UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO DE UNA ARQUITECTURA DE PROCESAMIENTO DIGITAL DE IMAGENES

T E S I S

QUE PARA OBTENER EL TITULO DE INGENIERO MECANICO ELECRICISTA

P R E S E N T A N :

LARRY HIPOLITO ESCOBAR SALGUERO
RANULFO RODRIGUEZ SOBREYRA

DIRECTOR: M. en C. JORGE MARTINEZ FLORES

MEXICO D. F. 1992

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

I. INTRODUCCION

En 1990, el Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) llevó a cabo la construcción de un dispositivo electrónico basado en una computadora personal para la adquisición y despliegue de imágenes de video. Entre las características de este dispositivo están la adquisición de una imagen, su digitalización y almacenamiento en disco como un archivo tipo byte. A partir de este desarrollo, se planteó la realización de una arquitectura electrónica que llevara a cabo el procesamiento de las imágenes adquiridas, las cuales ya procesadas también pudieran ser desplegadas.

Este trabajo presenta el planteamiento de una arquitectura para procesamiento de imágenes utilizando en una primera fase una computadora personal (PC) para controlar el sistema. La tarjeta principal de procesamiento tiene como unidad de procesamiento el Procesador de Señales Digitales (DSP) TMS320C25 (TMS), utilizado por sus relevantes características técnicas.

Para efectos de estudio y desarrollo, el trabajo se divide en:

Capítulo I. Hace una breve introducción de la Arquitectura de Procesamiento Digital de Imágenes, tema de este trabajo. Se plantean los objetivos a cubrir y la justificación necesaria para llevar a cabo su desarrollo. También se hace una descripción general del sistema, la cual engloba las diferentes partes y etapas a seguir, así como diagramas a bloques y esquemas del mismo.

Capítulo II. Estudia los Sistemas de Procesamiento Digital de Imágenes con un enfoque evolutivo. Se enuncian los conceptos del Procesamiento Digital de Imágenes, las etapas que lo conforman, el tipo de lenguaje y terminología utilizada, hasta llegar a los tipos, métodos y algoritmos de procesamiento. Este tema da el marco teórico para plantear la arquitectura a desarrollar.

Capítulo III. Hace referencia al procesamiento en paralelo, debido a que favorece el diseño de sistemas para lograr el procesamiento en tiempo real. Se describe la clasificación de estos sistemas, su nomenclatura y su proyección al tiempo real.

Capítulo IV. Presenta la implementación en hardware de la arquitectura, partiendo del módulo de interfase que comunica a la computadora personal con el módulo de procesamiento a través del módulo de conexión. Seguidamente se describe el módulo de procesamiento, parte central de la arquitectura, y el procesador TMS320C25, elemento central de la misma.

Capítulo V. Presenta el desarrollo del software necesario para el funcionamiento de la Arquitectura de Procesamiento Digital de Imágenes. Primeramente se presenta el programa de aplicación desarrollado en Pascal que permite la comunicación y control del sistema, así como la transferencia del programa para procesamiento y la información de la imagen a procesar. Seguidamente, se describen los programas de procesamiento desarrollados en lenguaje ensamblador del TMS32025.

Capítulo VI. Se hace una evaluación de la arquitectura en su conjunto, presentando información sobre tiempos de procesamiento del sistema para los algoritmos implementados y así evaluar el cumplimiento de los objetivos planteados. Además se hace una reflexión sobre posibles mejoras a la arquitectura para añadir versatilidad a la misma.

I.1 OBJETIVO

Diseño, desarrollo y prueba de una Arquitectura para el Procesamiento Digital de Imágenes de tipo modular, basada en un procesador rápido de señales y con la cual se pretende llevar a cabo procesamiento rápido de imágenes (y en algunos casos en tiempo real).

I.2 DESCRIPCION GENERAL

La figura 1.1, muestra un diagrama de bloques de un sistema básico para realizar procesamiento digital de imágenes. La información es adquirida por una cámara de video, digitalizada y luego almacenada digitalmente (un dato a la vez) en un registro denominado registro **fuente**. La información se entrega secuencialmente al bloque **operando** el cual permite aplicar sobre la información alguna transformación de tipo puntual, entregando posteriormente

su resultado a otro registro denominado registro **resultado**. Finalmente, la información así procesada es desplegada a través de un monitor de video.

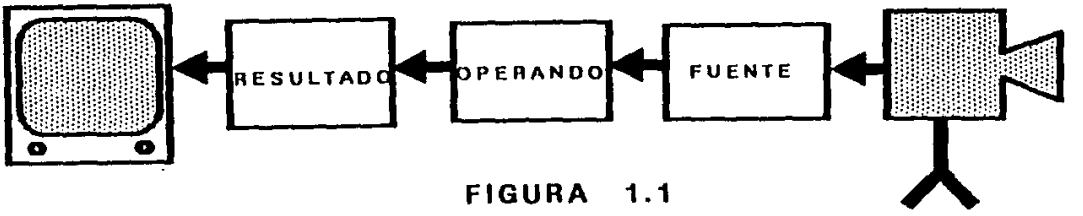


FIGURA 1.1

Siendo un sistema de procesamiento secuencial, este permite que sea realizado en tiempo real, aunque el tipo de procesamiento que se pueda realizar queda restringido a la aplicación de una tabla de correspondencia de la información conocida como LUT (Look-Up Table).

Concibiendo un diseño similar al de la figura 1.1, se presenta la arquitectura de la figura 1.2. El registro fuente se ha reemplazado por una memoria fuente con capacidad de almacenar una imagen. El bloque operando es reemplazado por un procesador programable y finalmente el registro resultado es reemplazado por una memoria resultado de dimensiones iguales a la memoria fuente, logrando resultados similares a los del sistema descrito en la figura 1.1, es decir, adquirir, procesar y desplegar, pero con un incremento significativo en la capacidad y versatilidad de procesamiento del sistema al contar con un elemento procesador programable.

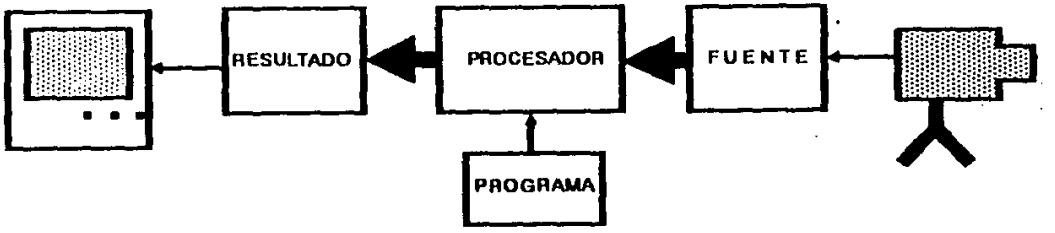


FIGURA 1.2

Si se utiliza únicamente un elemento procesador, éste soporta toda la carga de procesamiento, redundando en la lentitud del sistema. Una mejora sobre la arquitectura anterior se presenta en la figura 1.3. Esta contempla la utilización de varios elementos procesadores actuando paralelamente, aumentando la velocidad de procesamiento que puede llegar a ser un número de veces igual al número de procesadores incluidos.

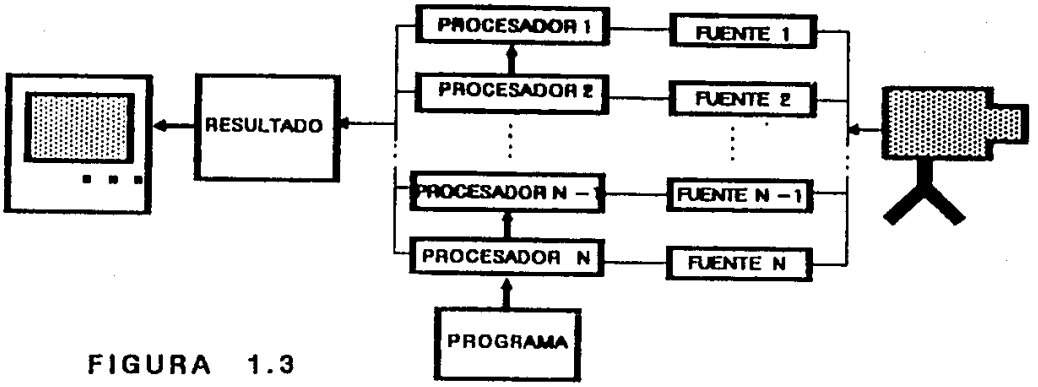


FIGURA 1.3

La arquitectura propuesta pertenece a la clase de máquinas conocidas como SIMD (Single Instruction Multiple Data stream). Consiste en un arreglo de N procesadores acomodados vectorialmente, donde cada procesador contiene su propia memoria local. Un controlador central transmite la secuencia de instrucciones a ejecutar por los procesadores, los cuales obedecen a la misma instrucción simultáneamente, pero operan sobre la información que contienen localmente.

El desarrollo de una arquitectura como la expuesta en la figura 1.3 y orientada hacia una construcción modular, involucra las siguientes partes:

- Módulo para adquisición de imágenes.
- Módulo controlador.
- Módulo de procesamiento.
- Módulo para despliegue de imágenes.
- Módulo de conexiones.
- Módulo de alimentación.

Estos módulos se describen en la figura 1.4

En la evaluación práctica de una arquitectura de procesamiento, es necesario contar con una herramienta para monitoreo y control del sistema (en este caso se utiliza una PC) que a través de un módulo interfase con la PC establezca la comunicación con el módulo de conexión y el módulo de procesamiento. El módulo interfase con la PC no está contemplado como parte de la arquitectura original, pero es importante en la etapa de diseño y puesta en marcha del sistema. El resto de los módulos es factible simularlos con la ayuda de la misma PC.

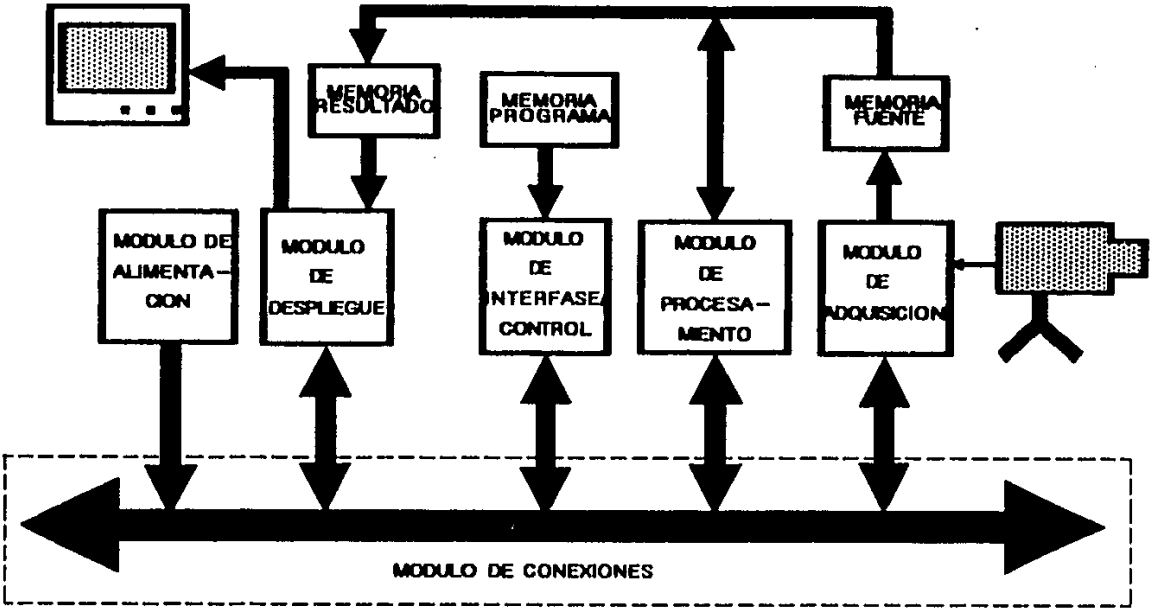


FIGURA 1.4

Como parte del presente trabajo, se propone implementar únicamente las etapas correspondientes a los módulos de procesamiento, módulo de interfase PC y módulo de conexiones. Los módulos restantes no se contemplan aquí y son considerados como una etapa posterior al presente trabajo.

Los módulos del sistema que no se incluyen (controlador, alimentación, adquisición y despliegue de imágenes) son emulados con ayuda del Módulo de interfase PC, aprovechando los recursos de la computadora personal. De este modo, se genera el ambiente de comunicación necesario para la prueba del sistema teniéndose la opción de integrar posteriormente los módulos restantes, y por lo tanto la arquitectura planteada no quede desligada de su concepción inicial.

Por lo anterior, se realiza el diseño de una arquitectura básica de procesamiento, con los módulos:

- Interfase PC
- Conexiones.
- Procesamiento

Para llevar a cabo la puesta en marcha y prueba de la arquitectura propuesta, se implementa un diseño en hardware como se ilustra en la figura 1.5.

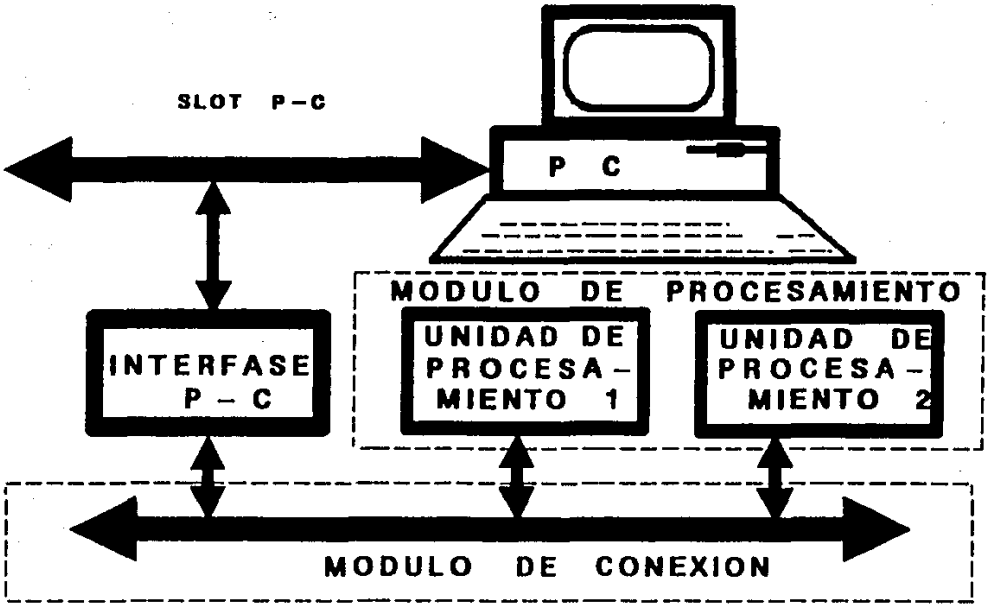


FIGURA 1.5

Se considera una arquitectura con características estáticas, es decir, que las imágenes están almacenadas en un disco flexible o en disco duro, las cuales han sido previamente digitalizadas con ayuda de un digitalizador. Así, las imágenes son transferidas al Módulo de Procesamiento. Cada unidad de procesamiento, contiene un banco de memoria fuente (MF) de tal modo que tienen acceso a la información completa de la imagen en cualquier momento.

Se incluye asimismo un Módulo de Interfase PC, que es el encargado de establecer el ambiente de comunicación con la PC y contiene un banco de memoria denominado memoria de programa (MP) para almacenar el programa de procesamiento.

El procesador de señales TMS, contenido en cada unidad de procesamiento, tiene acceso a la memoria MF y a la memoria MP para realizar el procesamiento y transferir sus resultados a un tercer banco de memoria denominado memoria resultado (MR). Al incluir dos unidades de procesamiento, la memoria MR es dividida en dos partes, donde cada unidad procesa la mitad de los datos totales.

Tanto las memorias MP, MF y MR, se accesan desde la PC a través del módulo de interfase PC con ayuda de un programa de aplicación del sistema. Se pretende que la Imagen Fuente y la Imagen Procesada se desplieguen en el monitor de la PC para apreciar y comparar resultados.

I.3 JUSTIFICACION

El procesamiento digital de imagénes involucra grandes volumenes de información, a los cuales se les aplican una gran cantidad de operaciones en tiempos aceptables de acuerdo a una aplicación específica, para lograr estos tiempos se requieren dispositivos electrónicos de alta velocidad y capacidad de procesamiento. En la actualidad los procesadores de señales digitales (DSP) como el TMS320C25, ofrecen grandes posibilidades de procesamiento como matemáticas intensivas y altas velocidades de operación.

En una arquitectura con un sólo procesador de imágenes, éste debe realizar todo el trabajo sobre la información existente, sin embargo, la información de una imagen puede ser distribuida entre varios procesadores con el fin de reducir el tiempo de procesamiento de una imagen, de esta forma se puede lograr el procesamiento de imágenes en tiempo real.

Entre las áreas de aplicación para el procesamiento de imágenes se tiene:
- Robótica (identificar o describir objetos, tareas industriales, etc.)
- Imágenes Aéreas y de Satélite (mejoramiento, análisis de recursos naturales, predicción del tiempo, etc)
- Vigilancia (detección de intrusos, rastreo, etc.)
- Medicina (Diagnosis de anormalidades, Patología, Citología etc.).

Como una aplicación evaluativa de la arquitectura, se implementan las funciones: filtro paso bajas, filtro Laplaciano (detección de bordes), el operador de Sobel (para detección simultánea de bordes verticales y horizontales) y filtros diferenciadores horizontales y verticales, utilizando el algoritmo de convolución de una matriz imagen (200x200 pixels) con una matriz plantilla (3x3 operadores) en el procesamiento de imágenes previamente digitalizadas o patrones desarrollados por software. Otros procesos pueden ser: detección de áreas, comparaciones con patrones y operaciones puntuales. Las imágenes así procesadas son almacenadas en disco o pueden desplegarse inmediatamente después de terminado el procesamiento.

II. SISTEMAS DE PROCESAMIENTO DIGITAL DE IMAGENES

II.1 INTRODUCCION

El Procesamiento Digital de Imágenes (PDI) es la acción de emplear técnicas computacionales para la manipulación y/o modificación de la información de una o varias imágenes. Los resultados obtenidos en el PDI pueden ser empleados, por ejemplo, para el control automático en procesos industriales, en el reconocimiento de patrones, la compresión y transmisión de imagen, en equipos de ultrasonido para el monitoreo de fetos, o bien para efectuar análisis estadísticos sobre la información contenida en una imagen.

El desarrollo de la tecnología orientada a Sistemas para Procesamiento Digital de Imágenes (SPDI), se vio influenciada por el conocimiento en torno al Sistema de Visión Humano. Este sistema de visión está integrado de varios sensores de señales, donde uno de los más importantes es la retina, que contiene dos tipos de fotoreceptores llamados bastones y conos. En su funcionamiento, ambos fotoreceptores proporcionan al nervio óptico una señal representativa de una imagen captada por el ojo. Esta señal es procesada en el cerebro y mediante un sistema de realimentación, compuesto de señales provenientes de distintos órganos, proporciona respuestas de adaptación del individuo al medio que lo rodea.

En ciertos aspectos, un SPDI tiene mayores ventajas sobre el Sistema de Visión Humano, debido a que existe una mayor consistencia en las decisiones determinadas en base a parámetros cuantificables, ya que la calidad en la medición de dichos parámetros es mayor al no presentar errores aleatorios debido a fatiga o distracciones, además de contar con una mayor velocidad de respuesta y ancho de banda dentro del espectro electromagnético.

Diversos factores como el desarrollo de la Optica, Computación, Matemáticas, Estadística y las recientes innovaciones tecnológicas en la Electrónica, nos prometen un amplio campo de trabajo en el PDI. El empleo en procesos industriales es cada día mayor, donde el desarrollo de esta tecnología requiere de un hardware confiable a un costo razonable, así como individuos con conocimientos en hardware y programación para aplicarla en la solución de diversos problemas y/o necesidades.

II.2 BREVE HISTORIA DEL PROCESAMIENTO DIGITAL DE IMAGENES

La evolución de los SPDIs ha sido fragmentada, según el uso de esta tecnología durante las últimas décadas, en tres aplicaciones importantes :

- La carrera espacial
- Industria militar
- Y limitadamente a la industria convencional

En los inicios del PDI (1950), sólo se reconocían patrones estáticos. Actualmente se emplea para el mejoramiento, restauración, codificación y transmisión de imágenes, al mismo tiempo se ha incorporado a la Computación, para extender la complejidad de los procesos digitales.

Durante 1960 en la carrera espacial, las imágenes enviadas por las naves espaciales de exploración lunar y planetaria fueron las primeras en procesarse, con el fin de obtener la mayor información posible. El Jet Propulsion Laboratory mediante los diseños de F. Billingsley y R. Brant, dieron la pauta en el PDI. Uno de tales diseños fue el Video Film Converter, con el cual no solo se almacenaba el video analógico sino que además se digitalizaban y grababan imágenes. Junto con el software de procesamiento de imágenes desarrollado por R. Nathan en una computadora IBM 7094, se mejoraban los contrastes, la corrección fotométrica y geométrica, así como remoción del ruido en la imagen [Castleman,1979].

John Morecroft implementó la transformada de Fourier y otras técnicas de remoción de manchas en la síntesis de imágenes de cráteres en una computadora NCR 102D. Nathan igualmente desarrolló la Función de Transferencia Modulada (MTF), empleada en la filtración de datos televisados. En 1964 en el programa espacial MARINER se enviaron datos de video digitalizados, los cuales eran almacenados en memorias de núcleo, para ser desplegadas en un Tubo de Rayos Catódicos (CRT) y fotografiados por una cámara.

En la compañía BIOMEDICIN se empleo el hardware y software del Jet Propulsion Laboratory en la solución de problemas relevantes de Medicina y Biología. En Medicina, Robert Selzer trabajó en el procesamiento de imágenes proyectadas por los rayos X, con la finalidad de mejorar el diagnóstico de enfermedades. En la actualidad existen imágenes proporcionadas por radiología, resonancia magnética nuclear y muestreo ultrasónico, que son empleadas para detección de tumores u otros padecimientos.

En otras ramas se mejoró el procesamiento de la imagen digital del sonar y radar, para la detección o reconocimiento de varios tipos de estructuras dentro un medio o como guía de navegación en aeroplanos, barcos, etc.

El programa MARINER para 1969 empleaba un complejo esquema de codificación para obtener más imágenes mientras se reducía la cantidad de datos transmitidos, el objetivo era obtener un mapeo de la superficie de Marte. Conforme se iniciaban nuevos programas espaciales, el incremento de las innovaciones era mayor, hasta llegar a la adquisición de imágenes por satélites para el reconocimiento de fuentes minerales de la Tierra o el mapeo geográfico, así como en la predicción de las condiciones ambientales de una determinada zona.

En 1970 los filtros digitales fueron un producto de las grandes ventajas de la electrónica de estado sólido. Estos proporcionaban un alto rendimiento a través de dispositivos electrónicos a bajo costo. Utilizar los filtros digitales significaba mejorar la calidad de la información empleada en un proceso.

En la década de los 80s, ante la gran variedad de arreglos representativos de datos visuales, se creó la necesidad de desarrollar equipos de una alta flexibilidad y con la capacidad de procesar información. Uno de los equipos que ha cumplido con tales características y de reciente creación es la computadora personal (PC). Por su parte, con el desarrollo y comercialización de las computadoras se han logrado crear arquitecturas enfocadas a procesar información digital con mayor velocidad. Dichas arquitecturas son las de arreglo en paralelo y de operación tipo pipeline. Con este tipo de herramientas ha sido factible pensar en procesar imágenes en tiempo real.

Una característica importante de los dispositivos de estado sólido ha sido la capacidad de implementar algoritmos para manipular señales. Con los avances en la tecnología digital se llegó a la creación del microprocesador, y con ello el Procesamiento Digital de Señales (DSP). De esta forma, el campo del PDI se vio enriquecido, reduciéndose los costos y los grandes volúmenes de equipo al emplear circuitos de muy alta escala de integración (VLSI) y aumentando la velocidad de procesamiento con el desarrollo de arquitecturas especializadas en el DSP.

Algunos rasgos importantes de las arquitecturas de DSP son el emplear arreglos en paralelo, tener instrucciones propias del DSP, poder ejecutar millones de operaciones de DSP por segundo, tener la capacidad de implementar y poder reprogramar los algoritmos propios del DSP. Con el uso de dispositivos especiales de DSP se ha incrementado la capacidad de

procesamiento y funcionamiento de los circuitos relacionados a telecomunicaciones, instrumentación, control, análisis de voz o imagen, etc.

Inevitablemente, las investigaciones que buscan alcanzar la superioridad militar se fundamentan en la ingeniería de procesamiento dentro de las áreas de codificación, restauración, reconocimiento de objetivos militares, rastreo y análisis de zonas para su reconocimiento, y en especial las relacionadas con las partes no-visibles del espectro electromagnético.

II.3 ETAPAS DEL PROCESAMIENTO DIGITAL DE IMAGENES (PDI)

Un sistema de PDI está compuesto de aquellos elementos necesarios para obtener una representación digital de una imagen, analizar la información contenida en ella, así como presentar los resultados de dicho análisis en algún formato, preferentemente de fácil interpretación (Fig.2.1). De esta manera, se pretende estudiar el contenido en una imagen, generar resultados en base a condiciones predeterminadas, y con estos resultados, poder proporcionar ya sea una nueva imagen, una decisión e inclusive una señal de control, mismas que permitan la solución a problemas prácticos, tales como: el control por visión de un robot, monitorear el nivel de calidad en un proceso de manufactura, remosión de ruido en imágenes, compresión de imágenes en telecomunicaciones, etc.

Las etapas del PDI se pueden agrupar en:

- Adquisición de la Imagen
- Digitalización de la Imagen
- Procesamiento de la Imagen
- Salida o Despliegue de la Imagen

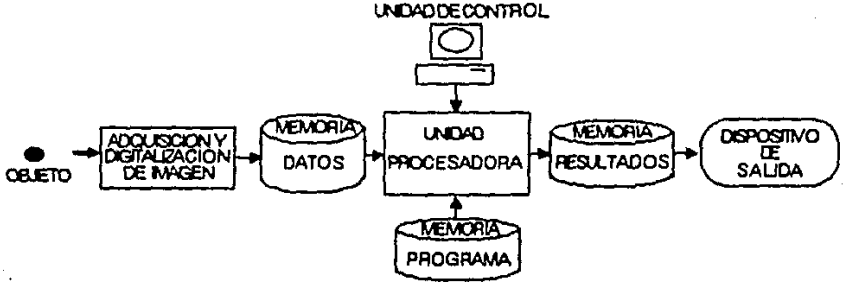


Figura 2.1 Etapas del PDI

II.3.1 Adquisición de la Imagen.

De una manera global, podemos decir que el universo de las imágenes está compuesto por:

- Las imágenes visuales, que incluyen a las ilustraciones (fotografías, dibujos y pinturas) y las formadas por medios ópticos.
- Las imágenes de representación matemática en base a una función continua o discreta.
- Las imágenes físicas no visibles por el ojo humano de manera inmediata, como: los mapas de densidad o crecimiento de la población, imagen por rayos X, tomografías computarizadas, imagen por ultrasonido, imagen de un radar, etc. [Castleman,1979].

Cada uno de estos grupos genera las imágenes empleando diferentes métodos o formas. Por ejemplo, una imagen visual es generada por las distribuciones en el espacio de la intensidad de luz.

La adquisición de una imagen es el proceso de transformar la imagen visual de cualquier objeto físico con sus características intrínsecas (color, textura, etc.), en un conjunto ordenado de datos. En dicho proceso es importante ubicar varios aspectos técnicos que redundarán en la calidad de la información obtenida:

a) Iluminación.
b) Formación y Enfoque de la Imagen.
c) Detección de la Imagen.

a) Iluminación.

La iluminación sobre un objeto busca resaltar sus características físicas, como la textura, el color, contornos, etc., donde el nivel total de iluminación determina la calidad de los datos obtenidos. Así, una deficiente iluminación provoca manifestaciones de ruido en los datos.

b) Formación y Enfoque de la Imagen.

Utilizando una cámara de video, un SPDI captura la imagen mediante un sensor, a partir del cual una imagen visual es convertida a una señal eléctrica. En la formación de la imagen sobre el sensor, se emplea una lente, por medio de la cual la imagen es ampliada o reducida cuidando el enfoque para obtener la mayor nitidez de la misma. Los parámetros

asociados con la óptica de la lente son la distancia focal, profundidad de campo y montado del lente, principalmente.

c) Detección de Imagen.

Los SPDI tienen un dispositivo electro-óptico, que convierten la radiación electromagnética proveniente de una imagen en una señal eléctrica. Por ejemplo, una cámara de video (Fig.2.2) tiene el elemento sensor localizado en su plano focal, y mediante un barrido de la superficie fotosensible del sensor se produce la señal de voltaje analógico, representativa de la imagen que se forma sobre dicha superficie.

