
        call cirscr
        call crear_archivo

        call espia_memoria
        call escribe_registros

        mov dx,offset archivo
        mov bx,manejador
        mov ah,3Eh
        int 21h

        ret
guarda endp

cirscr proc
        pusha
        mov ax,0600h
        mov cx,0
        mov dx,184Fh
        mov bh,7
        int 10h
        mov ah,2
        mov bh,0
        mov dx,0
        int 10h
        popa
        ret
cirscr endp

crear_archivo proc
        mov dx,offset archivo
        mov ah,3Ch
        mov cx,0
        int 21h
        jnc C1
C1:     mov manejador,ax
        ret
crear_archivo endp

escribe_registros proc

        mov cx,8
        mov eax,[gax]
        mov [buffer],eax
        mov dx,offset buffer
        call escribe_buffer

        mov eax,[gbx]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gcx]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gdx]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gbp]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gsp]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gsi]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gdi]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gip]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gcs]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gds]
```

```
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gss]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gss]
        mov [buffer],eax
        call escribe_buffer

        mov eax,[gflag]
        mov [buffer],eax
        call escribe_buffer

        mov eax,0
        mov ax,[gip]
        mov[buffer],eax
        call escribe_buffer
        ret
escribe_registros endp

copia_memoria proc
        mov ecx, 0
        mov [contador],0h
ciclo :
        mov eax,[contador]
        cmp eax, 0ffffh

        jg bloque2
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[2000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo

bloque2:
        mov ecx, 0
        mov [contador],0h

        mov bx,0001h
        mov dx,[0000:0298]
        mov cx,4
        mov ah,40h
        int 21h

ciclo1 :
        mov ecx,[contador]
        cmp eax, 0ffffh

        jg bloque3
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[1000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]

        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo1

bloque3:
        mov bx,0001h
        mov dx,[1000:0298]
        mov cx,4
        mov ah,40h
        int 21h


        mov ecx, 0
        mov [contador],0h
ciclo12 :
        mov ecx,[contador]
        cmp eax, 0ffffh

        jg bloque4
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[2000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo12

bloque4:
        mov bx,0001h
        mov dx,[2000:0298]
        mov cx,4
        mov ah,40h
        int 21h

        mov ecx, 0
        mov [contador],0h
ciclo13 :
        mov ecx,[contador]
        cmp eax, 0ffffh

        jg bloque5
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[3000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo13

bloque5:
        mov ecx, 0
        mov [contador],0h
ciclo14 :
```

```
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque8
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[4000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo14

bloque8:
        mov ecx, 0
        mov [contador],0h
ciclo15 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque7
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[5000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo15

bloque7:
        mov ecx, 0
        mov [contador],0h
ciclo16 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque6
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[6000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo16

bloque6:
        mov ecx, 0
        mov [contador],0h
ciclo17 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque9
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[7000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo17

bloque9:
        mov ecx, 0
        mov [contador],0h
ciclo18 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque10
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[8000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo18

bloque10:
        mov ecx, 0
        mov [contador],0h

ciclo19 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque11
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[9000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo19

bloque11:

        mov ecx, 0
        mov [contador],0h
```

```
ciclo110 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque12
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[0A000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo110

bloque12:
        mov ecx, 0
        mov [contador],0h
ciclo111 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque13
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[0B000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo111

bloque13:
        mov ecx, 0
        mov [contador],0h
ciclo112 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque14
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[0C000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo112

bloque14:
        mov ecx, 0
        mov [contador],0h
ciclo113 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque15
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[0D000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo113

bloque15:
        mov ecx, 0
        mov [contador],0h
ciclo114 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg bloque16
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[0E000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo114

bloque16:
        mov ecx, 0
        mov [contador],0h
ciclo115 :
        mov ecx,[contador]
        cmp ecx, 0ffffh

        jg regresa
        push ebx
        mov ebx,[contador]
        mov eax,0
        mov eax,[0F000h:ebx]
        pop ebx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        mov [buffer], eax
        call escribe_buffer
        jmp ciclo115


/*bloque17:

    mov ecx, 10000h
    mov [contador],10000h
```

```
ciclo116 :

    mov ecx,[contador]
    cmp ecx, 3000h

    jg regresa
    push ebx
    mov ebx,[contador]
    mov eax,0
    mov eax,[8000h:ebx]
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    mov [buffer], eax
    call escribe_buffer
    jmp ciclo116*/

regresa :


        ret

copia_memoria endp

escribe_buffer proc
        cli
        mov ah,40h
        mov bx,manejador
        mov cx,bufftm
        mov dx,offset buffer
        int 21h
E2:     ret
escribe_buffer endp

leer proc

        mov ah, 3fh
        mov bx, manejador
        mov cx, 4
        lea dx, buffer
        int 21h
        ret
leer endp