Los Tubos de Rayos Catódicos (CRTs), fueron los primeros dispositivos donde se pudo realizar la detección de una imagen, siendo empleados en las cámaras de video originalmente desarrolladas para la TV comercial en 1930, antes del desarrollo de los materiales de estado sólido. Básicamente son tubos al vacío revestidos por una mezcla cristalina, conteniendo un elemento de alta sensibilidad a la luz (fósforo, silicon, selenio-zinc, etc.), la cual es depositada sobre una capa de aluminio transparente. En la actualidad existen cinco tipos de tubos: Vidicon, Newvicon, Ultricon, Plumbicon, Arreglo Silicon y Saticon. La diferencia entre los tipos de tubos es el material empleado para detectar luz. En 1960 se desarrollan las cámaras de estado sólido (CCD), conteniendo fotodiodos como elemento sensor. Estos elementos forman parte de un arreglo lineal o rectangular y la señal de salida es la obtenida por cada diodo sensor.

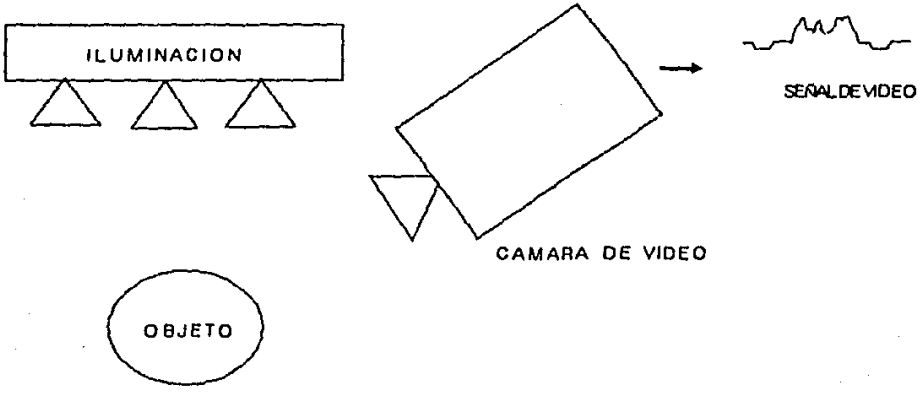


Figura 2.2 Adquisición de Imagen

II.3.2 Digitalización de la Imagen.

La transformación de la señal analógica de video al formato digital, requiere:

- El muestreo de la señal de video.
- La cuantificación del valor de las muestras en apropiados niveles de intensidad de luz.
- La digitalización de los valores cuantificados mediante el uso de convertidores analógicos/digitales (A/D).

El muestreo es el proceso de obtención de valores instantáneos del voltaje analógico representativo de la imagen. Si se incrementa la velocidad de muestreo del sistema se incrementa igualmente el número de datos y por lo tanto la calidad de la imagen.

La cuantización de los valores muestreados de la señal de video es necesaria para establecer el número adecuado de niveles de intensidad de luz, empleados en el sistema.

La etapa de conversión A/D es la encargada de obtener la representación digital de la intensidad de luz, correspondiente al nivel de voltaje muestreado de la señal de video.

Un digitalizador es en general un cuantizador físico de la intensidad de luz. Y en esta etapa de digitalización se pueden sufrir degradaciones por diversos factores, por ejemplo, si en la curva de transferencia del digitalizador se presenta secciones no-lineales, éstas provocarán alteraciones en la imagen digital obtenida, ya que la información real no será representada adecuadamente, por esta razón es importante obtener esta curva de transferencia para conocer la calidad del digitalizador. Por otra parte el ruido inherente en la digitalización es otra fuente de degradación de la imagen y puede ser relativamente pequeño, si el número de niveles de intensidad de luz es apropiado. El digitalizador, finalmente proporciona una imagen digitalizada que es transferida a un dispositivo de almacenamiento para su posterior manipulación, figura 2.3.

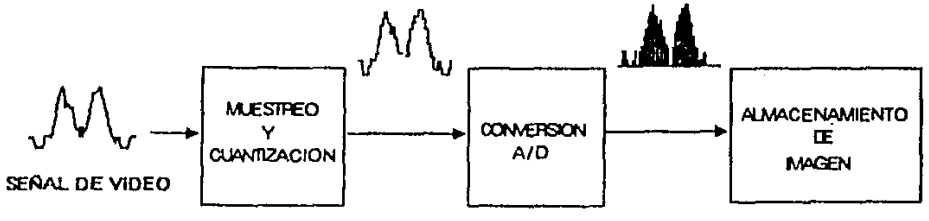


Figura 2.3 Elementos del Digitalizador

Error en la Cuantización de la Imagen.

La transformación de una señal analógica a un formato digital es posible mediante el muestreo de dicha señal, y para efectuar el muestreo es importante considerar el Teorema de Muestreo de Nyquist para determinar los intervalos uniformes de tiempo y garantizar la reconstrucción de la señal a partir de las muestras obtenidas.

La asignación de los valores discretos se realiza mediante el redondeo de los valores muestreados de intensidad de luz a niveles prestablecidos, por lo que se presenta una incertidumbre sobre la precisión de un valor dado. Así, un valor digital de intensidad de luz puede representar una intensidad menor o igual a la indicada, y la tolerancia del error en el valor depende del número de niveles de intensidad de luz empleados en la transformación, de este modo entre mayor sea el numero de niveles menor será la tolerancia.

II.3.3 Procesamiento de la Imagen.

La etapa de procesamiento contiene elementos de hardware y software para su funcionamiento. El uso de hardware para realizar la manipulación de los datos, logra incrementar la velocidad de los sistemas, y posibilita la implementación de algoritmos básicos en relación a las técnicas más comunes para proceso de datos, tales como: eliminar ruido, filtrado, mejorar cambios no deseados, modificar la escala de niveles de gris, etc.

Una imagen almacenada en un dispositivo de estado sólido o medios magnéticos, es la fuente que permite establecer la correspondencia entre una imagen de entrada y una de salida, donde la imagen de salida pueda ser mejorada en comparación con la imagen de entrada y/o se puedan obtener las características de interés, así como tener la posibilidad de obtener un nuevo formato de la imagen de salida. La imagen de salida ó resultado, es mantenida igualmente en otro dispositivo de almacenamiento, para posteriormente analizar los resultados (Fig.2.4).

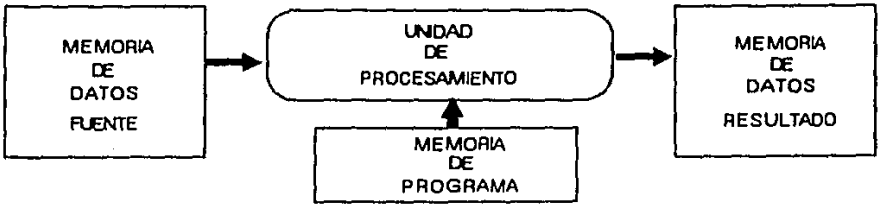


Figura 2.4 Procesamiento de Datos Digitales

II. 3.4 Salida o Despliegue de Resultados.

Los sistemas de PDI pueden tener la capacidad de salida en forma de imagen visual, datos impresos, señales de control o despliegues especiales a partir de una imagen previamente procesada. Dentro de los despliegues especiales se incluyen los que son usados para la identificación de la posición de cualquier objeto, en determinación de características de dimensiones numéricas (perímetros, áreas,etc.), o bien, en la clasificación de objetos.

Existen dos tipos básicos de despliegue, el permanente y el volátil. El permanente produce una imagen en papel, film u otro medio de grabado permanente, el volátil proporciona una imagen temporal como es el caso de los monitores de video, figura 2.5.

Ciertos factores en la etapa de despliegue de la información determinan su calidad, tales como:

a) Las dimensiones físicas. Un inadecuado tamaño del dispositivo de despliegue reduce la apreciación adecuada de la imagen.

b) La resolución fotométrica del sistema produce una correcta brillantez o densidad en cada posición de la imagen.

c) La linealidad del dispositivo. Una curva de transferencia donde la no-linealidad se presente entre el 10 y 20 % de la densidad o brillantez, no será apreciable por el ojo humano. En despliegues volátiles la curva de transferencia depende de los controles de brillantez y contraste del monitor.

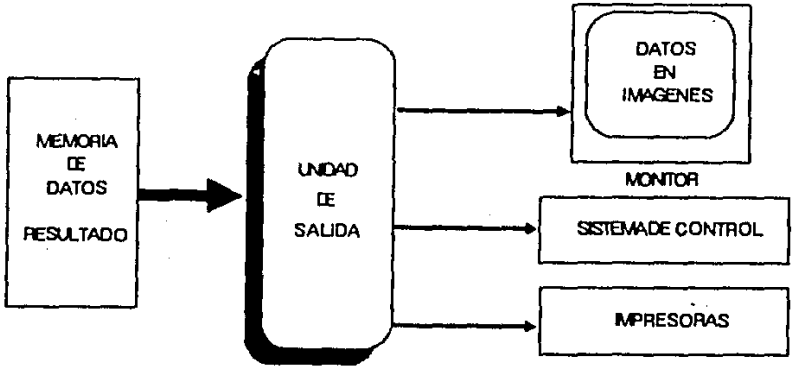


Figura 2.5 Despliegue de Imágenes

II.4 CONCEPTOS Y DEFINICIONES DEL PDI

Uno de los objetivos de procesar una imagen es corregir las limitaciones físicas del hardware usado para la adquisición de la imagen, por ejemplo el ruido aleatorio o sistemático contenido en los datos, así como analizar características o parámetros de interés. De ello podemos decir que el PDI busca mejorar características o extraer información de un conjunto de datos. En ambos casos es necesario conocer los conceptos básicos sobre imágenes y el PDI. Algunas características y conceptos importantes se definen en los siguientes párrafos.

II.4.1 Conceptos Generales.

Pixel: Una imagen digitalizada puede ser descrita por una matriz de MxN elementos, donde cada elemento P(i,j) se le denomina pixel (picture element). El valor numérico o magnitud del pixel indica un promedio de intensidad de luz sobre el área de dicho elemento de la imagen.

Ventana: Una ventana es una subregión rectangular de una imagen y es determinada por cuatro esquinas, correspondientes a los elementos P(a,b), P(a,c), P(d,b) y P(d,c).

Localidad del pixel: Son las coordenadas dentro del arreglo matricial MxN representativo de la imagen, mediante las cuales identificamos un pixel específico en su forma más elemental.

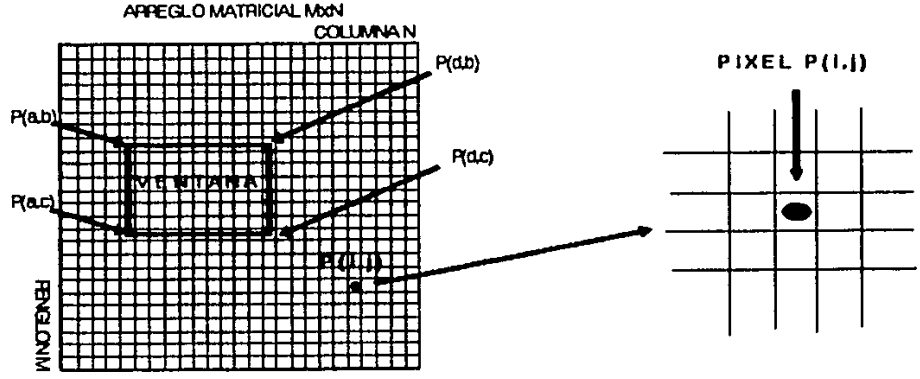


Figura 2.6 Representación matricial de imágenes.

Escala de Gris.

En un sistema digital se puede representar la intensidad de luz mediante un valor, sin embargo para presentar los datos en un medio visual se recurre, en el caso de un sistema en blanco y negro (monocromático), a la creación de una gama de tonalidades de gris para representar dichos niveles de intensidad.

En el caso más simple, donde el sistema digital tiene un bit para determinar la intensidad de luz, las regiones con falta de esta intensidad se indican por un cero y las de mayor intensidad por un uno. La figura 2.7 muestra un arreglo matricial de 5x4, donde se tiene en la localidad (1,1) un valor "0" indicando ausencia de luz y en (5,4) el valor "1" para indicar máxima intensidad de luz.

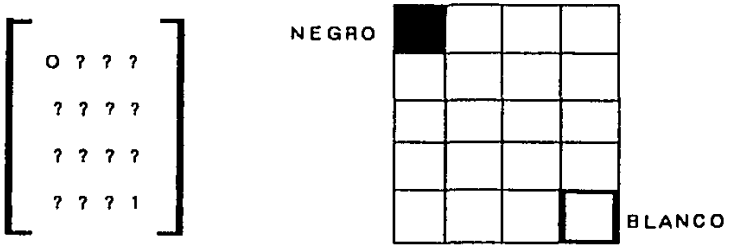


Figura 2.7 Representación matricial de brillantez.

Incrementar el rango de información, implica ampliar el número de bits en la representación del valor del pixel. El número de niveles de intensidad de luz en una escala de tonos de gris está dado normalmente por una potencia de 2, donde el valor mínimo de la escala representa al negro, y el máximo valor es asignado al blanco.

POTENCIA	NIVELES DE GRIS	RANGO
2^1	2 VALORES	DE 0 A 1
2^3	8 VALORES	DE 0 A 7
2^4	16 VALORES	DE 0 A 15
2^6	64 VALORES	DE 0 A 63
2^8	256 VALORES	DE 0 A 255

TABLA 2.1 Niveles de gris

La tabla 2.1 nos muestra que valores superiores a 64 niveles de tonos de gris proveen un número mayor de niveles con respecto a la capacidad de ojo humano. Un individuo es capaz de diferenciar cerca de 10 a 15 niveles de gris, sim embargo, puede identificar 40 diferentes tonos de gris. En la actualidad se ha limitado el valor a 256 niveles de tonos de gris para fines prácticos.

Un sistema puede tener la capacidad de manejar 256 tonalidades de gris (Fig.2.8), pero los factores asociados con la aplicación pueden hacer variar el número de tonos de gris. Si un sistema sólo utiliza una menor cantidad de tonos de gris, lo más conveniente es disminuir la escala de tonos de gris.

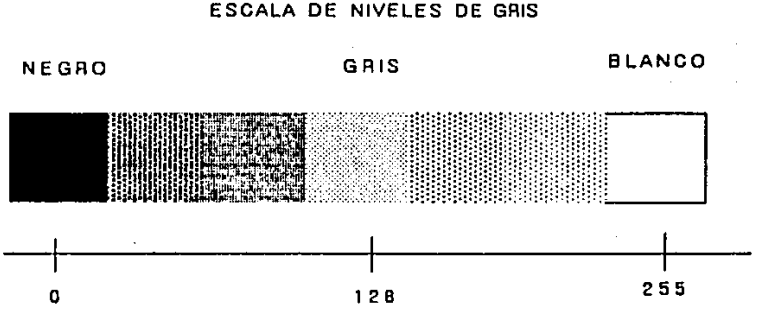


Figura 2.8 Escala de niveles de gris.

En general, al incrementar el número de niveles de intensidad de luz se mejora la calidad de la imagen, además de la oportunidad de aumentar la gama de tonalidades de gris en la presentación de la imagen por medio de un sistema monocromático.

Histogramas.

Un histograma es la representación gráfica de la frecuencia de ocurrencia de cada intensidad de luz (tono de gris) en una imagen. Este es construido en base a una imagen digital y por el conteo de los pixeles en cada nivel de gris, para finalmente dibujar la frecuencia de conteo de los pixeles en cada nivel de gris. El histograma (Fig.2.9) está formado por el eje de las abscisas(x), que corresponde al nivel de gris, y el eje de ordenadas (y) al número de pixeles con el mismo nivel de gris.

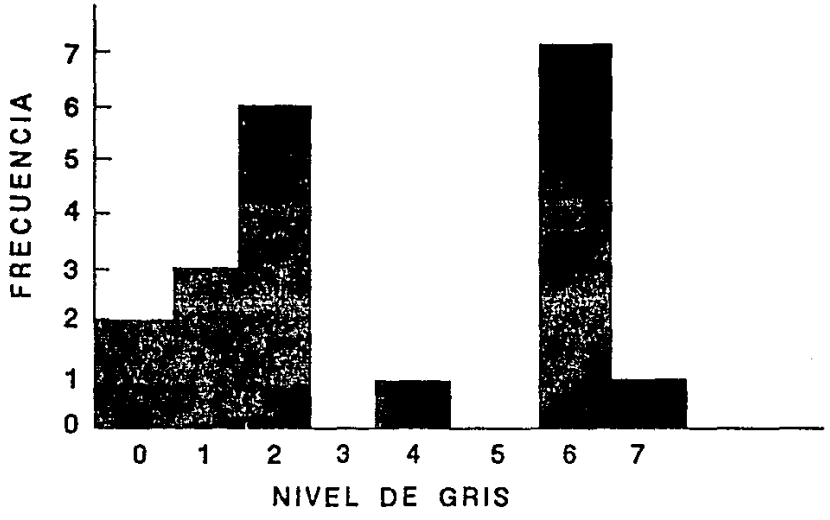


Figura 2.9 Histograma para una escala de 8 tonos de gris

La información que proporciona un histograma es la frecuencia de ocurrencia de un pixel para varios tonos de gris, pero sin determinar su localización o distribución en la imagen. Al realizar un cambio de posición de un objeto liso perpendicular al plano, con una iluminación uniforme, un fondo de textura y color iguales, su histograma no debe alterarse, de lo contrario podemos concluir que nuestro sistema de adquisión no es adecuado.

El histograma para una imagen específica es único, sin embargo la imagen para un histograma específico no es única. La información proporcionada por un histograma recae en los tonos de gris y mediante una modificación en la escala se puede lograr el más alto contraste en las características de la imagen.

II.4.2 Parámetros de una Imagen de Video.

La señal eléctrica representativa de una imagen es obtenida de dos formas, por rastreo del elemento sensible para cámaras de tubos de rayos catódicos o por la lectura de la salida en el arreglo de sensores para cámaras de estado sólido. En ambos casos el voltaje está en función del tiempo y la señal contiene información sobre el promedio de intensidad de luz en cada punto de la imagen.

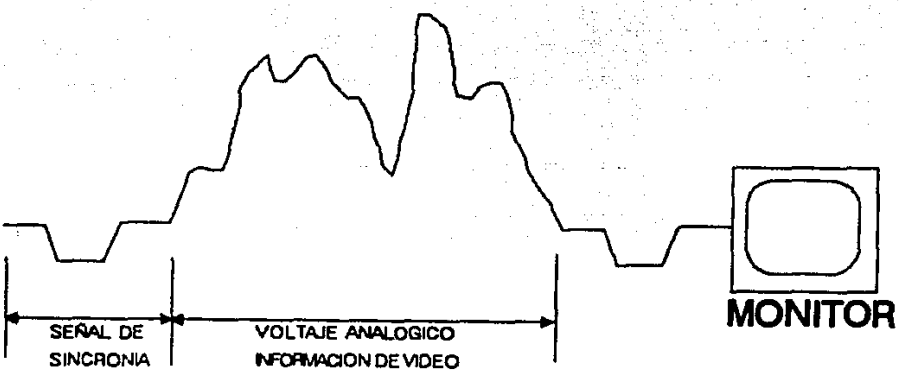


Figura 2.10 Señal de Video

La señal de video analógica (Fig.2.10) está concebida dentro de un formato estándar con el propósito de ser empleada por los monitores de TV comerciales, en este formato se tienen contenidas señales de sincronía que son necesarias para la formación de imágenes. De esta forma es posible conectar directamente al monitor la señal de video generada por la cámara de T.V.

En los equipos de video han adoptado un formato estándar llamado RS-170 desarrollado por la industria de la televisión, empleando un estrecho ancho de banda. De esta manera, es posible minimizar el espectro en la frecuencia. Un formato RS-170 estándar requiere para la generación de una imagen el uso de una estructura de líneas alternadas, que se denomina entrelazamiento.

La pantalla del monitor de video es recorrida dos veces por un haz de electrones para formar una imagen, con los dos grupos de líneas entrelazados sobre la pantalla como se observa en la figura 2.11. Cada recorrido vertical se le llama "campo", y dos campos consecutivos constituyen un "cuadro".

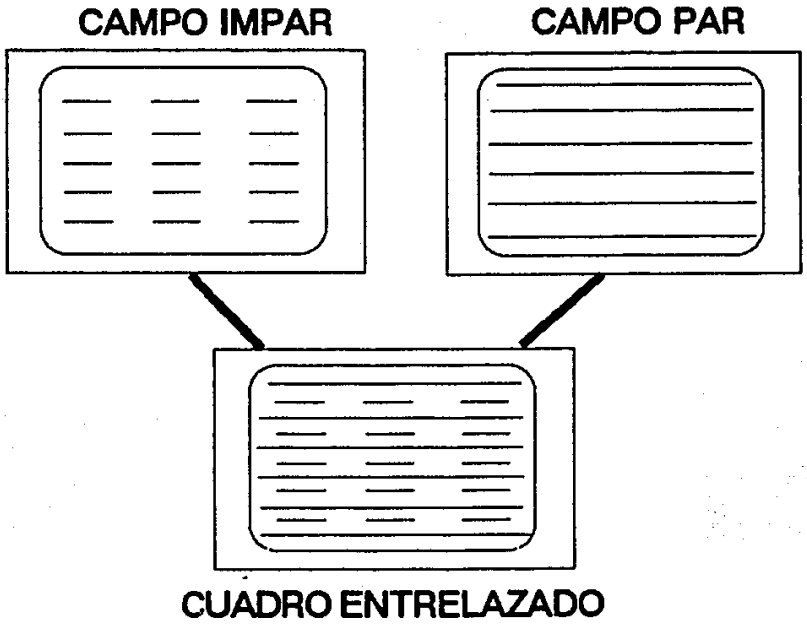


Figura 2.11 Constitución de un cuadro.

Una imagen presentada en un monitor contiene 480 líneas de 525 generadas en 1/30 de segundo, mismas que son dividas en dos campos de 240 líneas, denominados campo par y campo non. Cada campo es muestreado en 1/60 de segundo y para obtener una imagen completa se requiere de 1/30 de segundo. La señal compuesta especificada por el formato RS-170 estándar, contiene todos estos tiempos necesarios para la formación de la imagen en monitores de televisión.

El período de tiempo para un ciclo de rastreo completo de una línea es de 63.5 microsegundos, que consiste de una etapa activa de rastreo y una inactiva de retorno de trazo (blanking), donde la duración del tiempo activo de rastreo es de 52.1 microsegundos y el retorno de 11.4 microsegundos (Fig.2.12).

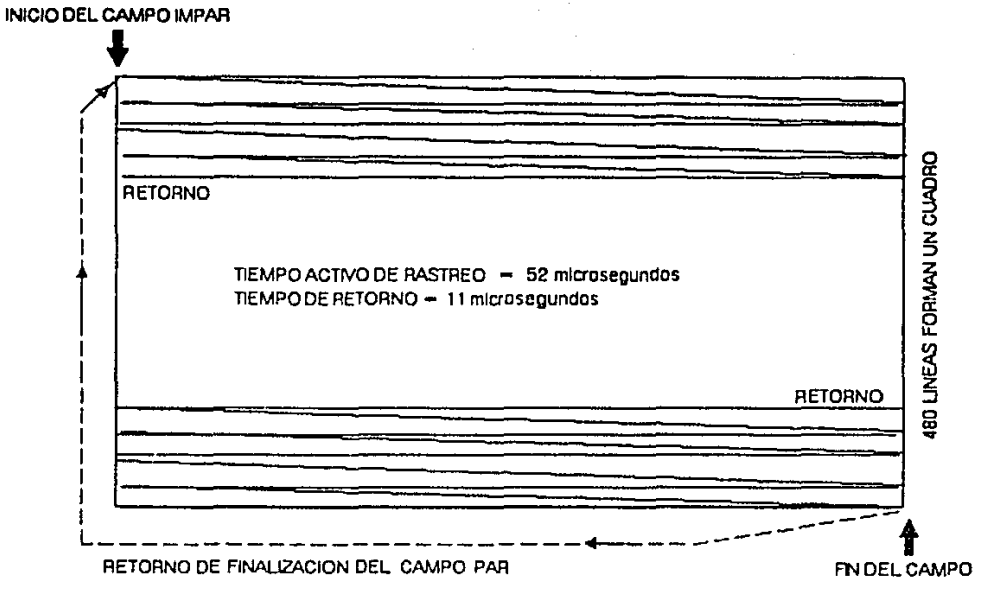


Figura 2.12 Formato RS-170 de la señal de video.

La resolución vertical es limitada por el número de líneas de rastreo y la horizontal es determinada por la razón de muestreo. Por ejemplo, en el PDI es común emplear imágenes matriciales en arreglos de 256x256, consecuentemente para generarla se necesitan 256 tiempos de trazos activos en el rastreo de cada renglón de la matriz y cada trazo tendrá 256 muestras que forman las columnas de dicha matriz. El propósito de la TV de alta resolución (HDTV) es emplear 1125 líneas en 1/30 de segundo.

Entre otros sistemas de video estándar que se han desarrollado están el NTSC para imágenes en color, el cual incrementa el formato RS-170 mediante el empleo de una subportadora de color de 3.48 MHZ sobre la señal de video. El RS-343-A incluye imágenes con 675, 729, 875, 945 y 1024 líneas verticales y el Sistema Europeo Monocromático (CCIR) que tiene 625 líneas a razón de un cuadro en 1/50 de segundo. PAL es el equivalente a NTSC para imágenes de color en el sistema Europeo.

II.4.3 Características de la Imagen.

Comprender el proceso de percepción visual implica desarrollar mediciones de la fidelidad de la imagen. La información contenida en éstas representa la distribución en el espacio de cantidades físicas como la luminancia y las frecuencias espaciales propias del objeto. La percepción de tales características permiten ser representadas en la imagen por atributos como la brillantez, color y bordes.

A) Luz.

La luz es la radiación electromagnética que estimula nuestra respuesta visual. Esto es expresado con una distribución de energía espectral $L(\lambda)$, donde λ es la longitud de onda que en el espectro de la región visible que está entre 350nm y 780 nm.

B) Luminancia.

La luminancia es la intensidad luminosa espacialmente distribuida de un objeto en una dirección predeterminada sobre una plano imaginario (perpendicular a esta dirección dada), y es definida como:

$$F(x,y) = \int I(x,y,\lambda)V(\lambda)\, d\lambda$$

donde $I(x,y,\lambda)$ es la distribución de la luz del objeto y $V(\lambda)$ es la llamada función de eficiencia luminosa relativa del sistema visual empleado en la percepción de la intensidad de luz. Por lo que la luminancia de un objeto es independiente de las luminancias de objetos circunvecinos.

C) Brillantez.

Es la cantidad de luz o luminancia percibida por un sistema visual, y depende de las luminancias de los objetos contenidos en la escena. De este modo se pueden tener dos objetos con idénticas luminancias pero si el entorno en ambos objetos es diferente la brillantez será diferente.

D) Contraste.

El contraste en una imagen es una medida de la variación de los tonos de gris en dicha imagen. Un bajo contraste indica un estrecho rango de niveles de gris, en cambio un alto contraste señala un amplio número de tonos de gris.

E) Bordes.

Un borde es la frontera entre dos regiones de diferente tono de gris, además de ser la primera información que se tiene en la percepción de imágenes. En una figura cuadrada se tiene cuatro bordes unidos a cuatro esquinas, mientras que un triángulo es una figura de tres bordes conectados en sus vértices. La medición del largo de un objeto en una imagen es lograda por la localización de dos bordes finales y contabilizando los pixeles entre ellos. El problema es que la información del borde puede variar por el ruido de la imagen, siendo una tarea importante en los SPDI el procesamiento de la imagen para lograr el realce de dichos bordes.

Los parámetros obtenidos por técnicas matemáticas y empleados para caracterizar un objeto en base a sus bordes, son:

- Perímetro, es el número de pixeles que rodean un objeto.
- Pixeles Totales, es el número de pixeles contenidos dentro del perímetro.
- Bordes del rectángulo, son los límites de una área o región.
- Redondez, medición esférica de un objeto o región.
- Espacios totales, conteo del número de huecos o regiones espaciales.
- Centro de Gravedad, localización del promedio de mayor peso o centro del área.
- Primer Momento, momentos con respecto a ejes dados, un momento es el valor del pixel por la distancia de un punto sobre el eje.
- Segundo Momento, el cuadrado del primer momento dividido por el número total de pixeles en el área.
- Largo y Ancho, dimensión del objeto en una dirección específica.

La relevancia de cada parámetro para la identificación de un objeto debe ser determinada por el diseñador del sistema. La varianza en un número de mediciones de similares objetos nos da una idea del valor de los parámetros en las tareas de identificación.

II.4.4 Principales áreas del Procesamiento Digital de Imágenes.

EL PDI es aplicado en diferentes áreas, como realce de imágenes, restauración de imágenes, análisis de imágenes, reconstrucción de imágenes por proyecciones y compresión de imágenes. En cada una de éstas se manipula la información digital de las imágenes por medio de algoritmos matemáticos para obtener resultados que ayuden a la solución de necesidades y/o problemas.

Realce de Imágenes.

Las técnicas de realce de imágenes buscan acentuar ciertas características de la imagen para el subsecuente análisis o despliegue de la misma. Ejemplos donde se efectúa un realce de las características incluyen al contraste, bordes, pseudocoloramiento, filtrado de ruido, etc. Mejorar una imagen no incrementa la información visual contenida en los datos, solamente hace énfasis en ciertas características específicas de la imagen.

Restauración de Imágenes.

La restauración de imágenes se refiere a renovar o minimizar degradaciones conocidas en una imagen. Dentro de las restauraciones se incluyen las imágenes degradadas por:

a) Las limitaciones físicas del sensor de adquisición.

b) El proceso de filtración de ruido.

c) La distorsión geométrica o las no-linealidades debidas a los sensores.

Análisis de Imágenes.

El análisis de imágenes relaciona las mediciones cuantitativas hechas de una imagen y que producen una descripción de ésta. Las técnicas de análisis ayudan a tomar decisiones más sofisticadas en comparación a las determinadas por el Sistema de Visión Humano, como lo es el control de un brazo de un robot, mover un objeto después de identificar a éste o controlar la navegación de un aeroplano.

Para llevar a cabo un análisis es necesario involucrar tres estudios básicos:

a) Extracción de características.

En un SPDI la imagen restaurada o mejorada es preprocesada para obtener la mayor cantidad de propiedades contenidas en los datos.

b) Segmentación.

Es la descomposición de una imagen en grupos de datos que tienen ciertas características; por ejemplo, para extraer de una imagen los objetos contenidos en ella se puede recurrir a la detección de bordes o texturas.

c) Empleo de Técnicas de Clasificación.

La mayor tarea después de extraer los rasgos (sea en regiones o segmentos) de una imagen, es su clasificación dentro de una gran variedad de objetos identificados previamente por una etiqueta. Cuando se determina la relación existente entre los diferentes objetos que describen la imagen de acuerdo a la capaciadad de descripción del sistema digital, se pueden generar decisiones.

Reconstrucción de Imágenes por Proyecciones.

La reconstrucción de Imágenes por proyecciones es un problema particular de la restauración de imágenes, sin embargo, en este caso un objeto bidimensional es reconstruido por varias proyecciones unidimensionales. Por ejemplo, para obtener la imagen de un órgano humano, se emplean rayos X u otro tipo de radiación en la proyección de dicho órgano. Las proyecciones planas son obtenidas con diferentes ángulos de rotación en un mismo eje con respecto a nuestro objetivo, y la reconstrucción de la imagen se realiza por métodos algorítmicos.

Compresión de Imágenes.

La cantidad de datos asociados con la información visual es muy extensa y requiere de dispositivos de gran capacidad de almacenamiento. En la imágenes típicas de TV la información manipulada excede los 10 millones de bytes por segundo, por lo que para transmitir y/o almacenar esta información se requiere de una gran capacidad y/o ancho de banda de los dispositivos, lo cual resulta demasiado costoso. Las técnicas de compresión de imágenes están relacionadas con la reducción del número de bits requeridos para almacenar o transmitir imágenes, con un costo implícito en la degradación de la información.

II.5 TIPOS DE OPERADORES PARA EL REALCE DE IMAGENES.

El realce de una imagen (dirigido a acentuar o perfeccionar las características de bordes, límites o contraste), persigue hacer un mejor despliegue gráfico y análisis de rasgos.

Fundamentalmente, la generación de un nuevo valor para un pixel puede ser en función de un valor de pixel o de varios valores de pixeles vecinos. Los tipos de operadores se dividen en dos básicamente:

a) Puntuales (modificación de datos de acuerdo a una escala general).

b) Dinámicos (determinación de una imagen matricial en base a una o varias imágenes).

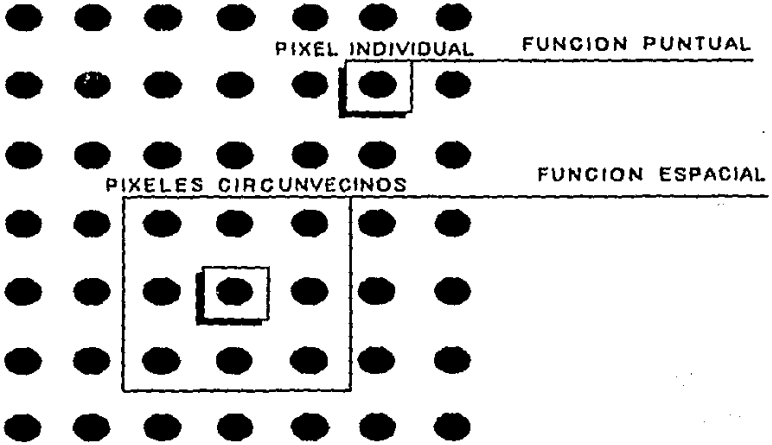


Figura 2.13 Funciones puntuales y espaciales.

II.5.1 Operaciones Puntuales.

Los operadores puntuales son los más simples o elementales procesos de operación en una imagen. Si se tiene una imagen de entrada P como se muestra en la figura 2.14, al aplicar

una operación puntual obtenemos la imagen de salida Q. Dentro de las operaciones puntuales se encuentran los operadores identidad, inversor, umbral y las combinaciones entre ellos.

La función general para transformar el valor de la intensidad de luz o tono de gris de cada pixel en una imagen, está dada por la expresión:

$$Q(i,j) = f [P(i,j)]$$

El término f puede ser un operador lineal o no lineal. Los procesos matemáticos son tan simples como obtener el producto entre los valores de una imagen matricial por una constante.

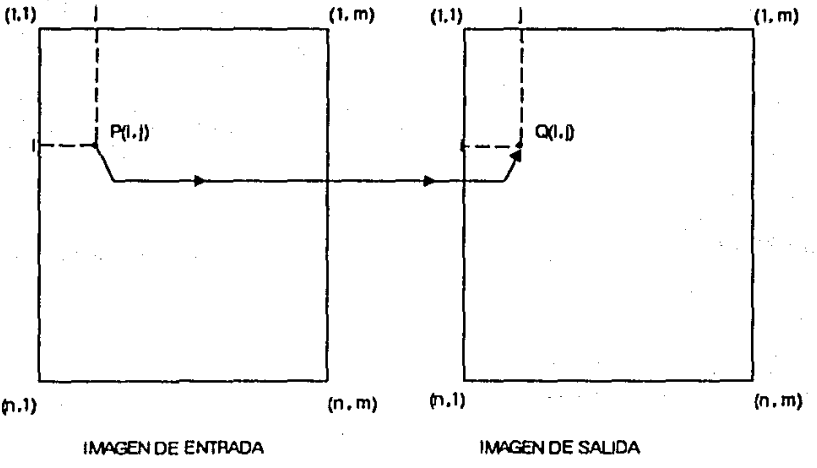


Figura 2.14 Operaciones Puntuales.

II.5.1.1 Operador Identidad.

Este operador obtiene una imagen de salida igual a la imagen de entrada. Los valores de cada pixel en la imagen resultado Q son idénticos a los valores correspondientes a cada pixel de la imagen fuente P. La función f(P) es una línea continua iniciando en el origen y extendiéndose al valor de intensidad de luz máximo posible del pixel, Fig.2.15.

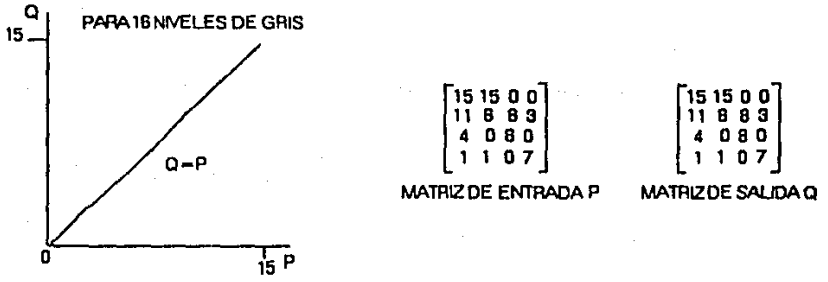


Figura 2.15 Operador identidad.

II.5.1.2 Operador Inversor.

Aplicar este operador implica obtener una imagen inversa en comparación a la imagen de entrada. La función f(P) es una línea donde el valor máximo de tono de gris esta ubicado en el mínimo valor de la escala de entrada y es igual a cero en el valor máximo de la misma escala de entrada (Fig.2.16).

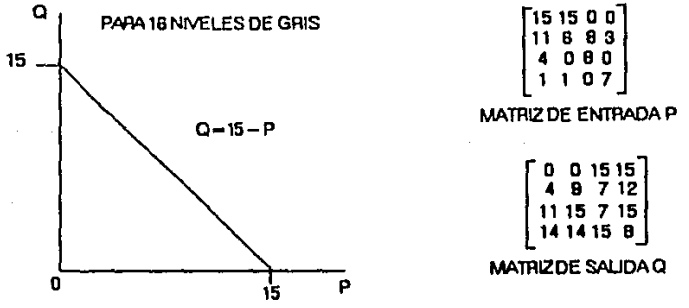


Figura 2.16 Operador inversor.

II.5.1.3 Operador de Umbral.

Esta clase de operador al aplicarse a una imagen de entrada con varios tonos de gris proporciona una imagen de salida con sólo dos tonos de gris. Para determinar los valores de los pixeles de salida se requiere un parámetro como nivel de decisión (P1), conocido como

umbral. Donde todos los valores menores o iguales de P1 son convertidos a cero y los valores mayores a P1 se les asigna el máximo valor, la función de transferencia se tiene en la siguiente figura.

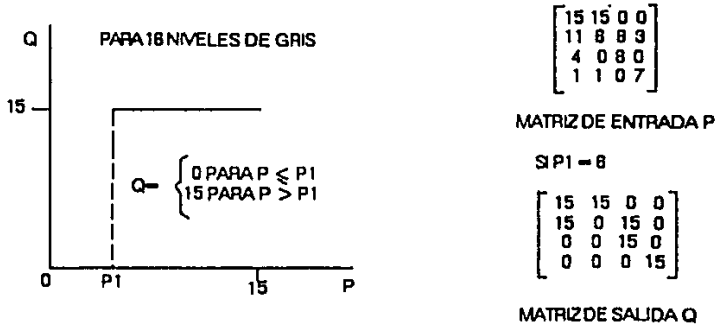


Figura 2.17 Operador de Umbral.

II.5.1.4 Otros Operadores.

Existen otros operadores derivados de las combinaciones de los operadores básicos: identidad, inversor o umbral (Fig.2.18.a).

A) Operador Inversor-Umbral: La salida de la imagen es invertida por la aplicación de una función inversora, además de tener un umbral para fijar el valor final (Fig.2.18.c).

B) Intervalo de Umbral-Binario: Esta clase de operador obtiene una imagen de salida binaria (solo dos valores posibles) donde todos los valores de tonos de grises de la imagen de entrada localizados dentro del intervalo P_1 a P_2 incluyendo ambos puntos, son convertidos al máximo valor de los tonos y los valores fuera de este intervalo toman el valor mínimo.

C) Operador Umbral-Binario-Inversor: Este operador puede ser usado para convertir una escala de tonos multinivel a una imagen binaria, y además invertir los valores finales (Fig.2.18.b).

D) Operador Umbral en la Escala de Tonos: Este operador es mostrado en la figura 2.18.d, donde se tiene una gama de grises en la imagen de entrada. El intervalo limitado por P_1 y P_2 , es el umbral que determina la imagen de salida, la cual tendrá los mismos valores de los pixeles de entrada si se encuentran dentro del intervalo y valores de cero para los pixeles cuyo valor de entrada estén fuera de dicho intervalo.

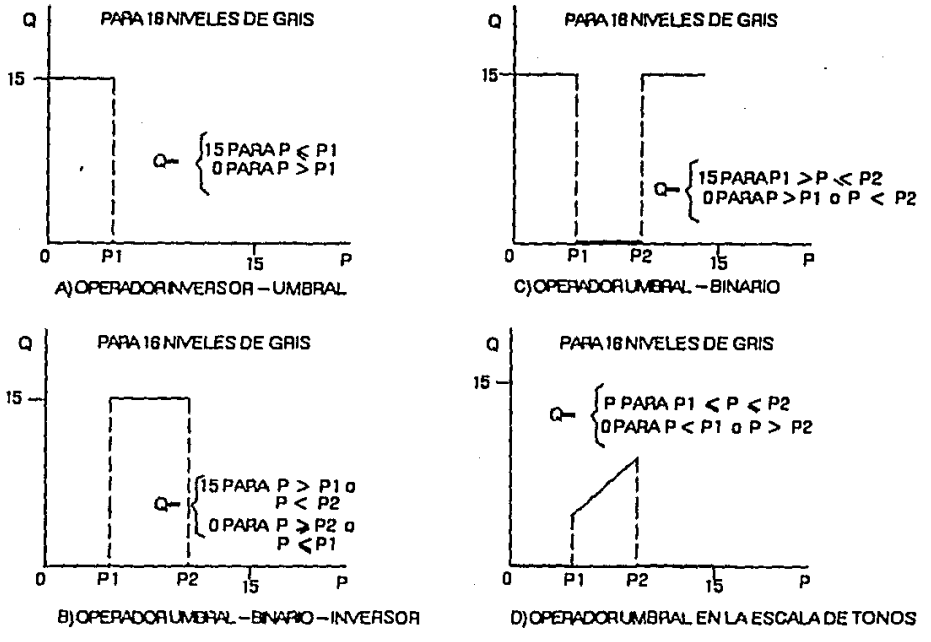


Figura 2.18 Operadores a)Inversor-Umbral, b)Intervalo-Binario, c) Umbral-Binario-Inversor y d) Umbral en la escala de Tonos.

E) Operador Extensión: Esta clase muestra una imagen de salida, donde los valores de los pixeles de la imagen de entrada que estén fuera del intervalo definido por P_1 y P_2 son suprimidos y los valores dentro del intervalo toman un nuevo valor empleando toda la escala del sistema de procesamiento (Fig.2.19.a).

F) Operador Reductor de Escala de Tonos: La imagen de salida es el resultado de la asignación de varios intervalos de tonos de gris y cada uno representa un número finito de valores de pixeles de entrada. De esta forma se reducen los niveles de grises de la imagen de entrada (Fig.2.19.b).

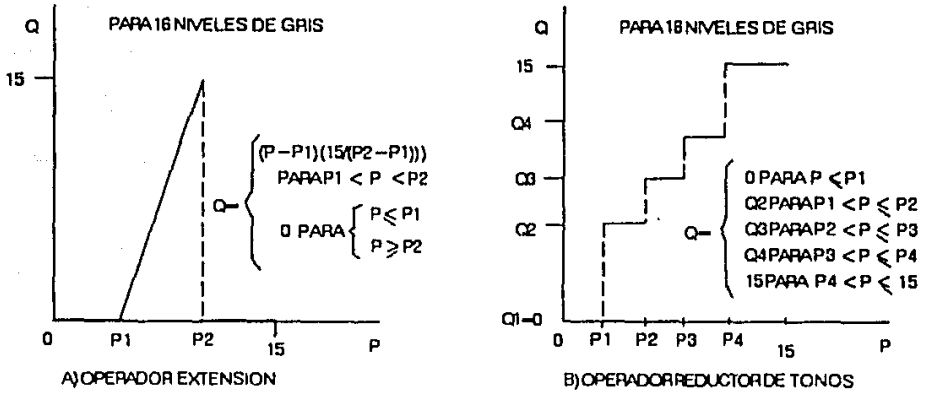


Figura 2.19 Operadores a) Extensión y b) Reductor de Tonos

II.5.2 Operaciones Dinámicas.

Los operadores dinámicos, empleados en el procesamiento, generan una nueva imagen mediante dos formas:

- Operaciones de Punto Múltiple: Donde el valor de cada pixel en el arreglo depende de la combinación de los valores correspondientes a la localidad del pixel en dos o más cuadros de imágenes similares.

- Operaciones espaciales: Los valores de los pixeles en la región adyacente a la localidad del pixel son combinados y determinan su valor. Estas operaciones tienen rasgos propios de la Convolución Discreta, misma que se explica en Operaciones Espaciales.

II.5.2.1 Operaciones de punto múltiple.

Un operador dinámico determina un nuevo valor, empleando la información contenida en la misma localidad en dos imágenes. En la figura 2.20, se muestran dos imágenes matriciales A y B, mediante una operación punto múltiple obtenemos la imagen resultado C. El tamaño de la matriz no cambia y la función de transformación puede ser lineal o no lineal. La relación de

transformación es aplicada a los pares de pixeles de las imágenes de entrada con la misma localidad, esto se expresa mediante la ecuación :

$$C_{i,j} = F_d (A_{i,j} , B_{i,j})$$

donde F_d es una función de dos variables y un rango definido por i y j desde 0 a M y N, respectivamente. La función del operador F_d puede ser suma, resta, multiplicación, división, exponenciación o alguna función propuesta.

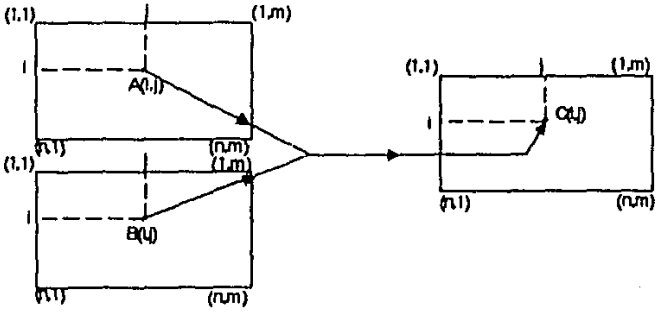


Figura 2.20 Operaciones de punto múltiple.

La función puede tener un apropiado factor de escala K para obtener la magnitud de los valores de salida dentro del rango de la escala para evitar un sobreflujo o condición negativa. La transformación involucra dos variables asociadas con los pares de puntos correspondientes y se representa mediante la siguiente relación:

$$R(i,j) = F[P(i,j),Q(i,j)]$$

donde P y Q son las matrices de entrada, F es el operador y R es la matriz de salida.

A) Adición de Imágenes: La adición de imágenes puede ser empleada para reducir los efectos del ruido contenido en los datos, como se muestra en la figura 2.21. El valor de la salida $C_{i,j}$ es dada por :

$$C_{i,j} = (A_{i,j} + B_{i,j}) / k$$

dentro del rango de valores i,j donde k es igual al número de imágenes. El proceso de adición es promediar dos imágenes matriciales de entrada y obtener una imagen en base al resultado del promedio. Si una de las imágenes es una constante, el resultado será una luminosidad u obscurecimiento sobre toda la imagen y su histograma presentará un corrimiento. Al emplear

un mayor número de imágenes para realizar la adición, implica aumentar la calidad y reducir el ruido en la imagen resultado.

B) Substracción de imágenes: Substraer imágenes ayuda a detectar cambios dentro en una imagen, siendo condición necesaria que las imágenes correspondan a una misma escena pero adquiridas en tiempos diferentes. Es necesario emplear números positivos durante el proceso o involucrar una reasignación en la escala de tonos, con el fin de poder definir la imagen de salida. La relación es dada por:

$$C_{i,j} = K1(A_{i,j} - B_{i,j})$$

donde K1 es una función constante, la cual garantiza que $C_{i,j}$ tenga un valor mínimo igual a cero y el máximo es 255.

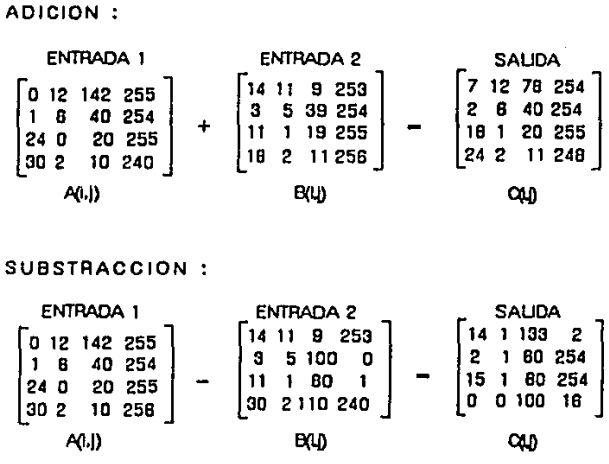


Figura 2.21 Adición y substracción de imágenes.

C) Multiplicación de imágenes: La multiplicación de dos matrices en el procesamiento de imágenes es mostrada en la figura 2.22. Esta operación de multiplicación puede ser empleada para corregir las degradaciones debidas a las secciones no-lineales del sensor de adquisición y su corrección es posible mediante la multiplicación de la imagen matricial por una matriz de corrección. La siguiente ecuación es una manera general de expresar lo antes descrito :

$$C_{i,j} = K2 [(A_{i,j} \times B_{i,j}) + A_{i,j}]$$

donde todos los valores $C_{i,j}$ son redondeados a el próximo entero, el máximo valor es 255 y $B_{i,j}$ es el factor de corrección.

Otro uso del operador multiplicación podría ser la creación de una ventana dentro de la imagen para reducir el área específica de interés. La matriz de resultado es dada por la relación:

$$C_{i,j} = A_{i,j} \times B_{i,j}$$

donde $B_{i,j}$ es igual 1 para todos los pares que estén dentro de la ventana y en el caso de estar fuera del área de interés su valor es igual a cero.

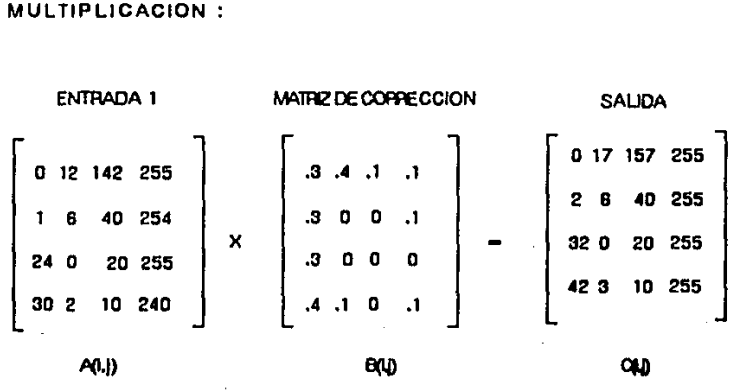


Figura 2.22 Multiplicación de imágenes.

II.5.2.2 Operaciones espaciales.

La mayoría de las técnicas de realce en imágenes son basadas en operaciones espaciales dentro de una zona vecina en los pixeles de entrada. La imagen comúnmente es convolucionada con la respuesta impulso finita, llamada máscara espacial.

II.5.2.2.1 Convolución.

Un sistema lineal puede tener una expresión general que relacione su señal de salida con una de entrada, como la siguiente:

$$y(t) = \int_{-\infty}^{+\infty} f(t,\tau)\, x(\tau)\, d\tau$$

donde a través de una serie de corrimientos, cambios y adición de constantes, se obtiene :

$$y(t) = \int_{-\infty}^{+\infty} g(t-\tau)\, x(\tau)\, d\tau$$

Esta es la integral de convolución, donde la salida de un sistema lineal invariante es dada por la convolución de la señal de entrada x(t) con g(t), que es una función característica del sistema llamada respuesta impulso finita.

La convolución en un sistema bidimensional es dada por la expresión:

$$h(x,y) = f^{*}g = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(u,v)\, g(x-u,y-v)\, du\, dv$$

Al realizar la convolución, las funciones son multiplicadas y el producto es integrado sobre las dos dimensiones.

En el caso de una convolución digital para una imagen de representacion matricial, se tiene:

$$Q(i,j) = P^{*}F = \sum_{m}\sum_{n} P(m,n)\, F(i-m,j-n)$$

donde P(m,n) es la función discreta representativa de la imagen matricial, y F(i-m,j-n) es una función de sensitividad espacial, (Fig. 2.23). Los valores de Q(i,j), que es la convolución de P y F, son el promedio local de la región convolucionada.

Este tipo de convolución es válida solamente cuando la imagen es finita en la región de convolución, esto es, tener P y F diferentes de cero en un dominio finito.

El proceso de convolución de una imagen es la acción de comparar un grupo de pixeles de acuerdo una referencia dada, con cada pixel que forma la imagen.

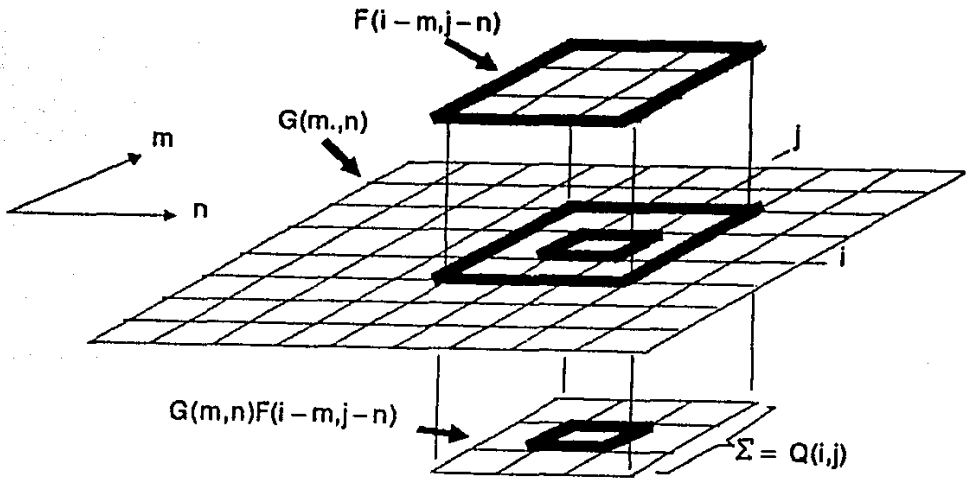


Figura 2.23 Convolución de una imagen matricial.

El desarrollo de una convolución entre dos matrices P y F (Fig.2.24), implica realizar una multiplicación entre todos sus elementos correspondientes. La suma de los productos de los elementos son multiplicados por una constante de normalización para dar el valor del pixel resultado, mismo que es colocado en una localidad correspondiente del nuevo arreglo Q. El procedimiento es repetido después de realizar el corrimiento de las coordenadas de los elementos en una u otra dirección hasta obtener todas las combinaciones del arreglo.

El valor del pixel en la matriz de salida es calculado mediante la convolución de los datos de la $P_{i,j}$, con la matriz de coeficientes $F_{i,j}$ y el resultado es el valor del pixel central del área de convolución $Q_{2,2}$, dado por la relación :

$$Q_{2,2} = (P_{1,1}xF_{1,1}) + (P_{2,1}xF_{2,1}) + \ldots + (P_{3,3}xF_{3,3})$$

$$P(i,j) \quad F(i,j) \quad Q(i,j)$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \ast \begin{bmatrix} +1 & +1 & -1 \\ -1 & +1 & +1 \\ +1 & +1 & -1 \end{bmatrix} = \begin{bmatrix} & & \\ & e' & \\ & & \end{bmatrix}$$

$$Q(2,2) = e' = +a + b - c - d + e + f + g + h - i$$

Figura 2.24 Convolución entre matrices.

La convolución puede involucrar una matriz mayor (4x4, 5x5, etc.), siendo lo más común una matriz de 3x3 que reporta un menor tiempo y capacidad de procesamiento en comparación a una matriz de mayor extensión. Con una matriz de 3x3, se emplean nueve valores de pixeles para dar un valor al pixel central de la matriz. Una vez calculado un valor, la localidad es corrida por uno y el proceso se repite hasta generar la imagen matricial por completo. Los valores de los bordes de la imagen matricial asumen un valor constante.

II.5.2.2.2 Filtro digital.

Los elementos claves en el PDI son la habilidad para manipular sistemáticamente los datos básicos en un perfodo de tiempo aceptable y lograr resultados confiables. Con el fin de obtener las características más relevantes de los parámetros a considerar durante el análisis de imágenes, la tarea de un operador espacial es rescatar en dicha imagen la información relevante en un tiempo razonable. El operador espacial puede ser diseñado para hacer diferentes transformaciones de imágenes, sin embargo la operación esencial es referida como un filtro digital.

Un filtro digital electrónico puede ser catalogado como un filtro paso bajas o un paso altas dependiendo en cada caso del espectro de frecuencia. En la figura 2.25.a se muestra la señal de salida de un Filtro Paso Altas (FPA), formada por componentes de alta frecuencia. En cambio la figura 2.25.b presenta un Filtro Paso Bajas (FPB) donde la señal de salida la forman la componente de DC y las componentes de baja frecuencia.