/* disco.asm*/

recupera proc

        call sigue
        mov es,[gss]
        mov ss,[gss]
        mov ds,[gds]
        mov esp,[gsp]
        mov fs,[gfs]
        mov gs,[ggs]
        mov edi,[gdi]
        mov esi,[gsi]
        mov ebp,[gbp]
        mov edx, [gdx]
        mov ecx, [gcx]
        mov ebx, [gbx]
        mov eax,[gax]
        push eax
        pop sp
        mov ax, [gip]
        push ax     /*
*********************************/
        mov eax,[gax]
recupera endp

sigue proc
  mov ah, 3dh               /* rutina para abrir el
archivo*/
  mov al, 0
  mov dx, offset archivo
  int 21h
  mov manejador, ax         /*rutina para abrir el
archivo*/

  mov ecx, 0          /*del disco duro*/
  mov [contador],0

ciclo2 :
  mov ecx,[contador]
  cmp ecx,0fffh

  jg bloque21
  mov ah, 3fh               /* y cargar en buffer*/
  mov bx, manejador
  mov cx, 4
  lea dx, buffer
  int 21h
  push ebx
  mov ebx,[contador]
  mov eax,[buffer]
  mov [8000h:ebx],eax
  pop ebx
  inc [contador]
  inc [contador]
  inc [contador]
  inc [contador]

  jmp ciclo2

bloque21:
        mov bx,0001h
        mov dx,[8000:0896]
        mov cx,0
        mov ah,40h
        int 21h

  mov ecx, 0          /*del disco duro*/
  mov [contador],0

ciclo21 :
  mov ecx,[contador]
  cmp ecx,0fffh

  jg bloque22
  mov ah, 3fh               /* y cargar en buffer*/
  mov bx, manejador
  mov cx, 4
  lea dx, buffer
  int 21h
  push ebx
  mov ebx,[contador]
  mov eax,[buffer]
  mov [1000h:ebx],eax
  pop ebx
  inc [contador]
  inc [contador]
```

```
        inc [contador]
        inc [contador]

        jmp ciclo21

    bloque22:
            mov bx,0001h
            mov dx,[1000:0200]
            mov cx,4
            mov ah,40h
            int 21h

        mov cx, 0          /*del disco duro*/
        mov [contador],0

    ciclo22 :
        mov cx,[contador]
        cmp cx,ffffh

        jg bloque23
        mov ah, 9h                /* y cargar en buffer*/
        mov bx, manejador
        mov cx, 4
        lea dx, buffer
        int 21h
        push cbx
        mov cbx,[contador]
        mov cax,[buffer]
        mov [2000h:cbx],cax
        pop cbx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        jmp ciclo22

    bloque23:
        mov cax, 0          /*del disco duro*/
        mov [contador],0

            mov bx,0001h
            mov dx,[2000:0200]
            mov cx,4
            mov ah,40h
            int 21h

    ciclo23 :
        mov cax,[contador]
        cmp cax,ffffh

        jg bloque24
        mov ah, 9h                /* y cargar en buffer*/
        mov bx, manejador
        mov cx, 4
        lea dx, buffer
        int 21h
        push cbx
        mov cbx,[contador]
        mov cax,[buffer]
        mov [2000h:cbx],cax
        pop cbx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        jmp ciclo23


    bloque24:

        mov cax, 0          /*del disco duro*/
        mov [contador],0

    ciclo24 :
        mov cax,[contador]
        cmp cax,ffffh

        jg bloque25
        mov ah, 9h                /* y cargar en buffer*/
        mov bx, manejador
        mov cx, 4
        lea dx, buffer
        int 21h
        push cbx
        mov cbx,[contador]
        mov cax,[buffer]
        mov [2000h:cbx],cax
        pop cbx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        jmp ciclo24

    bloque25:

        mov cax, 0          /*del disco duro*/
        mov [contador],0

    ciclo25 :
        mov cax,[contador]
        cmp cax,ffffh

        jg bloque26
        mov ah, 9h                /* y cargar en buffer*/
        mov bx, manejador
        mov cx, 4
        lea dx, buffer
        int 21h
        push cbx
        mov cbx,[contador]
        mov cax,[buffer]
        mov [2000h:cbx],cax
        pop cbx
        inc [contador]
        inc [contador]
        inc [contador]
        inc [contador]

        jmp ciclo25

    bloque26:

        mov cax, 0          /*del disco duro*/
        mov [contador],0

    ciclo26 :
        mov cax,[contador]
        cmp cax,ffffh
```

```
    jg bloque27
    mov ah, 3fh             /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [9000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo26

bloque27:

    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo27 :
    mov ecx,[contador]
    cmp ecx,0fffh

    jg bloque28
    mov ah, 3fh             /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [7000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo27


bloque28:

    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo28 :
    mov ecx,[contador]
    cmp ecx,0fffh

    jg bloque29
    mov ah, 3fh             /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [8000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo28

bloque29:

    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo29 :
    mov ecx,[contador]
    cmp ecx,0fffh

    jg bloque210
    mov ah, 3fh             /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [9000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo29

bloque210:

    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo210 :
    mov ecx,[contador]
    cmp ecx,0fffh

    jg bloque211
    mov ah, 3fh             /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [0A000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo210

bloque211:

    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo211 :
    mov ecx,[contador]
```

```
    cmp ecx,0ffffh

    jg bloque212
    mov ah, 3fh              /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [0B000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo211

bloque212:
    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo212 :
    mov ecx,[contador]
    cmp ecx,0ffffh

    jg bloque213
    mov ah, 3fh              /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [0C000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo212

bloque213:
    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo213 :
    mov ecx,[contador]
    cmp ecx,0ffffh

    jg bloque214
    mov ah, 3fh              /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [0D000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo213

bloque214:
    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo214 :
    mov ecx,[contador]
    cmp ecx,0ffffh

    jg bloque215
    mov ah, 3fh              /* y cargar en buffer*/
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [0E000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo214

bloque215:
    mov ecx, 0          /*del disco duro*/
    mov [contador],0

ciclo215 :
    mov ecx,[contador]
    cmp ecx,0ffffh

    jg regresa2
    mov ah, 3fh
    mov bx, manejador
    mov cx, 4
    lea dx, buffer
    int 21h
    push ebx
    mov ebx,[contador]
    mov eax,[buffer]
    mov [0F000h:ebx],eax
    pop ebx
    inc [contador]
    inc [contador]
    inc [contador]
    inc [contador]

    jmp ciclo215

/*bloque216:

    mov ecx, 10000h
    mov [contador],10000h

ciclo216 :

    mov ecx,[contador]
    cmp ecx,30ffffh
```

```
  jg regresa2
  mov ah, 3fh              ; y cargar en buffer
  mov bx, manejador
  mov cx, 4
  lea dx, buffer
  int 21h
  push ebx
  mov ebx,[contador]
  mov eax,[buffer]
  mov [00000h:ebx],eax
  pop ebx
  inc [contador]
  inc [contador]
  inc [contador]
  inc [contador]

  jmp ciclo216*/

regresa2 :

  mov ah, 3fh
  mov bx, manejador
  mov cx, 4
  lea dx, buffer
  int 21h

  mov eax,[buffer]
  mov [gax],eax

  call leer
  mov eax,[buffer]
  mov [gbx], eax

        call leer
        mov eax,[buffer]
        mov [gcx], eax

        call leer
        mov eax,[buffer]
        mov [gdx], eax

        call leer
        mov eax,[buffer]
        mov [gbp],eax

        call leer
        mov eax,[buffer]
        mov [gsp],eax

        call leer
        mov eax,[buffer]
        mov [gsi],eax

        call leer
        mov eax,[buffer]
        mov [gdi],eax

        call leer
        mov eax,[buffer]
        mov [gps],eax

        call leer
        mov eax,[buffer]
        mov [gfs],eax

        call leer .
        mov eax,[buffer]
        mov [gca],eax

        call leer
        mov eax,[buffer]
        mov [gds],eax

        call leer
        mov eax,[buffer]
        mov [gse],eax

        call leer
        mov eax,[buffer]
        mov [gss],eax

        call leer
        mov eax,[buffer]
        mov [gfss],eax

        call leer
        mov eax,[buffer]
        mov [gip],ax

        mov dx,offset archivo   /*rutina para cerrar el
archivo*/
        mov bx,manejador
        mov ah,3Eh
        int 21h

  ret
sigue endp

end main
  }
return 0;
}
```

BIBLIOGRAFÍA

CHOISSER, J.P.; Foster, J.O.; The XT-AT handbook, Annabooks, 1992.

FOX, G.; Johnson, M.; Lyzenga, G.; Otto, S.; Solving problems on concurrent processors, Prentice Hall.

HALL, D.V., Microprocessors and interfacing, Glencoe-Mc Graw Hill, 1992.

INTEL PERIPHERALS, Intel USA, 1990.

MEJÍA, J.A., Tesis de licenciatura, Facultad de Ingeniería UNAM, Diseño y desarrollo de las tarjetas objetivo de una Computadora Tolerante a Fallas, 1994.

POST PROBE, Technical Manual, Micro 2000 Inc., 1992.

RIVERA, M. R:, Tesis de licenciatura, Facultad de Ingeniería UNAM, Desarrollo de una tarjeta electrónica para controlar dinámicamente el flujo de información en los ductos de una computadora automaconfigurable, 1993.

SOLARI, E., AT bus design, Annabooks, 1991.

TEWSBURY, Dickinson, Schwartz, Concurrent Computations, Plenum Press, USA.

VICENTE, E., Computadoras tolerantes a fallas: evolución, análisis y oportunidades, Memoria del XIX Congreso de la ANIAC, septiembre de 1986.

VICENTE, E.; Mejía, J. A.; Rivera, M: R:; Diseño de una computadora de procesamiento concurrente para aplicaciones de alto riesgo, Memoria del II Congreso de Investigación en Ciencias Computacionales, octubre 1993.

ZIEGLER, J.; Hornback, T.; Jordan, A.; The ten commandments of debugging, Electronic Design, septiembre 3 de 1992.