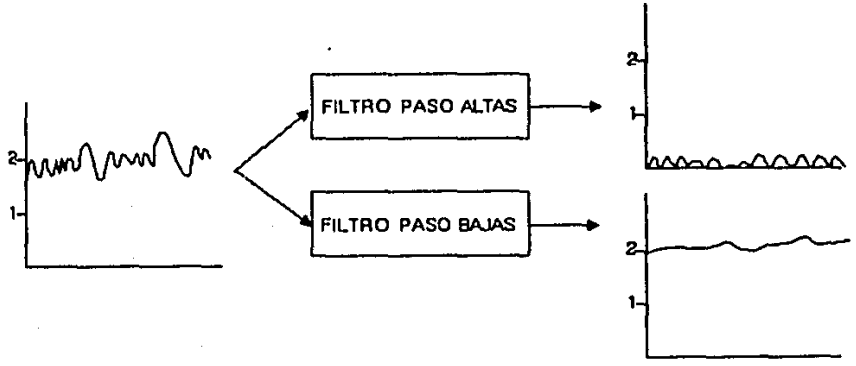


Figura 2.25 Filtros a)Paso altas, b)Paso bajas.

En una imagen las componentes de alta frecuencia se caracterizan por un cambio brusco en el contraste, en comparación a los pixeles adyacentes. En relación a las componentes de baja frecuencia, estas tienen una tenue modificación en el contraste dentro de un área determinada (Fig.2.26).

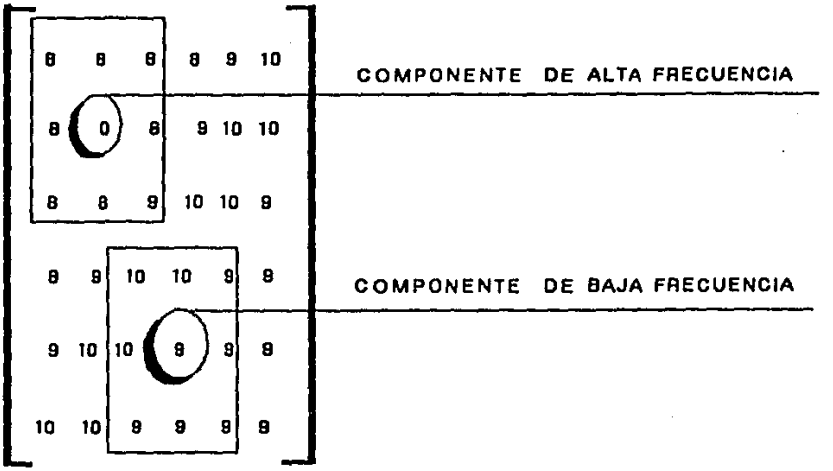


Figura 2.26 Componentes de Alta y Baja frecuencia en una imagen

A) Filtro paso bajas (FPB).

Los puntos aislados aleatorios en una imagen pueden ser considerados como ruido y son por regla general de alta frecuencia, reflejándose en el cambio brusco de los valores correspondientes a los pixeles adyacentes a un determinado pixel. El efecto ocasionado por estos puntos es reducido por medio del uso de un simple filtro promediador.

El FPB no afecta las componentes de baja frecuencia en los datos de una imagen y solo atenúa las componentes de alta frecuencia. Para la implementación de los filtros, se emplea la convolución matricial. Los nueve coeficientes del arreglo matricial (3x3) de un filtro promediador, utilizado en la convolución, son presentados en la siguiente figura.

$$\begin{bmatrix} F_{1,1} & F_{1,2} & F_{1,3} \\ F_{2,1} & F_{2,2} & F_{2,3} \\ F_{3,1} & F_{3,2} & F_{3,3} \end{bmatrix} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Figura 2.27 Coeficientes de la matriz del FPB.

Las características de un FPB son: los coeficientes deben ser positivos y la suma de todos los coeficientes igual a uno. Si la suma de los coeficientes es mayor de uno, el resultado es una amplificación; y una suma menor a uno provoca una atenuación [Galbiati, 1990].

La aplicación de los coeficientes de una matriz de 3x3 a un campo de datos de igual valor, no implicará la modificación de la información, excepto en donde exista un gradiente entre los datos adyacentes.

El efecto del filtro digital paso bajas es la reducción del ruido, ilustrado en la figura 2.28, el pixel de valor cero representa una alteración de la imagen al ser adquirida en cambio, los pixeles vecinos tienen igual valor. Una vez filtrada la información el dato es modificado por un valor promedio.

$$\begin{bmatrix} 7 & 7 & 7 & 7 \\ 7 & (0) & 7 & 7 \\ 7 & 7 & 7 & 7 \end{bmatrix} = \begin{bmatrix} 7 & 7 & 7 & 7 \\ 7 & (7) & 7 & 7 \\ 7 & 7 & 7 & 7 \end{bmatrix}$$

MATRIZ CON UN ELEMENTO CON RUIDO

MATRIZ DESPUES DE APLICAR UN FILTRO PASO BAJAS

Figura 2.28 Resultados de aplicar un FPB.

En la tabla 2.2 se observa que al incrementar el área de la matriz de convolución, donde se considera un mayor número de pixeles se incrementa el costo y tiempo de proceso. Por este motivo el arreglo de 3x3 es comúnmente empleado en la industria, donde lo importante es reducir costos y obtener resultados dentro de una tolerancia de error aceptable.

MATRIZ DE NxN	VALORES EMPLEADOS EN LA CONVOLUCION	TIEMPO DE PROCESO EN RELACION A UNA MATRIZ DE 3x3
3x3	9	1
5x5	25	2.7
9x9	81	9

Tabla 2.2 Comparación de diferentes área de matrices.

B) Filtro Paso Altas.

Un filtro de esta clase no cambia las componentes de alta frecuencia y sólo atenúa las de baja frecuencia, como lo son las componentes de una mínima diferencia en torno a una área específica. El efecto de aplicar un FPA a una imagen de 6x6 datos, se muestra en el histograma (Fig.2.29) Los valores máximos de los pixeles pueden o no cambiar dependiendo del grado de saturación de la imagen.

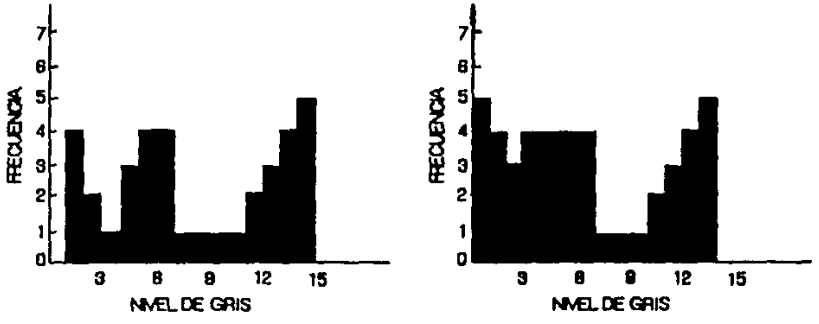


Figura 2.29 Histograma del efecto de un FPA.

En el caso de un filtro paso altas, la matriz de coeficientes para la convolución de 3x3 está dada por :

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figura 2.30 Matriz de coeficientes de un FPA.

donde los coeficientes pueden ser negativos o positivos, siendo la suma total de éstos igual a cero. En el caso de obtener un resultado igual a 1, el valor de las componentes de baja frecuencia podrían tener el mismo valor de la señal original. El siguiente ejemplo muestra el proceso de obtención de los valores para diferentes matrices de datos, empleando la matriz de coeficientes de la figura 2.30.

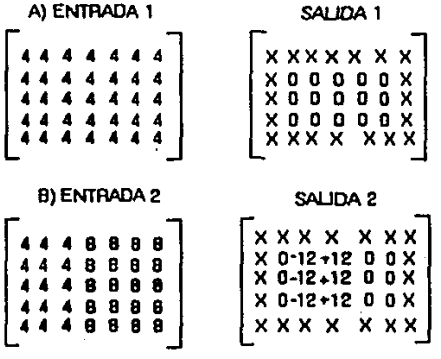


Figura 2.31 Ejemplo de aplicación de un FPA.

EL borde de una imagen matricial es apreciado por una cambio entre los valores de los pixeles adyacentes. La información relativa a la localidad y magnitud del borde es contenida o representada por altas frecuencias de los datos de la imagen. El filtro elimina el valor

constante (Fig.2.31.a) y determina el borde de separación, entre dos valores diferentes (Fig.2.31.b).

La detección de bordes es básico para poder estudiar una imagen, debido a que la información que contienen es relativa a los conceptos de contraste, localización, contorno y determinación de dimensiones, existiendo diferentes tipos de filtros paso altas para mejoramiento o detección de bordes.

II.5.2.2.3 Operador Laplaciano.

El operador Laplaciano empleado en el PDI está basado en la representación de la segunda derivada parcial de las funciones continuas en dos dimensiones:

$$\nabla^2 F = \frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial Y^2}$$

Esta expresión matemática continua puede ser aproximada mediante el operador diferencial discreto. La insensibilidad rotacional del operador laplaciano es esencial para determinar el cambio de gradiente de la intensidad en el pixel en las direcciones X y Y.

$$L(i,j) = \nabla^2 p(i,j) = \Delta x^2\, p(i,j) + \Delta y^2\, p(i,j)$$

donde :

$$\Delta x^2 = [\, p(i-1,j) - p(i,j)] - [p(i,j) - p(i+1,j)]$$
$$\Delta y^2 = [\, p(i,j+1) - p(i,j)] - [p(i,j) - p(i,j-1)]$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad \Delta x^2 = [d - e] - [e - f] \qquad \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad \Delta y^2 = [h - e] - [e - b]$$

El resultado del operador laplaciano es el siguiente :

$$L(i,j) = b + d + f + h - 4e$$

y es reducido a la matriz de coeficientes :

$$\begin{bmatrix} 0 & +1 & 0 \\ +1 & -4 & +1 \\ 0 & +1 & 0 \end{bmatrix}$$

En resumen, el operador laplaciano calcula la diferencia entre el tono de gris de un pixel central y el promedio de los tonos de gris de los cuatro pixeles adyacentes en las direcciones vertical y horizontal, teniendo características de un FPA. La aplicación del operador laplaciano en las matrices de entrada de la figura 2.32 muestra la detección de borde vertical e inclinado.

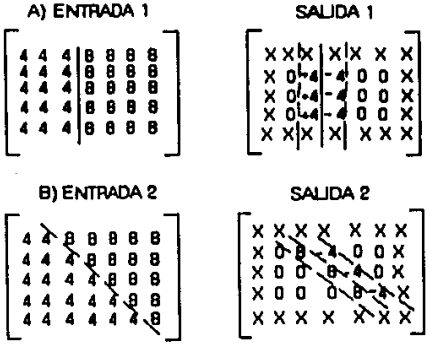


Figura 2.32 Resultados del operador Laplaciano.

II.5.2.2.4 Operador Gradiente de Robert.

El operador gradiente de Robert (Fig.2.33), es más simple que el operador Laplaciano, porque éste opera en una región de 2x2 de pixeles para cada punto. El empleo de un menor número de datos reduce el tiempo de proceso y capacidad del sistema. Este operador usa derivadas diagonales para estimar el gradiente de un punto.

La magnitud del operador es igual a la raíz cuadrada de la suma de los cuadrados de las dos diferencias diagonales. Una aproximación de dicho operador es el cálculo más simple, la suma del valor absoluto de cada diferencia de las diagonales.

$$\text{Magnitud del Operador} = (\Delta 1^2 + \Delta 2^2)^{1/2}$$

$$\text{Valor Absoluto estimado} = |\Delta 1| + |\Delta 2|$$

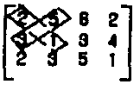
donde:

$$\Delta 1 = p(i,j) - p(i+1,j+1)$$
$$\Delta 2 = p(i+1,j) - p(i,j+1)$$

Q(1,1) = 1 + 2

DONDE: 1 = (a − d) Y 2 = (b − c)

MATRIZ DE ENTRADA



Q(1,1) = (2 − 1) + (5 − 3) = 1 + 2 = 3

MATRIZ DE SALIDA Q

$$\begin{bmatrix} 3 & 7 & 3 & X \\ 1 & 4 & 3 & X \\ X & X & X & X \end{bmatrix}$$

Figura 2.33 Operador Gradiente de Robert.

II.5.2.2.5 Operador detector de borde Sobel.

El operador de Sobel mostrado en la figura 2.34, es un operador que combina la detección de borde vertical y horizontal, el valor del pixel central de la matriz del detector Sobel es cero, lo que origina un valor nulo al efectuar la multiplicación, durante la convolución de la imagen matricial de entrada, en el punto correspondiente al pixel a calcular en la matriz de salida. Por lo que se dice, que el punto de la matriz de entrada no es considerado para determinar el de salida.

$$S = (\Delta x^2 + \Delta y^2)^{1/2} \quad .$$

ENTRADA

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\Delta x = (a + 2d + g) - (c + 2f + e)$$

$$\Delta y = (g + 2h + i) - (a + 2b + c)$$

DONDE:

$$\Delta x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \Delta y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

DETECTOR DE BORDES DE SOBEL $\quad \Delta x + \Delta y = \begin{bmatrix} 0 & -2 & -2 \\ 2 & 0 & -2 \\ 2 & 2 & 0 \end{bmatrix}$

Figura 2.34 Operador de Sobel.

II.5.2.2.6 Otros operadores.

Los métodos de realce presentados en las secciones anteriores son muy básicos, sin embargo son los más comunes en el PDI comercial. En la práctica es necesario experimentar con varios filtros con el fin de obtener los mejores resultados adecuados a la aplicación. La máscara para efectuar la convolución tiene un número variado de coeficientes, los cuales están relacionados en la siguiente tabla.

Tabla 2.3

1. FILTROS PASO BAJAS (COEFICIENTES POSITIVOS Y SU SUMA IGUAL A UNO)	$\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$	$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$
2. FILTROS PASO ALTAS (SUMA DE COEFIENTES IGUAL A CERO)	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$
3. FILTROS PASO ALTAS CON COMPONENTE DE DC (SUMA DE COEFIENTES IGUAL A UNO)	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$
4. GRADIENTE DIRECCIONAL (SUMA DE COEFIENTES IGUAL A CERO)	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & -1 & -1 \end{bmatrix}$ N	$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{bmatrix}$ NE	$\begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix}$ E
5. FILTROS DIFERENCIADORES (SUMA DE COEFIENTES IGUAL A CERO)	$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ BORDES VERTICALES	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ BORDES HORIZONTALES	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ BORDES VERTICALES Y HORIZONTALES

6. TRAZOS CONFUSOS

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{HORIZONTAL}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{VERTICAL}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{DIAGONAL}$$

7. FILTROS DIFERENCIADORES

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{VERTICAL}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{HORIZONTAL}$$

8. FILTROS DE DIFERENCIACION VERTICAL

TOMAR EL VALOR ABSOLUTO DE LOS COEFIENTES DEL INCISO 7

9. FILTRO DE DIFERENCIACION HORIZONTAL Y UNIFORMIZACION VERTICAL

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

10. FILTROS LAPLACIANOS (SUMA DE COEFICIENTES IGUAL A CERO)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

11. EXPANSION A UNA REGION BRILLANTE

MAXIMA

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

12. FILTRO INTERMEDIO (REDUCE EL RUIDO DE CAMARA)

LA MISMA DEL INCISO 11

13. REALCE DE SEGMENTOS DE LINEA (SUMA DE COEFIENTES IGUAL A CERO)

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \quad \text{VERTICAL}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{HORIZONTAL}$$

$$\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \quad \text{L-R DIAGONAL}$$

III. PROCESAMIENTO EN PARALELO

III.1. INTRODUCCION

Desde la introducción de las computadoras digitales, la tecnología se ha enfocado a la búsqueda de altas velocidades de procesamiento, siendo un obstáculo importante a vencer la llamada máquina Von Neumann. En un computador secuencial, una simple memoria buffer sirve como enlace entre una memoria de alta velocidad y la unidad central de procesamiento. Esto hace necesario organizar todas las tareas en un estricto orden secuencial, haciendo que el cálculo de las tareas consuma más tiempo.

Dos grandes innovaciones en el diseño de hardware de computadoras digitales ha permitido superar las restricciones de la máquina Von Neumann, alcanzándose mayores velocidades de procesamiento: el paralelismo y el pipeline. Con la implementación de estas técnicas, han surgido distintas familias de computadoras de alta velocidad, supercomputadoras y arreglos de procesadores.

El objetivo actual del diseño de sistemas digitales, en su organización y software, es la creación de sistemas que permitan el paralelismo, la concurrencia o la ejecución simultánea de tareas. La concurrencia es la realización simultánea de eventos en el mismo intervalo de tiempo alcanzado varios grados dependiendo del tipo de operación que el sistema ejecute, pudiendo ser:

- Ejecución concurrente de diferentes programas del usuario.
- Operaciones simultáneas entrada/salida (E/S) con la ejecución de programas del usuario.
- Operaciones múltiples E/S para la comunicación de datos, ejecutadas simultáneamente.
- Operaciones en sistemas distribuídos.
- Concurrencia de operaciones en el procesador central en general.

Estos logros sólo han sido posibles con la implementación del paralelismo y de la estructura pipeline.

III.2. GENERALIDADES

III.2.1 La Máquina Von Neumann

La organización más simple de una computadora o una unidad de proceso, es la clásica máquina de Von Neumann. Esta arquitectura busca una instrucción a través del contador de programa (CP), ejecuta la instrucción y luego vuelve a leer la siguiente instrucción:

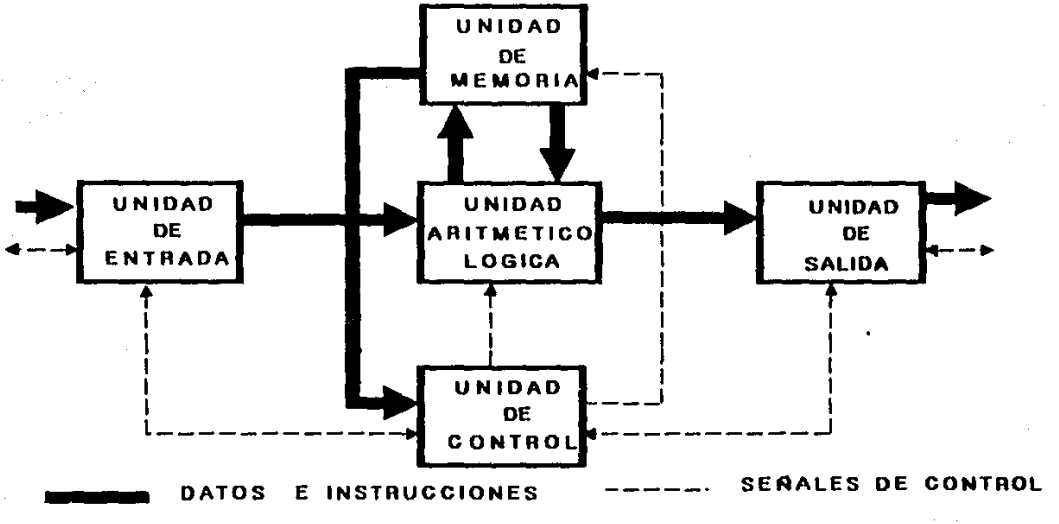


Figura 3.1 Máquina básica Von Neumann

La mayor debilidad en este diseño es lo rutinario de todas las operaciones de (E/S) que se efectúan a través de la unidad aritmético lógica (ALU), es decir, que detiene el proceso en ejecución mientras que se está haciendo alguna operación de E/S.

III.2.2 Acceso directo de memoria (DMA)

En este tipo de arquitectura, la ruta de transferencia E/S es alterada al proveer un acceso directo a memoria. Esta arquitectura libera a la unidad aritmético lógica proveyendo una mayor versatilidad, sin embargo, la unidad de control aún es la encargada de las operaciones de E/S en ejecución.

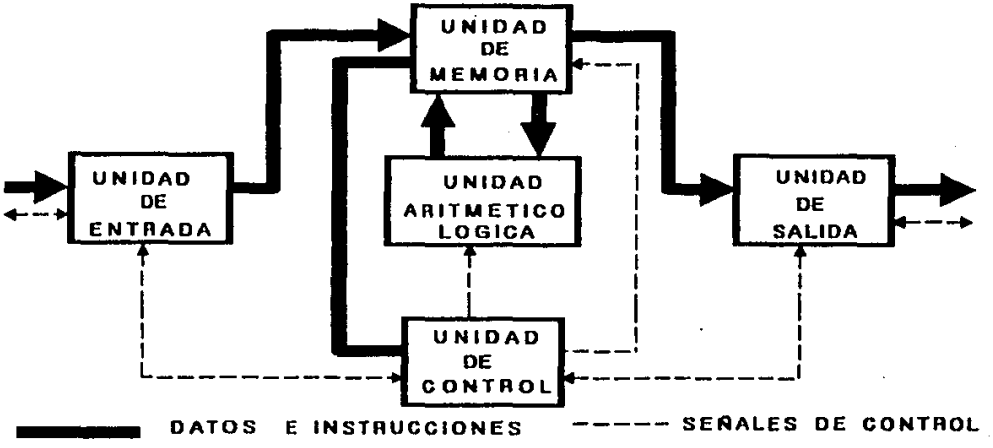


Figura 3.2

III.2.3 Canal de entrada/salida

El uso del canal E/s provee una ruta de acceso separada, hacia y de la memoria agregada al control de operaciones E/S. El canal es un procesador en sí con capacidad limitada; sin embargo, es capaz de operar en paralelo con el procesador principal.

Los programas para el canal son generados por el software de control ejecutado en el procesador principal y colocados en la memoria central para que sean accesados por el canal. El programa del canal debe especificar la localización de la memoria a utilizar y la operación a ser ejecutada sobre el dispositivo E/S. Entonces la unidad de control principal informa al canal de la localización del programa y lo direcciona para su ejecución. El procesador principal es totalmente libre de continuar operando hasta que el canal lo interrumpa para indicarle que ha completado la asignación de programas o se ha presentado una condición inesperada E/S tal como la presencia de un error.

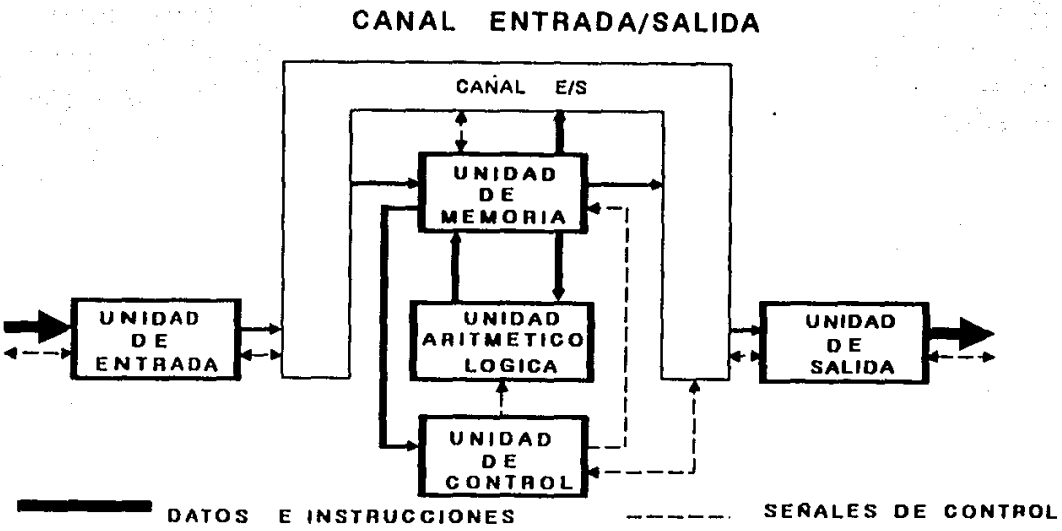


Figura 3.3

III.2.4 El Sistema Pipeline

Provee otro método para obtener la concurrencia o el paralelismo de operaciones. Mientras que los arreglos de procesadores alcanzan la concurrencia por la ejecución de operaciones comunes simultáneamente sobre múltiples datos, la computadora pipeline posibilita la ejecución de operaciones repetidamente a un flujo de datos.

En esta arquitectura, las operaciones aritméticas son transferidas a sucesivos estados, donde unidades separadas del hardware proveen los cálculos de cada estado. Estas computadoras son conocidas como de procesamiento vectorial. La eficiencia de las funciones sólo es posible si las operaciones aritméticas a ejecutar son vectorizables, es decir, arregladas como un flujo continuo de datos.

Una instrucción de pipeline consiste de una secuencia de operaciones externas de bus que ocurren durante la ejecución de la instrucción. Pipeline es el conjunto de operaciones prebúsqueda-decodificación-ejecución siendo esencialmente invisible al usuario, excepto en algunos casos donde el pipeline debe ser interrumpido con instrucciones tales como saltos. En la operación de pipeline, las operaciones prebúsqueda, decodificación y ejecución son independientes. Durante algún ciclo dado, dos o tres instrucciones diferentes pueden ser ejecutadas, cada una en un estado diferente de completez.

Los diferentes niveles en pipeline no necesariamente afectan la velocidad de ejecución, pero si la secuencia búsqueda/decodificación. Una mayor cantidad de instrucciones son ejecutadas en igual número de ciclos.

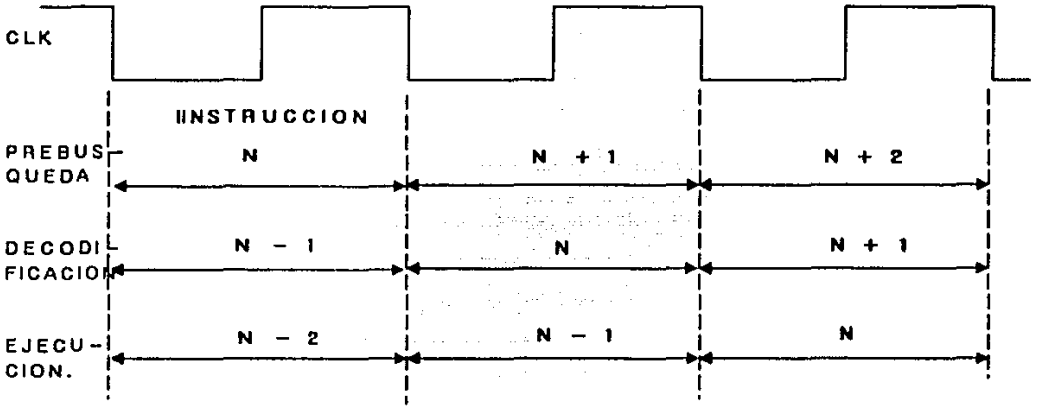


Figura 3.4 Operación de pipeline a tres niveles

En virtud de su amplia aplicación en los diversos procesadores existentes en la actualidad, específicamente la familia TMS320, a continuación se hace un enfoque más concreto sobre este tipo de operación.

En los tres niveles de pipeline, un contador de prebúsqueda contiene la dirección de la próxima instrucción a ser buscada. Una vez que la instrucción es buscada se carga a un Registro de Instrucción. Cuando el Registro de Instrucción contiene la instrucción en ejecución, entonces la instrucción prebuscada debe ser cargada en un Registro de Instrucción de Espera. El contador de prebúsqueda es incrementado una vez que la instrucción en ejecución ha terminado. La instrucción en el Registro de Instrucción de Espera entonces es cargada en el Registro de Instrucción para ser ejecutada a continuación.

De una importancia particular en el diseño de procesos vectoriales son las operaciones aritméticas en pipeline, por ejemplo, la suma de dos números requiere cuatro ciclos de instrucción. En pipeline, cada una de esas cuatro operaciones es ejecutada por una unidad separada de hardware, si se asumen características lentas del circuito electrónico tales que una operación requiera al menos 50 ns, entonces es posible pasar el dato de una unidad a la próxima y comenzar a trabajar sobre un nuevo par de entradas cada 50 ns. Una vez que el pipeline es

completado, una nueva suma aparecerá cada 50 ns. Como se observa en la siguiente figura, donde el flujo de dos bloques de números de entrada $a_1, a_2, \ldots a_n$ y $b_1, b_2, \ldots b_n$ son sumados uno a uno bajo la operación de pipeline.

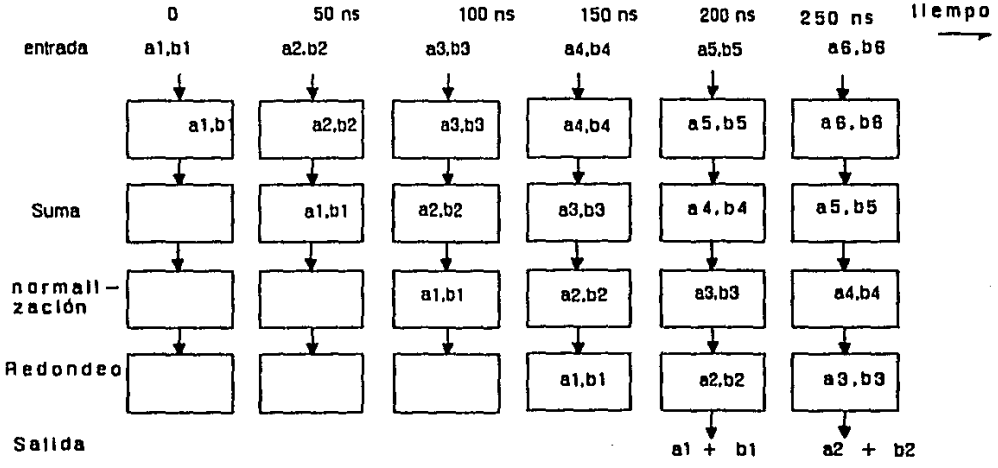


Figura 3.5

III.2.5 Topología Hipercúbica

Uno de los diseños más promisorios en sistemas de múltiples procesadores, es la topología Hipercúbica, desarrollada por Seitz y Fox en 1983. Este diseño también llamado cubo cósmico, demuestra un alto paralelismo y una arquitectura de procesamiento escalable que logra una alta eficiencia. Utiliza la interconexión de procesos elementales o nodos, donde existen $N=2^n$ nodos, cada uno de ellos es conectado por comunicaciones fijas a otros nodos. El valor de "n" es conocido como la dimensión hipercúbica.

En los siguientes diagramas se indica esta topología.

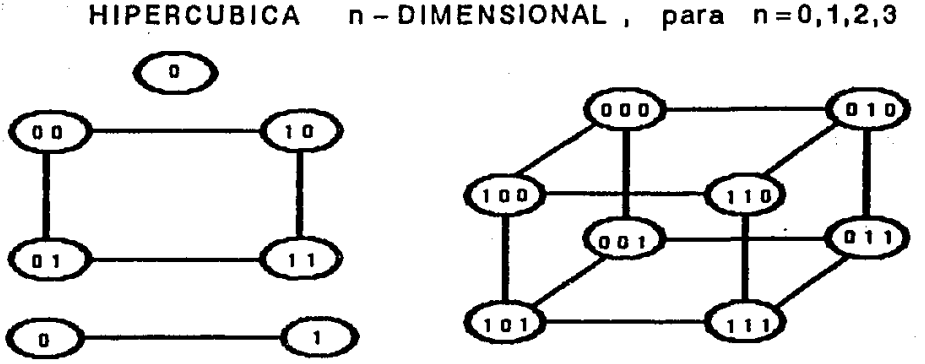


Figura 3.6

Cada nodo del sistema consiste en una unidad de procesamiento completa con la capacidad de comunicarse con otras unidades. El número de nodos puede ser expandido, dando la potencia y velocidad necesaria para el manejo de problemas a resolver en computación. Una extensión de este diseño básico es la adición de canales de E/S, cada uno conectado a los bordes hipercúbicos, o adicionados a los nodos.

La programación típica, consiste en tomar un gran problema y dividirlo en subproblemas. Estos subproblemas son asignados a los nodos hipercúbicos, los cuales comparten resultados o cooperan a través de mensajes enviados entre los nodos. El hipercubo es un ejemplo de un sistema de múltiples procesadores.

Actualmente la literatura reporta diversas implementaciones de sistemas de procesamiento, presentando variantes en cuanto a la arquitectura, tipo de procesadores, capacidad de procesamiento, algoritmos de procesamiento utilizados, etc. Todo esto nos indica la necesidad de obtener sistemas de procesamiento eficientes y rápidos, que a su vez permitan establecer las tendencias más adecuadas en cuanto a procesamiento de imágenes se refiere.

III.3 CLASIFICACION DE LOS SISTEMAS DE PROCESAMIENTO DISTRIBUIDO

Los sistemas de procesamiento distribuido constituyen una potencial solución a los problemas que se presentan en un sistema de computo donde los usuario comparten tiempo en un procesador. Estos sistemas constan de un poderoso computador central con una gran capacidad de memoria y cada usuario o grupo de usuarios tienen una microcomputadora en lugar de una simple terminal de video, es decir que cada estación del usuario es una microcomputadora independiente, pudiendo realizar tareas locales sin hacer uso de la gran computadora, sin embargo, el usuario puede accesar a los recursos de la gran computadora cuando así lo necesite.

Los sistemas de múltiples procesadores forman parte de los Sistemas Distribuidos de Cómputo. Algunas clasificaciones y taxonomías han sido publicadas sobre la materia, existiendo aún diferentes puntos de vista y desacuerdos sobre qué puede ser considerado un sistema distribuido.

A continuación se presentan las tres clasificaciones más comúnmente aceptadas.

III.3.1 Clasificación por granularidad

Una primera forma de clasificar los diferentes tipos de sistemas distribuidos, puede ser tomando en cuenta la granularidad, la cual se define como la interacción entre las actividades que son ejecutadas en paralelo por el sistema, es decir, la capacidad del sistema y el número de procesadores que posee, donde intervienen el tamaño de la palabra de instrucción y el tamaño de memoria a manejar entre ellos. Es posible considerar sistemas sobre los cuales la cooperación (intercambio de datos y/o sincronización) entre elementos ocurre raras veces y al mismo tiempo envuelve grandes bloques de estructura de datos. Por otra parte, se pueden considerar los sistemas distribuidos sobre los cuales el intercambio de datos es muy frecuente y la sincronización ocurre al nivel de instrucción.

III.3.1.1 Redes de cómputo

Es una clase entre los sistemas distribuidos que toma su origen en la conexión de grandes estructuras. Cada procesador trabaja muy autónomo y emplea sólo una parte limitada de su capacidad de procesamiento a actividades comunes; por esta razón no hay un acuerdo general al considerar las redes de cómputo como parte de un proceso distribuido. La principal característica de una red de cómputo es que se mantiene estable, por el hecho de que cada procesador es como una computadora única y sus actividades de procesamiento son independientes.

III.3.1.2 Sistemas de procesamiento múltiple

En los sistemas de procesamiento múltiple, cada procesador es una unidad completamente programable, la cual puede ejecutar su propio programa. Todos los recursos del sistema son coordinados hacia una tarea común con un simple mecanismo de control centralizado o distribuido. La cantidad de información intercambiada entre las unidades básicas de procesamiento es mayor que en el caso de redes de cómputo.

La topología de interconexión y la estrategia de comunicación entre los elementos de procesamiento, se convierte en este caso en un punto crucial del sistema. Una clasificación más detallada debe basarse en la estructura de la red de interconexión. Esta clasificación se basa en la capacidad de la interconexión para compartir o no el espacio de direccionamiento entre los procesadores, a saber:

- **Múltiples computadoras o ligeramente acoplados.** Se define como un sistema en el cual los elementos de procesamiento no comparten memoria y son conectados a través de líneas de E/S en general.

- **Multiprocesadores o sistemas estrechamente acoplados.** Se define como un sistema con estructuras comunes de espacio de direccionamiento donde los diferentes procesos pueden accesar una memoria común.

III.3.1.3 Máquinas de propósito especial

Este sistema de procesamiento es diseñado para resolver problemas que se presentan en campos de aplicación específicos. En esta clasificación podemos encontrar:

- **Altas estructuras de paralelismo.** Están compuestas de un gran número de unidades de hardware idéntico donde cada una es capaz de ejecutar una operación básica fija. Estas unidades se conectan juntas y trabajan en paralelo para soluciones rápidas como en operaciones matriciales o la transformada discreta de Fourier. Un ejemplo es el arreglo sistolítico, cuya arquitectura es muy conveniente para su implementación en circuitos de muy alta escala de integración (VLSI). Un arreglo sistolítico, consiste en un conjunto de celdas de cálculo igualmente interconectadas, de acuerdo a una topología regular, en la cual el flujo de información es permitido sólo en unidades adyacentes en un estilo pipeline. Cada unidad es diseñada específicamente y optimizada para efectuar sólo un tipo de operación definida.
- **Arreglo de procesadores.** Ejecuta a pasos marcados operaciones iguales sobre diferentes datos. Tienen un alto grado de programabilidad respecto a las anteriores, pero su uso es restringido a problemas con un alto grado de paralelismo, tales como la manipulación de grandes arreglos de datos.
- **Máquinas Von Neumann.** Son caracterizadas por la presencia de una unidad de procesamiento que ejecuta instrucciones (almacenadas en memoria) en orden secuencial bajo el control del contador de programa. La ejecución secuencial del programa no permite una eficiente explotación del paralelismo inherente al programa.

III.3.2 Clasificación de Flynn

Está dada de acuerdo a simples o múltiples instrucciones de procesos y el flujo de datos a manejar, considerando el producto cartesiano siguiente: (simple instrucción, múltiple instrucción) X (simple dato, múltiple dato), abreviado: (SI,MI)x(SD,MD) se obtienen las combinaciones: SISD, SIMD, MISD y MIMD.

Donde:

- **SISD**: Simple instrucción, simple dato. Representa el convencional sistema uniproceso de computadora, es decir, la clásica arquitectura de Von Neumann.

- **SIMD:** Simple instrucción, múltiple dato. Incluye más de un sistema de múltiples procesadores, donde las operaciones ocurren síncronamente y usualmente comparten espacio común de memoria.
- **MIMD:** Múltiple instrucción, múltiple dato, incluye más de un sistema de múltiples procesadores, las operaciones ocurren asíncronamente y usualmente comparten espacio común de memoria.
- **MISD:** Múltiple instrucción, simple dato, no tiene sentido.

III.3.3 Clasificación de Enslow

Propone el uso de tres espacios dimensionales para caracterizar los sistemas distribuidos:

- **Distribución de unidades de procesamiento.** Corresponde a la organización física de la estructura del hardware que puede ir de una simple unidad de procesamiento a sistemas geográficamente distribuidos en múltiples computadoras.
- **Organización del control.** Este puede espaciarse desde un sistema con un origen de control fijo a un sistema distribuido compuesto de un set de cooperación completa y unidades de procesamiento homogéneos.
- **La distribución de los datos.** Es posible tener un sistema con una estructura de centralización de datos y un sistema con una base de datos completamente compartidos.

Estas clasificaciones ofrecen un punto de vista global del espacio de diseño y de las múltiples soluciones para sistemas de procesamiento distribuido.

III.4 SISTEMA DE MULTIPLES PROCESADORES

Un sistema de múltiples procesadores consiste de un conjunto de módulos maestros, tales como procesadores, y un conjunto de unidades esclavas (memorias y/o módulos E/S), enlazados por medio de una estructura de interconexión. En general las unidades maestras, son los elementos en el sistema que permiten emitir un requerimiento de acceso de información a través de la estructura de interconexión. Por su parte, las unidades esclavas reciben los requerimientos de acceso de las unidades maestras.

Las memorias pueden dividirse en dos grupos principales. El primero es un conjunto de memorias independientes, cada uno asociado a un procesador y accesadas exclusivamente por éste. Estas memorias pueden retener por ejemplo los programas a ejecutar por cada procesador o datos específicos. El segundo es un conjunto de memorias comunes conteniendo la información que varios procesadores del sistema pueden accesar (con iguales o diferentes derecho de acceso).

La estructura más general de un sistema de múltiples procesadores es descrita por la figura 3.7.

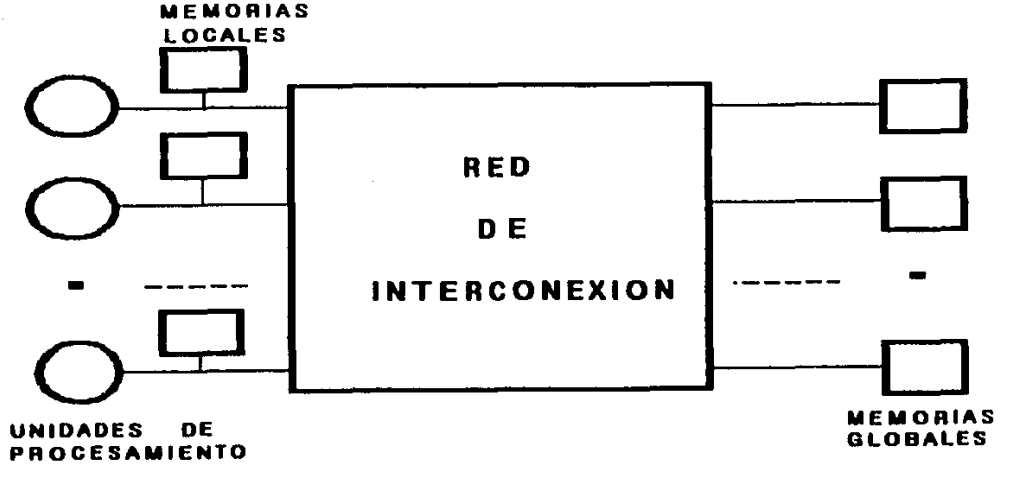


Figura 3.7

Las unidades maestras son representadas por círculos y las esclavas por los cuadros. Las unidades maestras pueden ser módulos de E/S con acceso directo de memoria (DMA) o unidades de procesamiento. Los requerimientos de acceso pueden ser de lectura o escritura. Las redes de conexión son siempre diseñadas de acuerdo a las necesidades de la aplicación.

III.4.1 Redes de Interconexión.

La unidad funcional para ejecutar un programa consiste de un procesador y una memoria. El objetivo de la red de interconexión es acoplar en un instante dado a cada procesador con el módulo de memoria requerido. Las razones para la utilización de estas redes son:

- Dos o más procesadores requieren usar la misma unidad de memoria. En este caso los tiempos de espera pueden ser eliminados sólo por el uso de módulos de memoria con rasgos peculiares, por ejemplo múltiples lecturas u operaciones L/E que se puedan permitir.
- Dos o más unidades de procesamiento necesitan la misma estructura de comunicación para establecer el acceso a diferentes unidades de memoria. Aquí la estructura de la red de interconexión puede reducir el tiempo desperdiciado por los procesadores que esperan cuando no existen recursos disponibles de comunicación.

En ambos casos, el procesador que no pueda accesar a la unidad de memoria requerida debe esperar. Desde el punto de vista de las redes de interconexión existen diferentes soluciones:

- Bus compartido.
- Switches cruzados.
- Redes multietapa.

III.4.1.1 Bus Compartido

Es un simple camino de comunicación en el cual las unidades funcionales (tales como memorias y microprocesadores) son conectados a un bus global:

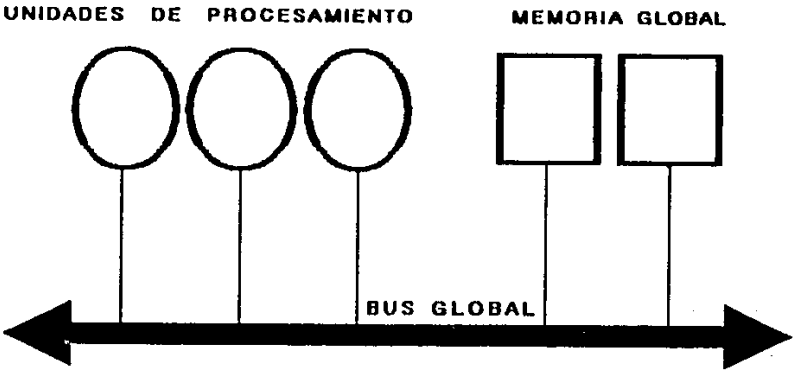


Figura 3.8

Cuando dos o más unidades maestras son conectadas al bus, algún criterio o jerarquía debe utilizarse para establecer la comunicación: una ventana fija de tiempo puede ser asignada a cada unidad maestra, o el sistema debe ser capaz de detectar la contención. En el caso de no existir una ventana fija de tiempo por procesador, los procesadores requieren accesar al módulo de memoria a través de un mecanismo de arbitraje, el cual maneja y resuelve simultáneamente los requerimientos.

Es obvio que esta interconexión no permite transferencias simultáneas entre los diferentes pares procesador/memoria y por tanto una simple estructura de bus puede convertirse fácilmente en un cuello de botella en un sistema complejo. Una solución funcional a esta situación es la utilización de memorias multipuerto, las cuales pueden coadyuvar en el desempeño del sistema, al permitir la creación de buses locales que liberen de ciertas actividades intensivas de comunicación al bus global. Las memorias multipuerto pueden ser direccionadas desde dos (o varios) buses diferentes, sin interferir con otros dispositivos de otros buses, lo que permite accesos independientes a alguna localización de memoria. Estas memorias contienen registros de semáforos que proveen un medio para controlar el acceso de multipuertos, sin embargo, a pesar de sus posibilidades de utilización, su limitada capacidad de almacenamiento de memoria es actualmente su principal desventaja.

III.4.1.2 Switches Cruzados

En tales sistemas un conjunto de rutas separadas es conectada a cada banco de memoria y otro a cada procesador como se observa en la figura 3.9.

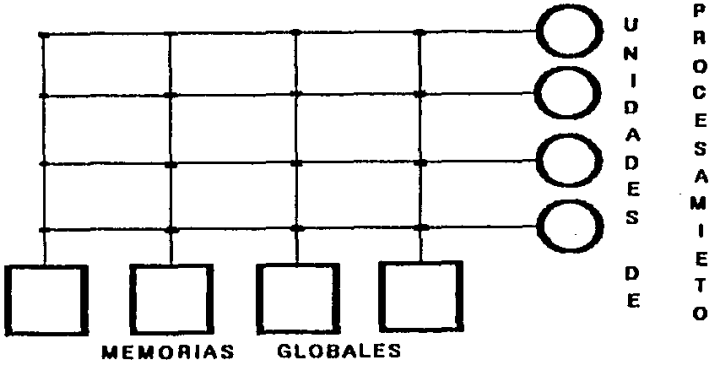


Figura 3.9

Un conjunto de switches puede conectar algún procesador a la ruta de la memoria. El sistema soporta accesos simultáneos hacia todas las unidades de memoria. El problema de contención puede darse sólo cuando iguales bancos de memoria son requeridos por varios procesadores al mismo tiempo. Aunque esta interconexión tiene sus límites, la complejidad y la reducción de costos establecidos por un número óptimo de switches básicos es un punto clave. Recientes propuestas sugieren el uso de elementos VLSI para su implementación.

III.4.1.3 Redes de conexión de múltiples etapas

Puede hacerse usando un arreglo de módulos construidos con bloques de un sólo tipo. Cada elemento puede ejecutar una simple función de switcheo.

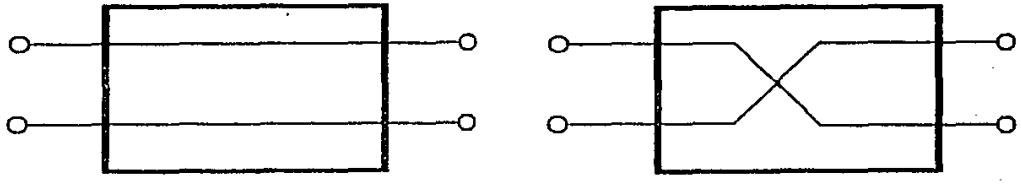


Figura 3.10

Considerando un switch elemental de 2 x 2 como el de la figura 3.10, se pueden tener dos configuraciones, una conexión directa y otra cruzada.

Con un arreglo de N switches elementales se puede ejecutar una etapa simple de la red de interconexión. Así, una matriz de switches elementales básicos, puede interconectar a un conjunto de N terminales de entrada y uno de N terminales de salida.

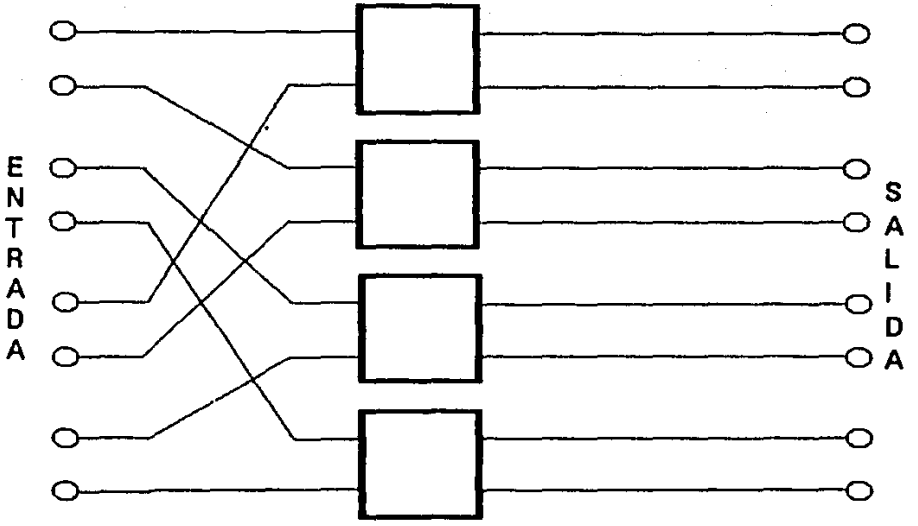


Figura 3.11

En el caso de sistemas de múltiples procesadores la terminal de entrada puede ser un elemento procesador (una unidad de procesamiento asociado a su memoria local) y la terminal de salida puede ser un elemento de memoria global. Una red, puede llamarse **reconfigurable**, si es capaz de arreglar las conexiones que permitan establecer nuevas rutas de comunicación.

III.4.2 Aplicaciones de Sistemas de Múltiples Procesadores

Las ventajas potenciales de los sistemas de múltiples procesadores se pueden expresar por su flexibilidad, extensibilidad y alta disponibilidad. Asimismo, es claro que los sistemas múltiples procesadores no ofrecerán una solución fácil a todos los problemas, que bien pueden convertir en difícil un problema simple de procesamiento.

Algunas razones que pueden complicar el diseño y el uso de las arquitecturas de múltiples procesadores son:

- La mayoría de los microprocesadores ahora disponibles no están específicamente diseñados para el ambiente de multiprocesos.
- No existen herramientas disponibles para desarrollar programas de múltiple procesamiento y por consiguiente herramientas para depuración, o si los hay, ésto sólo pueden utilizarse en algunos sistemas de prototipos específicos.
- No siempre se pueden particionar los problemas complejos en pequeñas actividades que deban ejecutarse en paralelo.

Cabe señalar que una de las aplicaciones más promisorias del procesamiento en paralelo está en los sistemas de procesamiento en tiempo real, debido a las inmensas posibilidades de reducción del tiempo de procesamiento al contar con varias unidades trabajando concurrentemente. Algunos ejemplos de estas aplicaciones son: el control de plantas de potencia, sistemas aerotransportados, sistemas marítimos, estaciones de trabajo, robótica, procesamiento de señales digitales, gráficas por computadora, procesamiento digital de imágenes, simulación, entre otras.

III.5 CARACTERISTICAS DE LOS SISTEMAS EN TIEMPO REAL

Entre las características más relevantes de los sistemas en tiempo real están:

- No pueden "volver atrás" y reiniciar una ejecución desde un contexto preexistente.
- La secuencia temporal de entradas es determinada por el mundo real y no por decisiones de la programación interna del sistema. Normalmente esta secuencia es asíncrona.

- Las demandas del ambiente externo al sistema suelen ser paralelas, no secuenciales. Esto implica problemas de scheduling, prioridades, posibilidades de reentrancia y reinicialización en los módulos de software involucrados, así como una natural tendencia a distribuir el procesamiento para atender localmente las demandas.
- Las restricciones de tiempo que habitualmente caracterizan a estos sistemas hacen que la corrección funcional de un sistema no permita garantizar su utilización en condiciones reales. Esto significa que no se recuperan en tiempo real ante situaciones imprevistas.
- A menudo estos sistemas son "de tiempo infinito", lo que significa que deben estar preparados para auto recuperarse de situaciones que ocurren raras veces.

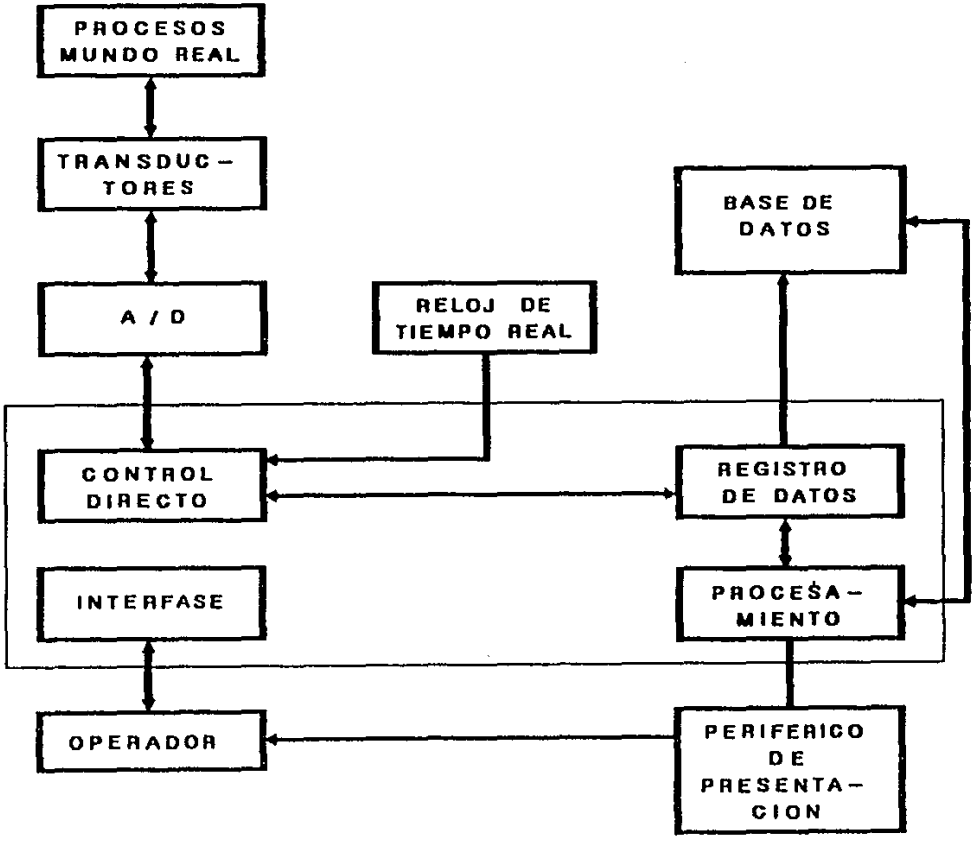


Figura 3.12 Visión esquemática de un sistema de tiempo real

III.6 PROCESAMIENTO DE IMAGENES Y EL TIEMPO REAL

En un SPDI con aproximación al tiempo real, debe tenerse en cuenta la suma de los intervalos de tiempo, que inician con la lectura de la imagen en el sensor de la cámara, la digitalización de la imagen, el tiempo de almacenamiento electrónico, el tiempo de procesamiento y el tiempo de despliegue de la imagen y/o la generación de una respuesta del sistema. La operación de una cámara convencional producirá 30 imágenes por segundo para monitores estándares comercialmente disponibles.

Un sistema que trabaja en tiempo real depende de las restricciones de tiempo impuestas por el medio para efectuar sus funciones dentro de un intervalo permitido. En el caso de un SPDI, el tiempo de respuesta del sistema se ve influido por la capacidad de retención del ojo humano. Es un hecho que no podemos detectar visualmente los 30 cuadros de imágenes por segundo en los monitores de la televisión, mientras que para un número menor de cuadros, la vista detecta discontinuidades de imágenes. Si el SPDI ha de generar una imagen procesada como respuesta del sistema, entonces un intervalo de 66 ms por cuadro es el tiempo con que se cuenta para llevar a cabo el procesamiento de la imagen, lográndose así procesar imágenes de televisión.

En la actualidad, un sistema SPDI usado en la industria para el control de calidad, puede examinar a razón de 900 productos/minuto (ó 15 productos/s tales como etiquetas, botellas, galletas o aquellos con geometría definida). El ojo humano por su parte responde en un tiempo de 60 tareas/min (una tarea/s refiriéndose a la inspección de producción anterior), sin tomar en cuenta que el sistema humano está expuesto a los efectos de la fatiga y las condiciones del ambiente, por lo que se deduce que la aplicación de SPDIs puede llegar a ser mucho más eficiente que el factor humano.

Recientes avances en Sistemas de cómputo en Hardware y Software han hecho del PDI una herramienta muy útil en diversas aplicaciones tales como: análisis de imágenes de satélites, reconocimiento visual de objetos para la industria robótica, restauración de imágenes, análisis de radiografías, técnicas digitales para reducción del ancho de banda en transmisiones de televisión y teleconferencia. Algunas de estas aplicaciones involucran una serie de resultados en tiempo real o requieren del procesamiento de grandes volúmenes de información. Los recientes avances en circuitos digitales y diseño de sistemas han sido posibles gracias a la implementación de la Muy Alta Escala de Integración (VLSI) de circuitos electrónicos.

La tecnología VLSI ofrece nuevos medios para la implantación de nuevas arquitecturas en los sistemas de cómputo, que pueden hacerse efectivos utilizando las estructuras paralelas

o pipeline. El uso del procesamiento en paralelo puede reducir el factor tiempo en un número igual a la cantidad de unidades de proceso utilizadas, lo que requiere del desarrollo de algoritmos que maximicen el aprovechamiento de estos sistemas. Los requerimientos de cómputo para el procesamiento en tiempo real de imágenes son tales que actualmente sólo pueden ser completados con el uso de hardware de propósito especial, de algoritmos de computación eficientes y su optimización para cada aplicación en especial.

Un ejemplo práctico para el análisis de una imagen en tiempo real, es el caso de una imagen de 512 filas de 512 pixeles cada una (256 Kpixels). Esto significa que en tiempo real, para la ejecución de una imagen completa deben procesarse aproximadamente 7.68 millones de pixels/s (teniendo en cuenta la razón de entrega 30 cuadros/s), es decir, un tiempo estimado de proceso de 130 ns/pixel. Un diseño aceptable sería entonces una razón de procesamiento de 100 ns/pixel.

Si se contemplan imágenes de 256 filas con 256 pixel por fila (64 Kpixels/imagen), se incrementa en cuatro veces el tiempo de procesamiento, conservando la razón de 30 cuadros/s, lo que significa que se dispone aproximadamente de 400 ns/pixel para el procesamiento. Ahora bien, si a esto se pretende agregar la posibilidad de contar con más de una unidad de procesamiento, esto es, contar con un sistema de múltiples procesadores, se reduciría el tiempo de procesamiento por imagen en una razón similar al número de procesadores utilizados, es decir, que si tenemos dos unidades de procesamiento disponemos de 800 ns/pixel y para cuatro procesadores 1600 ns/pixel. En el caso de procesadores digitales de señales (DSP) que pueden ejecutar algunas instrucciones en un ciclo de instrucción de 100 ns (tal como el TMS320C25), significa que al contar con un sistema de cuatro procesadores se puede pensar en un algoritmo que ejecute hasta 16 instrucciones/pixel, que dista todavía de efectuar algún tipo de procesamiento relevante en la imagen.

Por otro lado, si se empleara para el procesamiento de la imagen los tiempos que corresponden al despliegue de dos cuadros (66 ms) y con los cuatro procesadores mencionados, implica un tiempo de procesamiento de 3200 nseg/pixel, sin que el ojo humano aprecie notablemente el retardo y/o la pérdida de cuadros en la presentación de las imágenes de salida con respecto a la entrada. Este tiempo daría un margen adecuado para implementar algoritmos de procesamiento más capaces sobre las imágenes.

Estos aspectos prácticos son de utilidad en el desarrollo de la arquitectura propuesta como se presenta en el siguiente capítulo.

IV. DISEÑO DE LA ARQUITECTURA PARA EL PROCESAMIENTO DIGITAL IMAGENES (APDI)

En este capítulo se presentará la justificación de los lineamientos que se tomaron en cuenta para el desarrollo de la APDI, así como la descripción de la misma.

En el desarrollo de la APDI hubo que considerar las características más importantes del Procesamiento Digital del Señales (DSP) y así tener una base sobre los atributos de los sistemas encargados de procesar imágenes.

Primordialmente podemos observar que todos los sistemas del DSP poseen características en común:

1. Emplean algoritmos matemáticos muy intensivos, como la respuesta del impulso finito.
2. Los algoritmos de DSP tienden a ser ejecutados en tiempo real. Esto implica que no se deben producir retardos notables en la presentación de resultados al usuario. Por ejemplo, para poder procesar una imagen en un sistema de video en tiempo real, implicaría utilizar un período de tiempo igual al de la formación de la imagen para su procesamiento.
3. Requieren del muestreo de señales. En ocasiones los procesadores deben efectuar tanto los cálculos aritméticos como controlar el muestreo de datos.
4. Tienen una orientación muy definida por la aplicación.

Algunas características recomendables son:

5. Deben ser lo suficientemente flexibles como para incorporar mejoras en sus diseños, debido a las innovaciones tecnológicas y/o los avances en el desarrollo de los algoritmos.

6. Un adecuado balance del hardware y software en la arquitectura. Los sistemas de DSP involucran un compromiso intrínseco entre la velocidad de procesamiento del sistema y su flexibilidad. Para ello se recurre a diversas alternativas tales como: la capacidad para almacenar información, modularidad, reprogramabilidad, etc. Por ejemplo, teniendo en cuenta el tipo de aplicación se puede diseñar una arquitectura donde sus unidades de procesamiento involucren una baja programabilidad, pero a la vez optimizada para ejecutar eficientemente un algoritmo bien definido, es decir muy específico para la aplicación. La pérdida de flexibilidad para este caso, beneficia en la reducción de los tiempos de procesamiento.

En el caso de la APDI, el procesamiento digital de imágenes ha tomado como base los puntos anteriores, destacando los siguientes:

- El empleo de memorias de alta capacidad.
- La utilización de un software específico a cada aplicación.
- Alta velocidad de procesamiento, con el empleo de uno o varios procesadores de señales para lograr en ciertas aplicaciones procesamiento en tiempo real.
- Facilidades de reprogramación.
- Modularidad, que permite manejar separadamente las principales funciones de la APDI en varios módulos.

IV.1 DESCRIPCION DE LA ARQUITECTURA PARA EL PROCESAMIENTO DIGITAL DE IMAGENES

La implementación de la APDI reune las siguientes características (Fig.4.1):

- Se concibió del tipo SIMD (Simple Instrucción Múltiple Dato), lo cual sujeta a todas las unidades de procesamiento a compartir la memoria de programa (MP). El tipo de memoria empleado como memoria MP es memoria RAM estática, de esta manera las modificaciones de los programas y algoritmos empleados en la APDI son más fáciles de realizar, en comparación con el proceso de borrar y regrabar, como sería el caso de emplear memorias tipo ROM.

- Cada unidad de procesamiento cuenta con un bloque de memoria fuente (MF) para almacenar la imagen a procesar. Con lo anterior se garantiza el acceso a toda la información de la imagen por parte de las unidades de procesamiento.

- La arquitectura está enfocada a la aplicación de la convolución a imágenes matriciales con una máscara de sensitividad espacial, por lo que los resultados de dicho proceso son almacenados en un bloque de memoria independiente denominado memoria resultado (MR). El hecho de contar en la APDI con dos bloques de memoria (MF y MR) para el almacenamiento de la información tanto procesada como a procesar, permite llevar a cabo diversos tipos de procesamiento sobre la imagen fuente.

- En las unidades de procesamiento se utiliza un procesador de señales denominado TMS320C25, perteneciente a la familia TMS320 de la compañía Texas Instruments, capaz de efectuar cálculos intensivos con una alta velocidad de procesamiento (para mayor información en el Anexo A se tiene una descripción detallada de sus características de funcionamiento). Esta familia ha evolucionado incrementado su velocidad de operación, el número de instrucciones e incluso ha hecho más versátil su arquitectura. Es así como en la actualidad se cuenta con los procesadores de señales TMS32010 (20 Mhz), TMS32020 (20 Mhz), TMS320C25 (40 Mhz), TMS32030 (66 Mhz) y TMS32040 (100 Mhz).

- La arquitectura está diseñada para realizar el procesamiento sobre imágenes matriciales de 256x256, mismas que son proporcionadas por medio de una tarjeta digitalizadora. El formato de la información es de tipo byte con un tamaño de 64 Kbytes (256 x 256). Las imágenes obtenidas son almacenadas en forma de archivos.

- Se emplea una PC para tener un enlace de comunicación sencillo con la APDI, ya que, aprovechando los recursos de la PC se pueden emular las condiciones necesarias para evaluar el funcionamiento del sistema. Por lo anterior, la transferencia de la información se realiza a través de una liga sencilla, como lo son los puertos de entrada y salida (E/S).

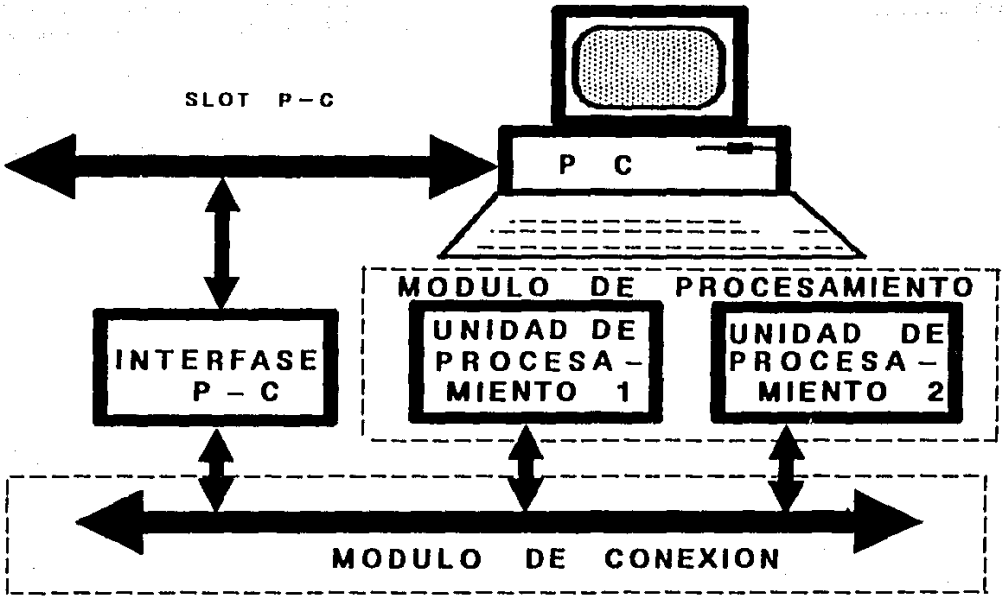


FIGURA 4.1

Figura 4.1

En el diagrama se observan los tres módulos que conforman la APDI, los cuales fueron propuestos en la realización del presente trabajo (ver Capitulo I.3). Las secciones básicas de la APDI se describen a continuación:

a) **Módulo Interfase PC**, el cual consta de puertos de E/S para la transferencia de información y control, además de la memoria de programa.

b) **Módulo de Conexión**, compuesto por un conjunto de conectores que permiten la interconexión entre módulos de las diferentes señales de la APDI (bus de datos, bus de direcciones, señales de control, alimentación, etc).

c) **Módulo de Procesamiento**, el cual consta de una o más unidades de procesamiento, donde cada unidad está integrada por un procesador de señales, los bloques de memoria MF y MR así como la lógica de control.

Para entender mas claramente como funciona la arquitectura APDI, se describe a continuación la teoría de operación.

La arquitectura APDI funciona típicamente de la siguiente forma:

1. Desde un programa de aplicación siendo ejecutado por la PC, se envían las instrucciones de control para la transferencia de información. La PC mantiene en todo momento el control la arquitectura y a la vez conserva un dialogo con el usuario sobre las tareas a realizar.
2. Se realiza la transferencia del programa (en código objeto del TMS) desde un archivo en la PC a la memoria MP. El programa utilizado contiene un conjunto de rutinas que permiten ejecutar los diferentes tipos de algoritmos de procesamiento concebidos para esta aplicación.
3. Una vez elegida la imagen y el tipo de proceso a efectuar desde un menú en el programa de aplicación, la imagen es transferida desde un archivo en disco a la memoria MF. Al finalizar la transferencia, la información sobre qué tipo de proceso ha sido seleccionado por el usuario es enviada a la unidad de procesamiento.
4. La unidad de procesamiento ejecuta el proceso sobre la imagen, informando la finalización del mismo.
5. Para recuperar la información procesada se generan nuevas señales de control desde la PC, con el fin de transferir los datos desde la memoria MR a un archivo de datos y/o desplegarlos en el monitor de la PC.

Una vez obtenidos los resultados se está en condiciones de poder realizar otro tipo de procesamiento sobre la misma imagen, o bien, procesar otra imagen.

La descripción detallada del funcionamiento y elementos que conforman cada uno de los módulos, se realiza en los incisos siguientes.

IV.2 MODULO INTERFASE CON LA COMPUTADORA PERSONAL (MODULO INTERFASE-PC)

La computadora personal (PC) es un recurso al que se ha recurrido para la implementación de la Arquitectura de Procesamiento Digital de Imágenes (APDI), debido a que través de ella se puede simular la operación de los módulos no incluidos en el desarrollo de este trabajo. Por lo anterior, es necesario que a partir de la PC se establezca una interfase que permita la comunicación con la arquitectura APDI.

IV.2.1 Funciones de la interfase

La funciones asignadas a la interfase son:

- Transferir el programa de procesamiento en código objeto del TMS (previamente almacenado en forma de archivo), a la memoria MP.
- Transferir la Imagen Fuente (IF) almacenada en forma de archivo a la Memoria Fuente (MF) en el módulo de procesamiento.
- Transferir de la memoria MR, ubicada en el módulo de procesamiento, la Imagen Resultado (IR), para ser desplegada en el monitor de la PC y/o almacenarla en forma de archivo.
- Enviar las señales de control necesarias para efectuar las funciones anteriores.
- Mantener el control desde la PC de la arquitectura APDI por medio del programa de aplicación a través de los puertos de E/S.

Las PC cuentan normalmente con 8 ranuras de expansión (slots) para la conexión de tarjetas que permiten el manejo de dispositivos periféricos, los cuales al interactuar con la PC incrementan su potencialidad de trabajo al conjuntarse todos estos recursos. Sin embargo, dichos dispositivos deben de observar una serie de lineamientos, para asegurar el correcto funcionamiento de la computadora cuando éstos quedan instalados a través de un slot.

El slot de una PC tipo XT cuenta con: un bus de datos de 8 bits (D0..D7), un bus de direcciones (A0..A19) en el cual se puede direccionar hasta 1 Mbyte de memoria, señales de

control y líneas de alimentación (+5V, -5V, +12V, -12V d.c. y GND). Este conjunto de líneas permite recibir y enviar información entre la PC y los dispositivos conectados al slot.

Tomando como referencia la descripción para un slot, se presenta en la figura 4.2 un diagrama a bloques del Módulo Interfase-PC.

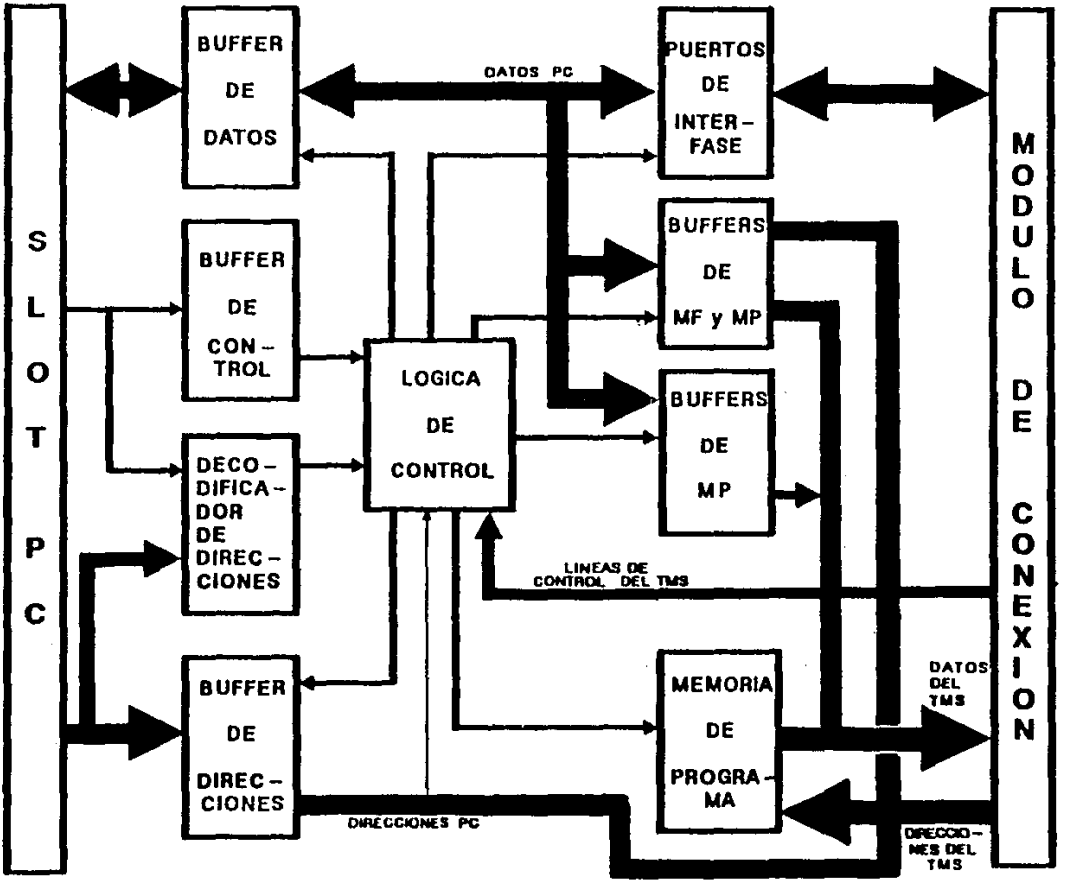


Figura 4.2 Diagrama a bloque de la Interfase PC

Con ayuda del diagrama de la figura 4.2 y tomando en cuenta además el circuito electrónico del anexo B, se hace una descripción de cada bloque, las líneas de interconexión y las señales que intervienen en su funcionamiento.

IV.2.2 Memoria Programa (MP)

Del diagrama de la figura 4.2 se observa la memoria MP, la cual en el circuito correspondiente al módulo de interfase PC (anexo B) está formada por los circuitos CI U14A y U14B, que son memorias RAM estáticas de 2Kx8. Al emplear dos memorias en paralelo se forma un bloque de 2Kx16, obteniendo los 16 bits de información necesarios para las instrucciones del TMS. Por otro lado, para la transferencia de datos con la PC se controlan en forma individual (2Kx8) y no como un bloque (2Kx16). Estas razones hacen necesaria una lógica de control para ambos casos.

IV.2.3 Etapa de buffereo

Al emplear un slot de la PC, es muy importante respetar el manejo de cargas (fan-out). Este lineamiento establece que cada señal activada desde la PC debe de observar una sola carga por slot de parte del dispositivo periférico instalado. Para ello, los buffers a la entrada de la interfase aseguran no se cargue excesivamente a los dispositivos de salida de la PC en el slot, dando un manejo adecuado de los niveles lógicos de las señales.

Refiriéndose de nueva cuenta al circuito electrónico del Módulo Interfase PC, el circuito integrado CI U9, corresponde al buffer para el bus de datos, el cual es de doble dirección para manejar lectura y escritura. Los CIs U3 y U4 corresponden a los buffers de la direcciones bajas A"0..A"11, así como de las señales de control. Las señales A"1..A"11 forman un bus interno de direcciones que permite accesar a la memoria de programa desde la PC. Se hace la observación que las direcciones A'0..A'1 se utilizan exclusivamente en la decodificación de puertos. La señal PAR_IMP se utiliza en la selección de la memoria par o impar de MP y corresponde a la salida A0 buffereada.

IV.2.3.1 Buffers para L/E de la memoria MP

El bloque de buffer para la memoria MP, corresponde a los CIs U13A y U13B, que envían o reciben datos de la memoria MP formada por una parte alta, así como por una parte

baja. La memoria MP está constituida por un bloque de 2Kx16. De este modo, dos localidades consecutivas forman un espacio de 16 bits, que físicamente están divididas en dos espacios separados de memoria. Los CIs U13A y U13B son habilitados por la lógica de control cuyas funciones son:

- Convertir el bus de datos de 8 bits de la PC a 16 bits del TMS.
- Servir como un medio para escribir y leer el contenido de la memoria MP a través de la PC.
- Aislar el bus de datos de la memoria MP del bloque de puertos E/S de la interfase (debido a que la PC requiere mantener una comunicación con la APDI por medio de puertos mientras que el TMS puede estar accesando a la memoria MP).

IV.2.3.2 Buffers de transición de las memorias MF y MR

Los elementos que integran esta sección son los CIs U31, U32 y U33, donde los dos primeros son tipo registro con salida tres estados para el manejo de direcciones y el tercero es un buffer con salida tres estados de doble dirección para datos. El intercambio de los datos correspondientes a las imágenes fuente o resultado, se realiza a través de una lectura o escritura de puerto desde la PC. Para direccionar un localidad de memoria con 65 Kbytes son necesarios 16 bits de direcciones, por lo tanto se emplean dos registros (U31 y U32) para tener los 16 bits de dirección. Al enviar una dirección, ésta se divide en dos partes: las direcciones altas (A"8 .. A"15) y las direcciones bajas (A"0 .. A"7).

IV.2.4 Decodificador de direcciones

Para la utilización de la interfase es necesario trabajar en un espacio libre de memoria y/o de puertos en los mapas de memoria y puertos de la PC para no interferir con otros dispositivos y/o memorias ya instalados. Esto se logra con el decodificador de direcciones, que permite hacer la selección de un espacio previamente elegido. En nuestro caso, la decodificación se hace dentro del espacio de memoria correspondiente a D0000 a DFFFF (hexadecimal). Este espacio corresponde al asignado a la tarjeta de expansión de ROM BIOS , el cual normalmente no es utilizado [Murray and Shoemaker, 1986].

El decodificador, es un comparador dado por el CI U1, el cual por un lado tiene un banco de interruptores programables para seleccionar la dirección base, la cual se compara con las líneas del bus de direcciones A13..A19 provenientes de la PC y habilitadas desde el

programa de aplicación. La señal AEN (Address Enable) también se involucra en la comparación, debido a que nos permite validar la información presente en el bus de direcciones. AEN es enviada por el controlador de Acceso Directo de Memoria (DMA) de la PC, e indica que un ciclo DMA está en progreso. Cuando las direcciones A13..A19 son iguales a la dirección base y la señal AEN es baja, entonces el comparador activa la señal /P=Q, indicando que se tiene una dirección en el espacio de memoria escogido.

La señal /P=Q llega a la lógica de control, la cual permite habilitar los buffers ya mencionados para permitir el flujo de información entre la PC y la interfase.

IV.2.5 Lógica de Control

Este bloque recibe las señales de control necesarias de la PC, el decodificador de direcciones y del módulo de procesamiento. A su vez, envía señales a los respectivos dispositivos para que efectúen sus funciones asignadas.

La señal /HOLD, es activada por la PC a través del programa de aplicación para mantener en alta impedancia a los TMS instalados en el módulo de procesamiento. La lógica de control también recibe esta señal, que junto con /CE_MFR garantizan que durante el acceso de la PC a las memorias MP, MF y MR, el TMS no pueda efectuar ningún direccionamiento a ellas.

Las señales /PStms y /(R/w)tms, provienen de la lógica de control del módulo de procesamiento y sirven para determinar el momento apropiado de acceso por parte del TMS a la memoria MP. Las señales /SMEMW' y /SMEMR', son utilizadas por la PC para efectuar una lectura o escritura en memoria. Con la señal /SMEMR' se establece en los buffers de datos (U9, U13A, U13B) el sentido del flujo de información, ya sea de lectura (bajo) o escritura (alto).

IV.2.5.1 Control de la memoria MP

El diseño contempla las operaciones de lectura/escritura de datos desde la PC a la memoria MP. La información se envía en dos partes una alta (D"8 .. D"15 a la memoria MP parte alta) y otra baja (D"0 .. D"7 a la memoria MP parte baja), con el fin de formar un dato de 16 bits al tenerse un bloque de memoria de 2Kx8.

La lógica de control realiza el manejo de las señales de escritura, lectura, habilitación de la localidad alta (/CEA) y la localidad baja (/CEB) de la memoria MP. Debido a que a través de la PC sólo podemos enviar una palabra de ocho bits, es necesario dividir el código de operación del TMS en parte alta y parte baja, y así enviarlo en dos partes para escribirlo adecuadamente en la memoria MP. Bajo esa necesidad, la lógica de control tiene que enviar las señales necesarias para escribir correctamente la parte alta y baja de la memoria MP. Para el control de memoria MP se utilizan las señales:

- /CEA : habilita el buffer y la memoria MP para la parte alta de datos.
- /CEB : habilita el buffer y la memoria MP para la parte baja de datos.
- /WE_PC : habilitación de escritura en la memoria MP desde la PC.
- /OE_PC : habilitación de lectura de la memoria MP desde la PC.

Para realizar el proceso de escritura se emplean las señales /CEA , /CEB y /WE_PC a través de un decodificador de memoria formado por compuertas AND y su función es presentar las señales de habilitación para la parte baja y alta de los buffers y la memoria MP. Para efectuar la lectura de la memoria MP, se utilizan las señales /CEA, /CEB y /OE_PC del mismo decodificador. Esta lógica permite además habilitar la lectura de la memoria MP como un bloque de 2Kx16 desde el Módulo de Procesamiento al recibir de este módulo las señales /Pstms y /(R/w)tms.

IV.2.5.2 Control de puertos

Dentro del espacio de memoria seleccionado, se ha ubicado asimismo los puertos de lectura y escritura del módulo interfase PC. La línea de dirección A12, sirve para hacer la selección entre las secciones de memoria o puerto (D0 hex para memoria y D1 hex para puertos).

Las señales /A12, /P=Q y /SMEMR' que son activas bajas, habilitan el decodificador de puerto de lectura (U7B). Con las señales /A12, /P=Q y /SMEMW', se habilita de igual manera el decodificador de puerto de escritura (U7A). Con las direcciones A'0..A'1, se selecciona uno de cuatro puertos en ambos decodificadores.

IV.2.5.3 Control de lectura y escritura en las memorias MF Y MR

La selección de las memorias MF y MR desde la PC se realiza utilizando los puertos que contiene el módulo interfase PC, los cuales se accesan a través de una dirección base D100X programada en los switches del comparador U1, como se muestra en la tabla siguiente:

PUERTOS DE ESCRITURA		
DIRECCION	PUERTO	FUNCION
D1000	/P.ESC1	Escribe señales de control al TMS: /HOLD, /RESET TMS, /CE_MFR, /SINC, PROC0, PROC1 y PROC2.
D1001	/P.ESC2	Control de la parte baja de la dirección para escribir en la memoria MF y leer en la memoria MR.
D1002	/P.ESC3	Control de la parte alta de la direcciónpara escribir en la memoria MF y leer en la memoria MR.
D1003	/P.ESC4	Control del buffer de datos para escribir en la memoria MF.
PUERTOS DE LECTURA		
D1000	/P.LEC1	Sensa el status del TMS
D1001	/P.LEC2	Control del buffer de datos para lectura en la memoria MR
D1002	/P.LEC3	No utilizado
D1003	/P.LEC4	No utilizado

Tabla 4.1

Del puerto de escritura /P.ESC1, los bits identificados como **PROC0, PROC1 y PROC2** son utilizados por la PC para enviar a la APDI un dato que corresponde al tipo de procesamiento en la imagen seleccionado por el usuario.

El puerto de lectura /P.LEC1, se encarga de detectar el estado del TMS a través de los bits **B1_CONTROL, B2_CONTROL**, con los cuales se identifica el inicio y finalización del procesamiento.

IV.3 IMPLEMENTACION DEL MODULO DE CONEXION

En la APDI, el módulo de conexión constituye la convergencia de señales entre los módulos Interfase PC y de Procesamiento. Esta característica se debe al tipo de arquitectura planteada, que al ser de tipo modular, algunas señales son comunes a varios módulos. Asimismo se prevee que los módulos no involucrados en este trabajo hagan un manejo de sus señales a través de este módulo de conexión.

IV.3.1 Función del módulo de conexión

La función primordial del módulo es:

-Disponer de todas las señales de control, bus de datos y de direcciones de la APDI y alimentación a través de un conjunto de Slots.

La APDI está constituida por varios módulos que interactúan formando el sistema global. Para satisfacer las necesidades de una arquitectura modular se implementa un módulo de conexión cuya representación en un diagrama de bloques es el siguiente :

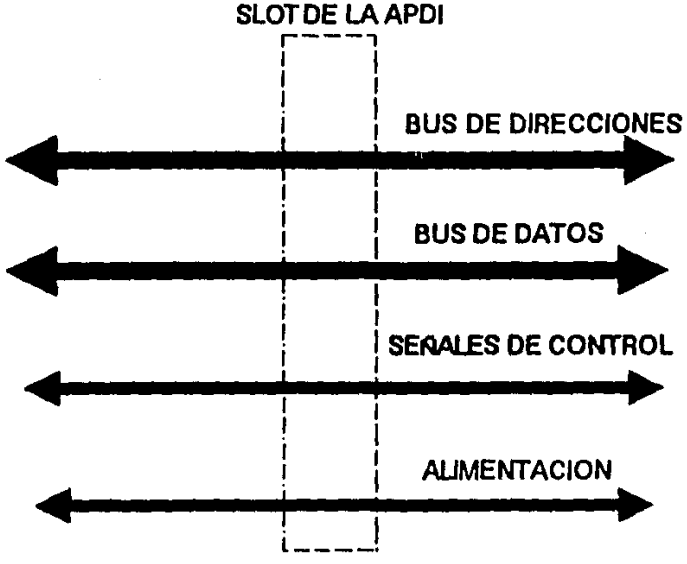


Figura 4.3 Módulo de Conexión

IV.3.2 Slot de la APDI

Constituye todas las señales de uso común en la APDI, y que son:

- **Bus de direcciones :**
  Son las líneas de A"0 .. A"15, con las cuales se efectúan los direccionamientos dentro del sistema.
- **Bus de Datos :**
  Este es integrado por las líneas D"0 .. D"15 a través de las cuales se transfieren los datos a la APDI.
- **Líneas de Control TMS**
  /HOLD: Es la señal que permite mantener en estado de espera al TMS y dejar a todos sus buses en alta impedancia.
  /SINC : Esta señal es enviada desde el programa de aplicación durante la activación de la señal /RESET TMS para la sincronización de los relojes internos de cada procesador en sus respectivas unidades, lo que permite que los TMS ejecuten sus operaciones a pasos cerrados en la lectura de su código de operación en la memoria programa MP común a las unidades.
- **/RESET TMS:** Señal de reset de la APDI por medio de software.
- **/CE_MFR:** Se envía a través del programa de aplicación, y se utiliza para la habilitación de los registros que direccionan la escritura de la memoria MF y la lectura en la memoria MR.
- **/HOLDA :** Esta señal es enviada por el TMS para indicar que este ha puesto sus buses en alta impedancia.
- **/Pstms :**Esta señal es enviada por el TMS para la habilitación del bloque de MP.
- **/(R/w)tms :** Es la señal de control del TMS en la lectura/escritura para su transferencia de datos y programa.
- **/SMEMW':** Señal de control de la PC en la escritura de datos.
- **/SMEMR':** Señal de control de la PC en la lectura de datos.
- **/SMEMR1:** Es la señal de control de la PC en la lectura de datos almacenados en la MR.
- **/CE1M-D:** Señal de habilitación de la PC para la lectura/escritura de datos en las MF y MR.
- **A"15:** Es la señal de habilitación para identificación de los bloques de MF y MR en su partes alta o baja.

- **/RESET** : Es la señal de reset general de la APDI.
- **CLOCK** : Esta es la señal de reloj de la APDI, a una frecuencia de 20 Mhz.
- **Alimentación:**

  +5V : Fuente de voltaje de 5 Vdc.

  GND : Referencia de voltaje (tierra del sistema).

IV.4 IMPLEMENTACION DEL MODULO DE PROCESAMIENTO

Este módulo constituye la parte medular de la APDI, debido a que en él se lleva a cabo el procesamiento de las imágenes con ayuda del procesador el TMS320C25 (TMS), el cual cuenta con un amplio conjunto de instrucciones orientadas al procesamiento de señales digitales y puede efectuar una gran cantidad de operaciones por segundo.

IV.4.1 Funciones del módulo de procesamiento

Las funciones del módulo de procesamiento son las siguientes:

- Manejo de la Imagen Fuente (IF) en el banco de Memoria Fuente (MF).
- Manejo de la Imagen Resultado (IR) en el banco de Memoria Resultado (MR).
- Recibir las señales de control que provienen del módulo interfase PC.
- Manejar señales de control del módulo de conexión.
- Accesar el banco de Memoria Programa (MP) para ejecutar el programa de procesamiento a realizar por el TMS.
- Llevar a cabo la ejecución del programa de procesamiento sobre la imagen fuente (IF) para obtener la imagen resultado (IR).
- Implementar un bus local para transferencia de datos, que le permite a cada unidad de procesamiento ejecutar su programa sobre un bloque de datos correspondiente a una parte de la imagen.
- Decodificar la dirección de la memoria MR enviada por la PC de acuerdo al número de tarjetas de procesamiento instaladas.

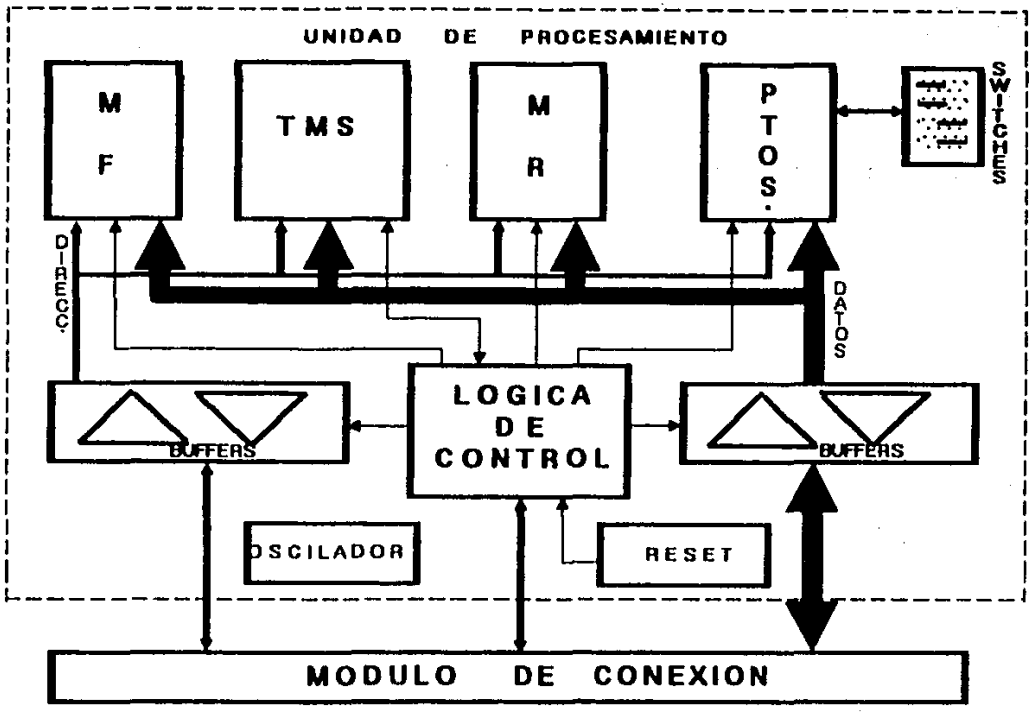


Figura 4.4 Módulo de Procesamiento

Del diagrama de bloques (Fig. 4.4) y el circuito electrónico (anexo B), se observa que el TMS es el elemento central de este módulo. Las direcciones A"P0..A"P14 del TMS están conectadas directamente a cada una de las memorias MF y MR de 32k x 8 con el objeto de leer la información de la imagen IF en la memoria MF y escribir la imagen IR para almacenarla en la memoria MR. La dirección A"P15 va conectada a un decodificador que junto con la señal R/wtms establecen la selección de memorias. Cabe resaltar, que el TMS sólo puede direccionar un banco de 64k palabras de memoria dato. Sin embargo, para los efectos de procesamiento dentro de esta arquitectura, se ha recurrido a la señal de lectura/escritura (R/w) que permite incrementar aparentemente la memoria de dato a 128K. Esto se logra diferenciando con la señal R/w el banco de memoria MF del banco de memoria MR, lo que posibilita el manejo de una imagen fuente de 64k bytes, procesarla y escribir el resultado en otro banco de 64k bytes. Contar con estos dos bancos nos permite mantener en la memoria MF una imagen IF sin que

sufra alteraciones durante el procesamiento, lo que significa que a través del programa de aplicación se puede llevar a cabo otro tipo de procesamiento sobre la misma imagen.

Las líneas de direcciones (A"P0..A"P15) y de datos (D"P0.. D"P15) constituyen un bus local en el momento que la unidad de procesamiento está efectuando la lectura o escritura de un dato. Este módulo se comunica con el de conexiones a través de buffers para que el TMS pueda estar leyendo sus códigos de programa en la memoria MP, o cuando la PC efectúa la escritura en la memoria MF o la lectura en la memoria MR.

Las líneas D"P0..D"P7 sólo se conectan a las memorias MF, MR, y a los puertos de lectura y escritura, ya que para efectos de manejo de datos se están utilizando ocho bits.

El TMS cuenta con las señales de control /PS, /DS e /IS, las cuales se activan cuando el TMS está direccionando la memoria programa, memoria dato y acceso de puertos, respectivamente. La señal de /STRB se activa baja para indicar que el TMS está efectuado un ciclo de bus externo. La señal R/w en nivel alto indica que el TMS puede efectuar una operación de lectura en tanto que para nivel bajo una de escritura. La señal de /MSC estando activa baja indica el inicio de una operación interna o externa tanto de memoria como de operaciones entrada/salida. Todas estas señales son activadas por el TMS y se mandan a la lógica de control del módulo de procesamiento, así como algunas de ellas a la lógica de control del módulo interfase PC. La señal /HOLD es una entrada activa baja que obliga al TMS a poner sus buses en alta impedancia, por lo que es conectada directamente al módulo de conexión. La señal de /HOLDA es una señal de salida y corresponde al reconocimiento que el TMS envía cuando recibe una señal de /HOLD previa; ésta la emite para indicar que sus buses han sido puestos en alta impedancia y por lo mismo también se conecta al módulo de conexión.

IV.4.2 Memoria Fuente (MF)

El bloque de memoria fuente (MF), corresponde a dos memorias de 32k x 8, las cuales funcionan como seudo doble puerto, ya que pueden ser accesadas en un lado por el módulo de conexión para la escritura de la IF, y por otro lado por el TMS para obtener la información a procesar. La función principal de este bloque consiste en almacenar la imagen fuente a procesar. Se hace notar que este bloque de memoria tiene una lógica de control, en tal sentido que el TMS pueda sólo leerla y por medio del módulo de conexión pueda sólo ser escrita.

El bloque de memoria resultado (MR) es de características similares a la memoria MF pero con lógicas opuestas, es decir, que a través del TMS sólo se escribe y por medio del módulo de conexión sólo se lee.

IV.4.3 Buffers

El módulo de procesamiento puede constar de una, dos o cuatro unidades de procesamiento, por lo que se hace necesario evitar cualquier tipo de conflicto entre los buses de las unidades, ya que cada unidad está ejecutando el mismo programa pero sobre diferentes datos. Esto se logra utilizando los buffers bidireccionales en los buses de datos y direcciones (CIs U44, U45, U46 y U47).

La razones para que estos buffers sean de tipo bidireccional son debido a que la unidad de procesamiento debe direccionar y leer a través de ellos la memoria MP. Por otro lado, le permite a la PC transferir información a las memorias MF y MR. El control de estos buffers se efectúa por medio de las señales /CE_B, DIR_A y DIR_D provenientes de la lógica de control del módulo.

IV.4.4 Puertos

El modulo de procesamiento consta de dos decodificadores de puertos que permiten tener hasta cuatro puertos de lectura y cuatro de escritura, de los cuales sólo se están utilizando uno para lectura y otro para escritura.

El puerto de escritura utiliza el registro CI U18, a través del cual el TMS envía directamente al módulo interfase PC dos bits de control (B1_CONTROL y B2_CONTROL), que permiten al programa de procesamiento avisar cuándo se inicia o finaliza y/o se encuentra en alguna etapa intermedia de procesamiento. Una vez terminado el procesamiento de la imagen, el programa de aplicación puede tomar nuevas decisiones desde la PC respecto al tipo de procesamiento a seguir.

El puerto de lectura del TMS (U19) permite recibir información en los bits D"P0..D"P2 sobre qué proceso debe efectuar o seguir haciendo. En los bits D"P4..D"P7 se tienen conectados cuatro switches programables, los cuales proporcionan la información necesaria al programa de procesamiento para identificar el número de unidades de procesamiento instaladas y la cantidad de datos que va a procesar cada unidad.

SWITCHES	NUMERO DE UNIDADES	BLOQUE DE DATOS A PROCESAR
S7 S6 S5 S4		
0 0 0 0	1	64 KB
0 1 0 0 0 1 0 1	2	32 KB
1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1	4	16 KB

Tabla 4.2

Para las combinaciones faltantes en los switches S5 y S4 en los casos de una o dos unidades, se debe de tener cuidado de no utilizarlas, ya que el programa de procesamiento los leería involucrando errores en la lectura de una tabla de constantes correspondientes al inicio y fin de procesamiento[1].

IV.4.5 Lógica de control

La lógica de control recibe las señales que activa el TMS para el control de los dispositivos que componen este módulo.

Estando las señales de /STRB y /PS en nivel activo bajo, produce en el CI U20C la señal /Pstms baja, para la selección del espacio de memoria MP a través del módulo de conexión. La señal /(R/w)tms se envía al módulo de conexión para la selección de la memoria MP; esta señal está formada por la negación de R/w y /STRB al pasar por el CI U20D.

La señal /IS junto con la señal R/w, sirven para habilitar los decodificadores de puertos de lectura U34A y escritura U34B, a los cuales se les envían las direcciones A"P0..A"P1 del TMS para la decodificación de puertos. Con la salida /CS_PL0 y PTOS_LEC se selecciona el puerto de lectura del TMS para recibir información desde la PC. La señal PTOS_LEC, viene de invertir /IS (a través de U14B) y R/w al pasar ambas por el CI U15B. El puerto de

[1] Ver implementación del programa de procesamiento, capítulo 5.

escritura 1 (el registro U18) es seleccionado por la señal CS_PE0 para escribirle una palabra de salida, de lo contrario mantiene siempre la última palabra escrita.

Para la selección de las memorias en general, existen dos decodificadores, uno de los cuales recibe las señales de control del módulo de conexión para efectos de escribir en MF y leer en MR. Para accesar desde la PC, el decodificador U34A recibe una señal de selección /CE1M-D para su habilitación y /SMEMW' con A''P15 para la selección de cualquiera de los cuatro bloques de memoria de 32K:

A''P15	/SMEMW'	Memoria seleccionada
0	0	Escribe en MF1
0	1	Lee en MR1
1	0	Escribe en MF2
1	1	Lee en MR2

Tabla 4.3

Para accesar las memorias desde el TMS, el decodificador U34B, es habilitado por la señal /CE2M-D que proviene de la actuación de las señales /DS y /STRB al pasar por U37B. Este decodificador selecciona cualquiera de los cuatro bloques de memoria de 32 K por medio de las señales R/w y A"P15 :

A"P15	R/w	Memoria seleccionada
0	0	Escribe en MR1
0	1	Lee en MF1
1	0	Escribe en MR2
1	1	Lee en MF2

Tabla 4.4

Antes de seleccionar las memorias, las salidas de los decodificadores pasan por las compuertas U36A, U36B, U36C y U36D que a la vez habilitan un bloque de memoria. La lógica de escritura en la memoria MF queda establecida por /Y0 y /Y2, que van a los pines /WE de habilitación de escritura de las memorias y provienen del decodificador U34A. Para la lectura de la memoria MF, la señal /OE es habilitada por la señal /(R/w) del TMS. Para el bloque de

memoria MR, los pines /WE son habilitados por /YY0 y /YY2, provenientes del decodificador U34B. Las entradas /OE son habilitadas por /SMEMR1 que vienen del módulo de conexión.

Como se mencionó en el bloque de buffers, la unidad de procesamiento establece un bus local cuando está efectuando sus funciones locales, el TMS utiliza este bus local para la transferencia de información entre las memorias MF y MR. Sin embargo, cada unidad de procesamiento debe compartir la misma memoria MP por lo que se hace necesario habilitar y deshabilitar estos buffers utilizando las señales /PS y /CE_FR. La lógica de selección del sentido de transferencia de información utiliza el multiplexor CI U48 con las señales que se explican en la siguiente tabla:

/PS	R/w	DIR_D	DIR_A	FUNCION
0	1	1 (Vcc)	0 (GND)	Habilitación del sentido de transferencia de los buffers para la lectura de memoria MP
1	1	/SMEMR'	1 (Vcc)	Habilitación del sentido de transferencia de los buffers para la escritura de la memoria MF o lectura de la memoria MP desde la PC

Tabla 4.5

La combinaciones faltantes de /PS y R/w no se presentan, ya que es el caso cuando R/w = 0 el programa de procesamiento no contempla la escritura en memoria programa.

La lógica de decodificación agrega otra parte que pueda habilitar los buffers. Esta lógica se vuelve más compleja debido a que tiene que responder cuando el TMS requiera ir a leer su código de programa a la memoria MP, habilitar los buffer cuando la PC vaya a leer el resultado en la memoria MR de cada tarjeta y habilitar solamente los buffers de una tarjeta a la vez cuando corresponda a la dirección A"15..A"14.

UNIDADES	S7 S6	S5 S4	UNIDAD	N_UND
1	0 0	0 0	1/1	0
2	0 1	0 0	1/2	
		0 1	2/2	0
				/A=B
		0 0	1/4	0
4	1 0	0 1	2/4	0
		1 0	3/4	0
		1 1	4/4	0

Tabla 4.6

Con las posiciones de los switches S7..S6 se seleccionan las entradas al multiplexor U50, el cual pone en la salida N_UND la señal adecuada dependiendo del número de unidades utilizadas; para una unidad la salida es cero, para dos, la salida depende de A"P15 y S4 que provienen a la vez de una compuerta XOR U51B, para el caso de cuatro unidades se presentan cuatro posibilidades que dependen de la comparación de las direcciones A"P15..A"P14 con los switches S5..S4, obteniéndose una posibilidad por unidad en el comparador de cuatro bits U49. La señal N_UND junto con /CE_B habilita una salida baja en la compuerta XOR U51A para seleccionarse con la señal DIR_D' (señal que proviene de la lógica cuando el TMS requiere habilitar los buffer de datos para la lectura de la memoria MP) en la compuerta U52A, lo que permite habilitar el bloque de datos indistintamente por el TMS o la PC. La compuerta U53A garantiza la habilitación correcta de los buffers de datos con la señal /CE_BD.

IV.4.6 Bloque de reloj

Los sistemas digitales secuenciales síncronos requieren de una base de tiempo. En este caso la APDI cuenta con una señal de reloj que es empleada en las unidades que forman el Módulo de Procesamiento. La generación de la señal de reloj se basa en una configuración de oscilación con cristal. El arreglo de oscilación se forma con dos inversores (U13) realimentados y conectados con un cristal de 20 Mhz (según las especificaciones del TMS, éste puede trabajar hasta una frecuencia máxima de entrada de 40 Mhz). A la salida del oscilador, que es aproximadamente senoidal, se le da la forma cuadrada con niveles lógicos TTL a través de dos inversores.

IV.4.7 Bloque reset

En cualquier sistema digital es importante inicializar todos sus componentes para garantizar un estado preestablecido y tener el control desde el momento de energizar el sistema. La señal encargada de efectuar esta función en la APDI se le denomina /RESET, que es habilitada por el push button (SW1), o la energización del sistema y/o el programa de aplicación con una instrucción que proporciona la señal identificada como /RESET TMS.

IV.5 TIEMPOS DE ACCESO A LOS BANCOS DE MEMORIA

Como se ha descrito previamente, los bancos de memorias MP, MF y MR, son accesados tanto por la PC como por el TMS, siendo necesario hacer un análisis de los tiempos requeridos en la transferencia de información entre estos dispositivos y las memorias.

Para el análisis se toma en consideración que se trabaja con una PC XT o AT que trabaja a un máximo de 12 Mhz, por lo que los requerimientos de lectura/escritura resultan no ser tan restringidos como los requeridos por el TMS. La PC normalmente utiliza memorias con tiempo de acceso de 100 a 150 ns, mientras que el TMS320C25 requiere para su memoria de programa una memoria con tiempo de acceso menor a 40 ns para cero estados de espera (operando a 40 Mhz). En este diseño, debido a la lógica de doble acceso a los bancos de memoria, la lógica de decodificación y selección, resulta ser más compleja y por lo mismo incorpora tiempos de retardo. Debido a estas características el análisis de tiempo se realiza únicamente con respecto al TMS, deduciéndose por lo tanto que al cumplirse con las restricciones de

tiempo del TMS, la transferencia de información entre los bancos de memoria y la PC estará dentro de las especificaciones.

El análisis se concreta a determinar el tiempo máximo de acceso requerido para cada banco de memoria, considerando los tiempos máximos/mínimos de retraso ocasionados por la lógica de direccionamiento, selección y control para cada banco de memoria, así como los tiempos de acceso propios de la memora.

También se presenta un análisis sobre los tiempos típicos de retraso de las componentes para establecer un rango de valores que garantice un margen adecuado en la operación del sistema sin problemas de retraso de tiempo.

Para el análisis se parte del circuito electrónico (anexo B) haciendo diagramas de bloques que involucran a las componentes que intervienen en las selección y control de las memorias (figuras 4.5, 4.7, 4.9 y 4.11). En el interior de cada componente, se especifica el tiempo de retraso mínimo o máximo para la evaluación del peor de los casos. Externamente a cada cuadro se muestra el tiempo de retraso acumulado incluyendo el del propio dispositivo. Las flechas indican la trayectoria de las señal etiquetada con el mismo nombre del circuito electrónico. Asimismo, se parte del los esquemas de tiempos que da el fabricante para el TMS320C25, donde se toma como referencia la señal CLKOUT1 (que es 1/4 de la frecuencia de entrada X/2CLKIN) que genera internamente el TMS, el tiempo que dura una dirección valida, las señales /DS y /PS que se habilitan para el acceso a memoria dato o programa, el tiempo de un dato de salida o entrada y la señal /STRB que valida el ciclo de bus externo. Adicionalmente a esta señales se ha hecho un seguimiento de las señales que intervienen en la decodificación y selección de las memoria dibujando en el diagrama los tiempos de retardo que agregan dichas componentes.

En la selección de una memoria, dos señales resultan ser críticas: la de selección de la memoria /CS y la habilitación de la lectura o escritura del dato /OE o /WE, por lo que en los diagramas se centran en la determinación de estos tiempos. Cabe anotar que el tiempo de acceso de una memoria viene determinado por el tiempo que dura la dirección en su bus de direcciones, por lo que este tiempo de acceso puede ser mayor o igual que la habilitación de la lectura o la escritura de la memoria.

IV.5.1 Lectura de la memoria programa

Del diagrama de tiempos de la figura 4.6, se aprecia que el dato se presenta al menos 23 ns antes que la señal /SRTB tome un nivel alto, razón por la cual se ha utilizado esta señal

en la decodificación de las memorias. Tanto en los diagramas de bloques como en los de tiempos, se toma como referencia para tiempo cero la activación de la señal /STRB, ya que es la que valida la información, por lo que los retrasos que involucran la dirección válida, la señal de lectura/escritura o las habilitaciones de memoria dato o programa no se toma en cuenta debido a que se dan mucho antes que /STRB. Las señales de habilitación de la memoria /CEBAJ y /CEALT como la señal de habilitación del dato a leer /OEMP son bajas dentro de la ventana de tiempo para leer el dato:

Valores máximos

Tcs = 77 - 27 = 50 ns

Toe = 77 - 42 = 25 ns

Valores típicos:

Tcs = 77 - 14 = 63 ns

Tos = 77 - 24 = 53 ns

Esto que significa que el tiempo de acceso a la memoria debe ser hasta 25 ns y si se basa el cálculo en los tiempo típicos, el tiempo de acceso crece a 53 ns, tiempo suficiente para hacer la lectura en la memoria MP.

DIAGRAMA DE TIEMPOS PARA LA LECTURA DE LA MEMORIA PROGRAMA POR EL TMS

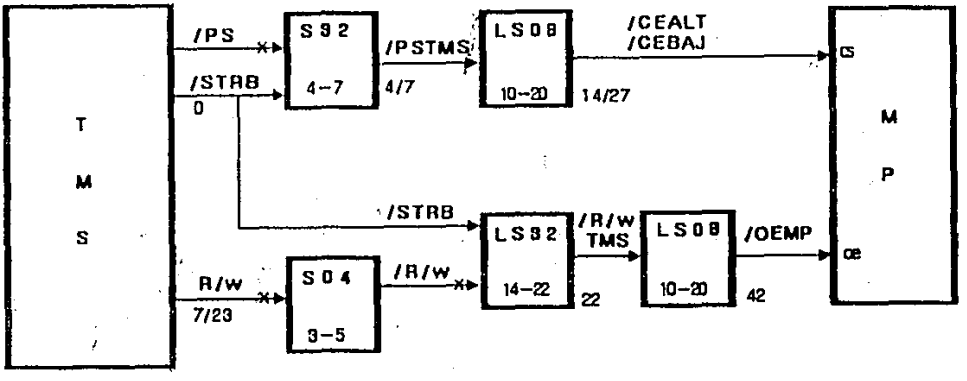


Figura 4.5

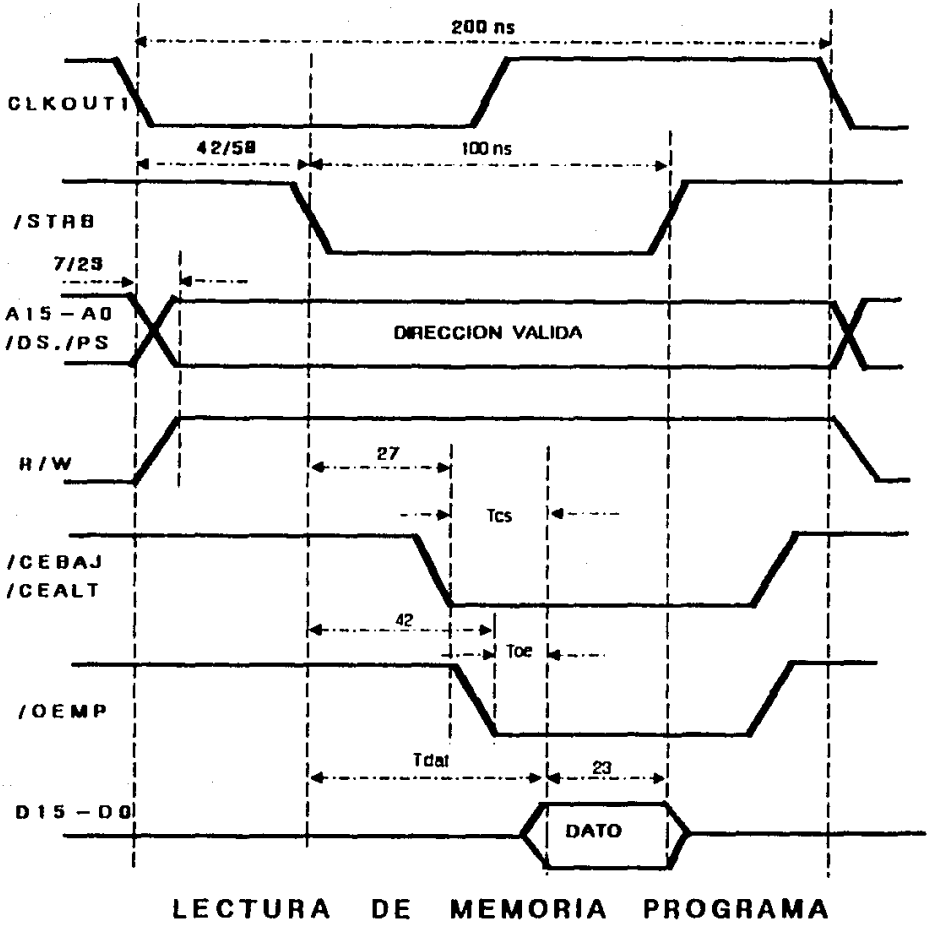

200 ns
CLKOUT1
42/58
100 ns
/STRB
7/23
A15 – A0
/DS./PS
DIRECCION VALIDA
R/W
27
Tcs
/CEBAJ
/CEALT
42
Toe
/OEMP
Tdat
23
D15 – D0
DATO
LECTURA DE MEMORIA PROGRAMA

Figura 4.6

Considerando que pueden existir varias unidades de procesamiento trabajando en paralelo, implica integrar al circuito un buffer bidireccional y la lógica de decodificación correspondiente para separar en el bus local del bus de datos de la APDI. Esto se debe a que cada código de instrucción debe leerse de la memoria MP ubicada en el módulo de interfase_PC. Debido a las características antes descritas, esta lógica involucra más compuertas, lo que significa un incremento en los tiempos de retardo.

Realizando un análisis de tiempos se obtiene un tiempo Tg', que corresponde al tiempo en el cual los datos son transferidos del bus de la APDI al bus local y es igual a:

$$Tg' = 149 \text{ ns}$$

Debido a que Tg' es mayor que el tiempo Tdat + 27 ns (fig. 4.8), no es factible determinar la velocidad de acceso a la memoria ya que los tiempos Tcs y Toe no son críticos al compararlos con Tg'. Por lo tanto, se requiere de tomar otras medidas como la inserción de estados de espera (wait states) en el TMS.

Considerando un ciclo de espera que corresponde a un ciclo de reloj, se tiene que:

$$Tdat = 77 + 200 = 277 \text{ ns}$$

de esta forma se obtienen lo valores de tiempo de acceso de memoria:

Valores máximos:

$Tcs = 277 - 27 = 250$ ns

$Toe = 277 - 42 = 235$ ns

Valores típicos:

$Tcs = 277 - 14 = 263$ ns

$Toe = 277 - 24 = 253$ ns

Al agregar el ciclo de espera, el tiempo Tg' está dentro del rango para poder efectuar la transferencia de datos.

DIAGRAMA DE TIEMPOS PARA EL TRASLADO DE UNA INSTRUCCION LEIDA EN MEMORIA PROGRAMA EN EL BUS LA APDI AL BUS LOCAL DEL TMS

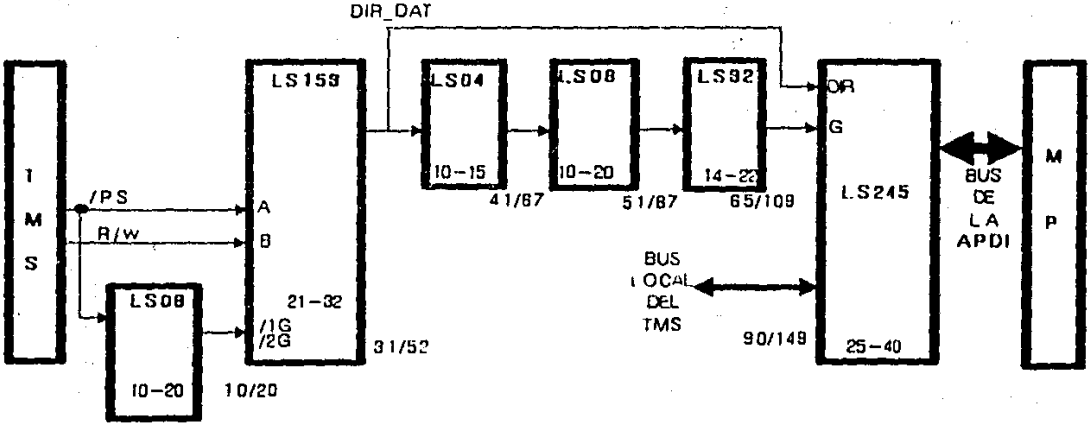


FIGURA 4.7

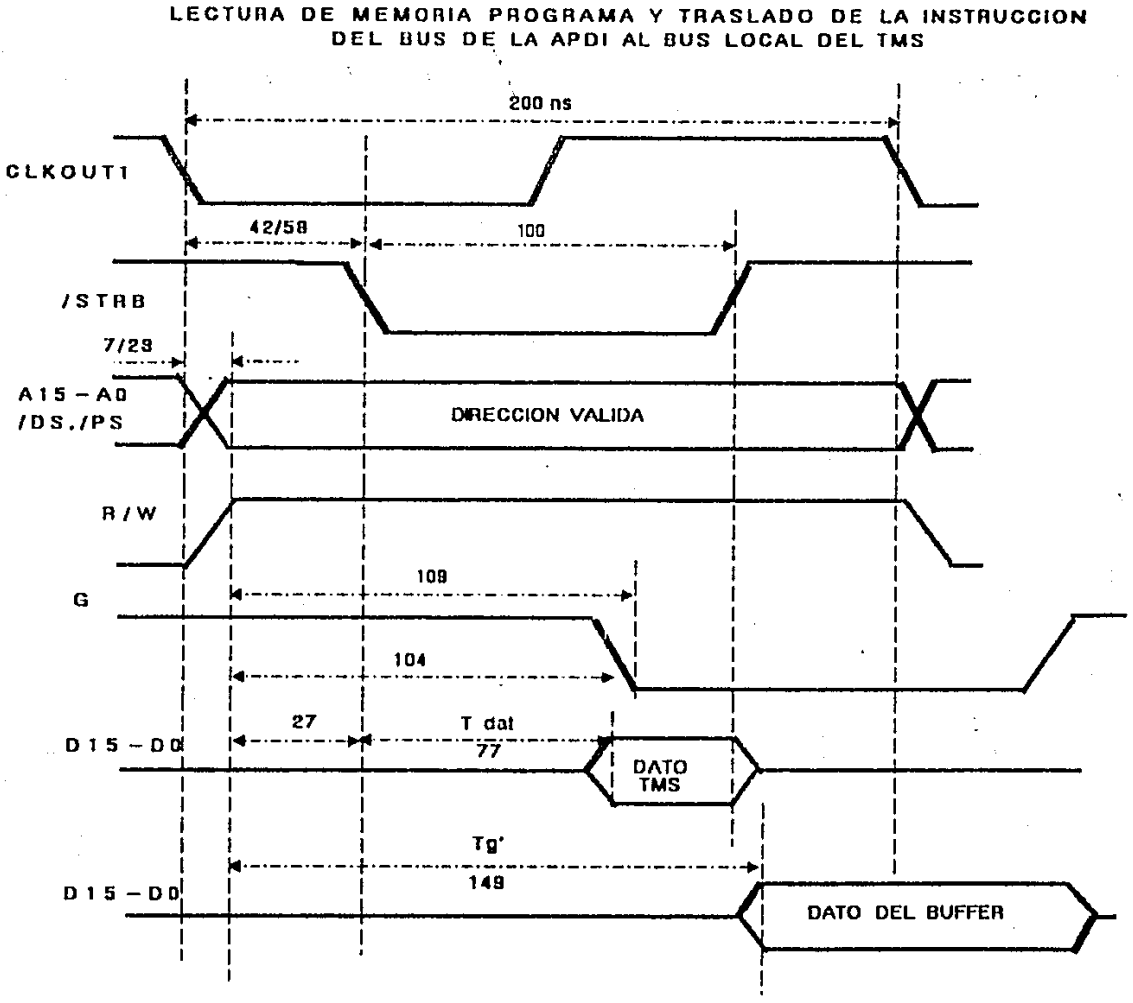

LECTURA DE MEMORIA PROGRAMA Y TRASLADO DE LA INSTRUCCION
DEL BUS DE LA APDI AL BUS LOCAL DEL TMS
200 ns
CLKOUT1
42/58
100
/STRB
7/29
A15 – A0
/DS./PS
DIRECCION VALIDA
R/W
109
G
104
27
T dat
77
D15 – D0
DATO
TMS
Tg'
149
D15 – D0
DATO DEL BUFFER

Figura 4.8

IV.4.2 Lectura de la memoria fuente en el bus local

De manera similar se pueden seguir las señales en la lógica de decodificación de la lectura de memoria fuente, llegándose a los tiempos:

Valores máximos:

Tcs = 77 - 24.5 = 52.5 ns

Toe = 77 - 7 = 70 ns

Valores típicos:

Tcs = 77 - 15.5 = 61.5 ns

Toe = 77 - 4 = 73 ns

Por lo que tenemos de nuevo un tiempo de acceso de 52.5 ns y para retrasos típicos en las componentes 61.5 ns.

DIAGRAMA DE TIEMPOS PARA LA LECTURA DE LA MEMORIA FUENTE POR EL TMS

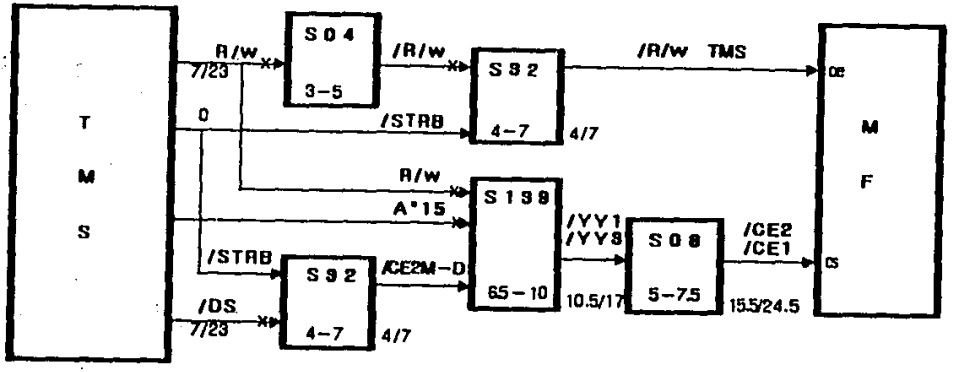


Figura 4.9

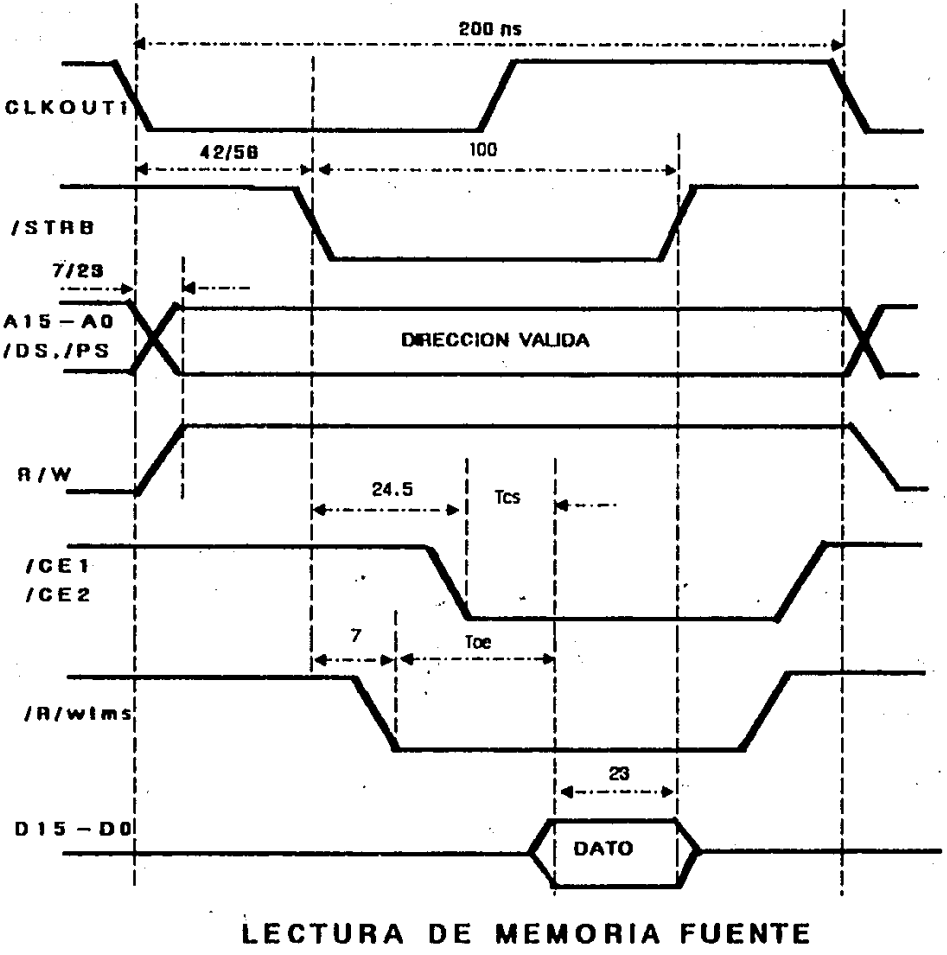

200 ns
CLKOUT1
42/58
100
/STRB
7/23
A15 – A0
/DS,/PS
DIRECCION VALIDA
R/W
24.5
Tcs
/CE1
/CE2
7
Toe
/R/wims
23
D15 – D0
DATO
LECTURA DE MEMORIA FUENTE

Figura 4.10

IV.5.3 Escritura en memoria resultado en el bus local

El análisis de estos tiempos lleva a considerar que el dato a escribir se manifiesta en el bus de datos 120 ns antes que termine el ciclo de escritura, tiempo dentro del cual debemos de hacer la selección de la memoria y control de escritura de la memoria, de los diagramas obtenemos:

Valores máximos:

Tcs = 117 - 24.5 = 92.5 ns

Twe = 117 - 17 = 100 ns

Obteniéndose un tiempo de acceso de 92.5 ns, suficientes para efectuar la escritura en la memoria MR.

DIAGRAMA DE TIEMPOS PARA LA ESCRITURA DE LA MEMORIA RESULTADO POR EL TMS

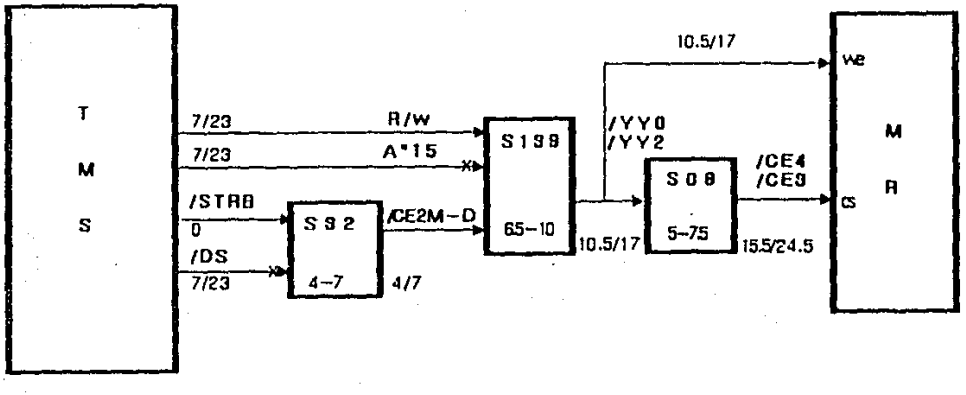


Figura 4.11

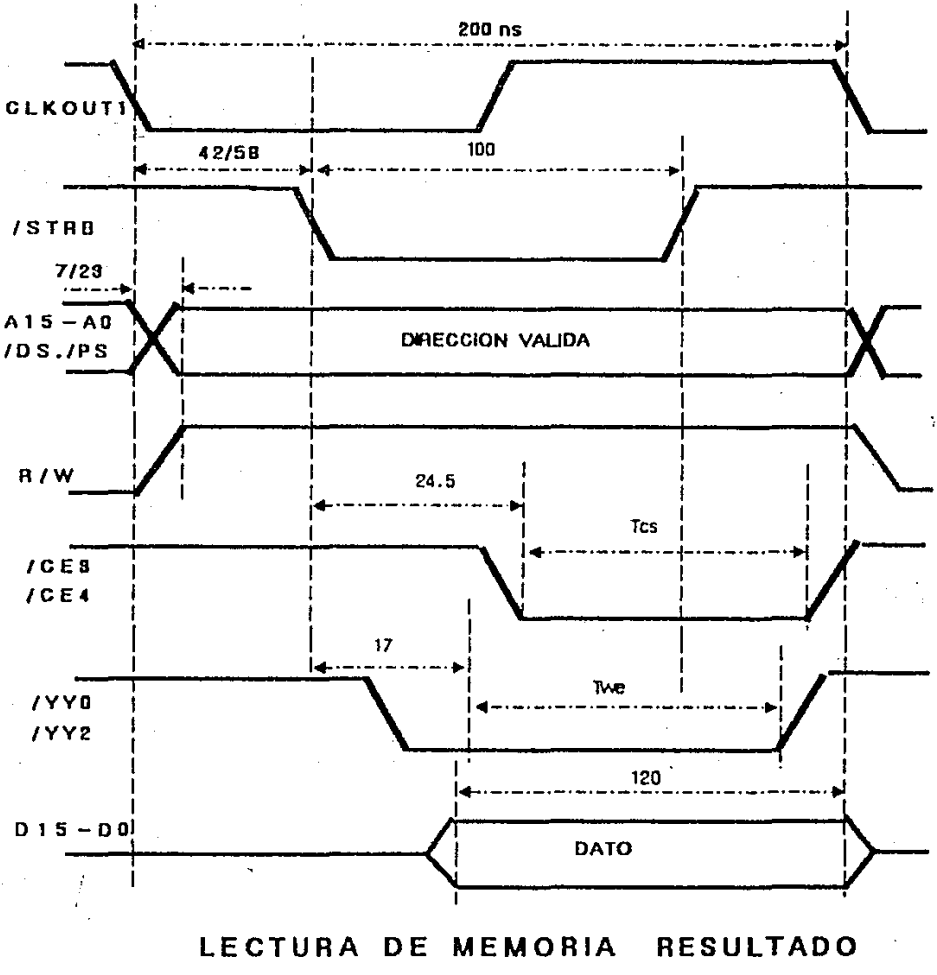

200 ns
CLKOUT1
42/58
100
/STRB
7/23
A15 – A0
/DS./PS
DIRECCION VALIDA
R/W
24.5
Tcs
/CE3
/CE4
17
Twe
/YY0
/YY2
120
D15 – D0
DATO
LECTURA DE MEMORIA RESULTADO

Figura 4.12

Del análisis se deduce que los tiempos de escritura en la memoria MR son suficientes para accesarla Tcs = 92.5 ns, de igual forma para la lectura en la memoria MF el tiempo es suficiente Tcs = 52.5 ns y para la lectura de la memoria MP se requiere presentar un dato en el buffer al menos 104 ns después de que el TMS haya mandado sus señales de control por lo que se hace necesario la utilización de un tiempo de espera. Este tiempo de espera permite extender el tiempo de acceso a la memoria en un ciclo de máquina (200 ns para la APDI), lo que significa que la señal de /STRB esté activa hasta 300 ns, este tiempo es introducido utilizando las señales del /SRTB y /MSC del TMS. La señal /MSC es activada por el TMS durante el flanco de bajada del reloj CLKOUT1, e indica el inicio de una operación interna o externa de memoria o E/S. Si la señal de entrada READY es activada baja, el TMS extiende el ciclo de memoria en un ciclo de máquina manteniendo las señales restantes válidas.

Con el estudio realizado sobre los tiempos de acceso a memorias, se tiene que el tiempo de acceso debe ser menor o igual a 52.5 ns. Este valor es mayor con repecto al tiempo de acceso de las memorias empleadas en la APDI que es de 45 ns.

El tiempo de espera agregado a la APDI junto con el tiempo de acceso de las memorias utilizadas posibilitan una transferencia de información adecuada.

V UN PROGRAMA DE APLICACION PARA LA ARQUITECTURA PDI

V.1 INTRODUCCION

El hardware de un sistema es la base sobre la cual se programa al mismo. Por ejemplo, para comprender las funciones de cualquier sistema es importante conocer los componentes del hardware y de esta forma se sabrán las diferentes facilidades disponibles. Estos conocimientos permiten realizar un mejor manejo de los recursos del sistema.

Los sistemas electrónicos programables requieren de una serie ordenada de instrucciones para realizar cualquier tarea, es decir, una plataforma que proporcione todas las facilidades de programación y modificación para el sistema. Dicha plataforma provee un lenguaje que facilite el desarrollo de la aplicación.

En todo lenguaje de programación se tiene un conjunto de reglas asociadas que describen cómo puede ser construido un programa dentro del ambiente del lenguaje. Estas reglas son necesarias para que el programador puede describir y corregir el programa, y que a su vez éste pueda ser entendido por otro programador. Las normas de estructuración están constituidas por la sintaxis (vocabulario) y la semántica (sentido y significado) del lenguaje.

Un programa de aplicación es un conjunto de instrucciones, regidas por las reglas de estructuración de un lenguaje de programación con el propósito de ejecutar diversas tareas, tales como: implementar algoritmos matemáticos, crear archivos de datos, programar periféricos, etc.

El desarrollo adecuado de un programa (software) permite el mejor aprovechamiento de los recursos disponibles de un sistema electrónico. Para poder discriminar la programación más conveniente, se recurre a la evaluación del funcionamiento global del sistema (hardware y software), en base a la ejecución de diversas rutinas de programación y en búsqueda de la más apropiada.

V.2 UTILIZACION DE UN LENGUAJE DE ALTO NIVEL: TURBO PASCAL, PARA EL DESARROLLO DE RUTINAS Y PROGRAMAS

La elección de un lenguaje de programación depende de las ventajas que ofrece y de las necesidades a satisfacer. El empleo de un lenguaje de alto nivel para el desarrollo de rutinas y programas, permite una programación estructurada, modular y sistemática.

Turbo Pascal (TP) es un lenguaje de programación que cumple con tales características, y probablemente es en la actualidad uno de los lenguajes de programación más usados en las microcomputadoras.

Dentro de las características sobresalientes o más importantes de Turbo Pascal se encuentran las siguientes :

- En TP se pueden definir constantes como parámetros para procedimientos y funciones.
- TP simplifica la entrada y salida de datos con dispositivos externos permitiendo que los programas los traten como variables o archivos.
- Comprobación de errores después de cada operación de Entrada/Salida.
- Ambiente de programación completo.
- Depuración de programas.
- Ejecución paso a paso con capacidad de observar/modificar valores de las variables.
- Amplia bibliografía y programas.
- Recursos para desarrollar rutinas normalizadas para interfaz de usuario.
- Uso de los servicios del BIOS en ROM y del DOS.
- Estructuras de control.
- Rutinas de gráficos.
- Manejo de unidades de rutinas precompiladas.

Por lo anterior se ha seleccionado TP en el desarrollo de las rutinas de programación de la APDI. Con ello se ha buscado crear una biblioteca de rutinas que cumplan con los atributos y normas siguientes :

a) Utilización congruente de los recursos disponibles.

b) Documentación completa de su funcionamiento.

c) Codificación estructurada.

V.3 DESARROLLO DE UN PROGRAMA DE APLICACION PARA EL APOYO Y MANEJO DE LA APDI

El desarrollo de cualquier programa implica el empleo de tres pasos esenciales :

a) Especificar la labor que el computador realizará, en cuanto a la entrada y salida de datos durante su ejecución.

b) Plantear un algoritmo o secuencia de pasos, por los cuales el computador pueda producir los resultados deseados a partir de los de entrada.

c) Expresar este algoritmo en el lenguaje de programación tal como TP.

De acuerdo a éstas etapas, primero se deben especificar las tareas a realizar por la rutina en la administración de los recursos de la APDI. Las funciones a efectuar por el programa son las siguientes :

- Establecer las condiciones para la transferencia de información entre la PC y la APDI, mediante la creación de un código.
- Transferir el programa en lenguaje ensamblador TMS320C25 al bloque de memoria programa, a partir de un archivo proporcionado por el programa ensamblador del TMS320C25. Esto lo efectúa a través de un procedimiento que lee los códigos de un archivo].[MPO, asocia la dirección respectiva con su código de operación respectiva, dividiendo en dos el código para enviarlo en el orden correspondiente a la memoria programa.
- Proporcionar una interfaz con el usuario por medio de un menú de opciones para la elección de la imagen a procesar y el tipo de proceso a realizar en la misma. Este menú es necesario para minimizar los errores del usuario en el uso de la rutina.
- Transferir los datos correspondientes a las imágenes fuente y resultado, ya sea desde un archivo en disco a la memoria MF, o bien, de la memoria MR a un archivo resultado o monitor.

Con estas funciones se pretende realizar el control de la APDI y a su vez evaluar el funcionamiento de la arquitectura propuesta.

El siguiente diagrama de bloques presenta esquemáticamente las tareas antes descritas (figura 5.1).

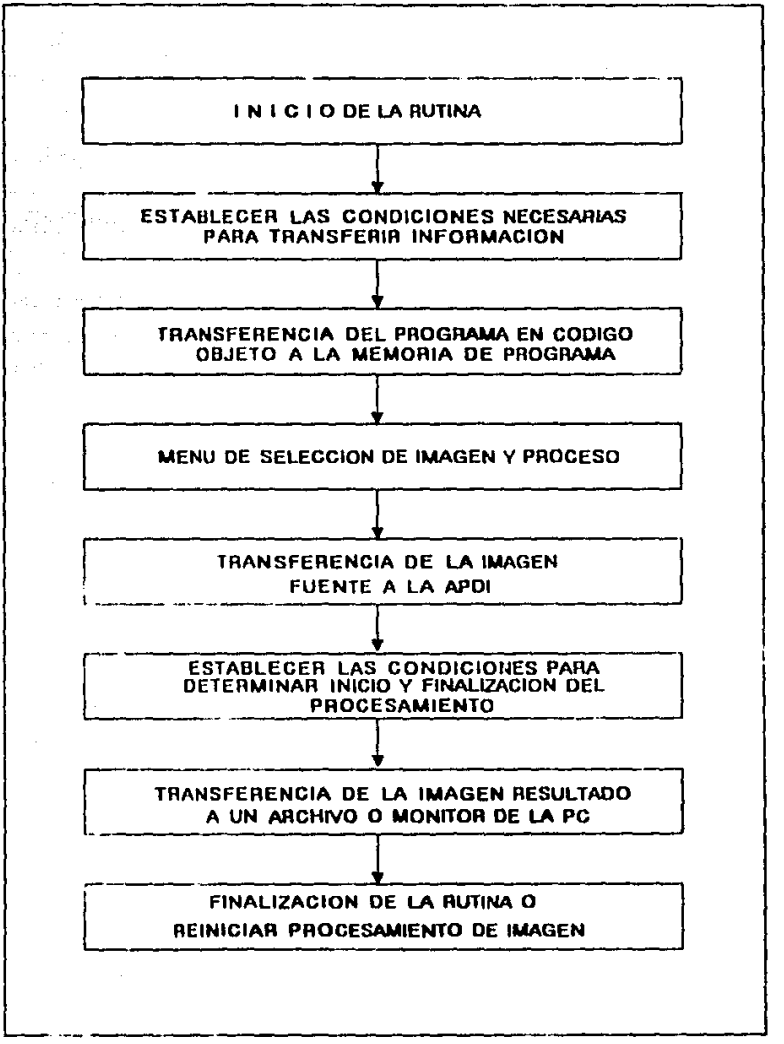


Figura 5.1 Diagrama de bloques del programa de Aplicación.

La implementación de estas funciones de acuerdo al diagrama de bloques previo, es realizada en un rutina en TP versión 5.0, con el nombre de PROAPLIC.PAS. En esta rutina se tiene un programa principal, el cual esta constituido por una serie de subrutinas, mismas que realizan ciertas funciones cuya finalidad es la de ejecutar las tareas ya establecidas. La descripción del programa está contenida dentro del listado del mismo (Ver Anexo C).

V.4 LENGUAJE ENSAMBLADOR DEL TMS320C25 (TMS)

V.4.1 Creación del código fuente en lenguaje ensamblador

Para la creación del Código Fuente empleado en la operación del TMS, se cuenta con el software que provee el fabricante, el Macro Ensamblador TMS320C25, que permite hacer la conversión de las instrucciones en mnemónicos y directivas específicas en código fuente a código objeto ejecutable. Este software, al tener el programa en código objeto, permite efectuar una simulación del mismo, o bien a través de un programa simulador tener la opción de depurar el mismo.

El lenguaje ensamblador del TMS consiste de códigos de operación llamados mnemónicos, que corresponden a las directivas de las instrucciones del lenguaje binario de máquina. Un programa en lenguaje ensamblador es llamado Programa Fuente y antes de poder ser ejecutado por el procesador, el programa fuente debe de ser ensamblado para obtener el programa en código de máquina. Cuando se edita un programa, se escribe un archivo].[ASM, a través del ensamblador se genera un archivo].[LST, el cual contiene el archivo editado más una columna de direcciones y una de códigos; además se genera el archivo].[MPO el cual contiene las direcciones con sus correspondientes códigos de operación en código hexadecimal.

V.4.2 El programa ensamblador

El ensamblado se realiza en dos pasadas. En la primera el ensamblador mantiene un contador de localización, el cual define la dirección de memoria programa asignada a la palabra

resultante en código objeto. En la segunda pasada, el ensamblador produce el código objeto que corresponde al código de operación asignándole su respectiva palabra.

Un programa fuente en lenguaje ensamblador, consiste de expresiones que pueden contener directivas, instrucciones de máquina o comentarios. Estas expresiones son examinadas por el ensamblador y pueden contener cuatro campos:

- Etiquetas
- Comandos
- Operandos
- Comentarios

cada una separada por uno o más espacios blancos. Las expresiones que contienen un asterisco (*) en la primera columna, corresponden a comentarios y no afectan el programa. Una línea de una expresión fuente, puede ser tan larga como el formato fuente lo permita, sin embargo, el ensamblador trunca las líneas a 60 caracteres, por lo que sólo los comentarios pueden extenderse más allá de la columna 60 sin ocasionar error. El ensamblador utiliza caracteres ASCII.

Sintaxis de una expresión fuente:

[<ETIQUETA>| <MNEMONICO> [<OPERANDO>][<COMENTARIO>]

V.4.2.1 Campo de la etiqueta:

Inicia en la primera columna de la expresión y puede contener hasta 6 caracteres, de los cuales el primero debe ser una letra, el resto puede ser alfanumérico. La etiqueta es opcional y se utiliza muy frecuente para indicar hacia donde debe transferir el control del programa bajo cierta decisión. Cuando no se utiliza etiqueta, al menos la primera columna debe dejarse vacía.

Una expresión que consiste de sólo una etiqueta es válida, ya que a través de esta posibilidad, a la etiqueta se le asigna una constante numérica, ej.

<etiqueta> EQU >número

donde el número puede representa el valor de localización del contador, permitiendo definir algunas variables a utilizar dentro del programa.

V.4.2.2 Campo del comando

Inicia después de un espacio en blanco al terminar el campo de la etiqueta. En el caso de no existir etiqueta el comando inicia después de un espacio en blanco. El campo del comando es terminado por uno o más espacios en blanco y no debe extenderse más allá del margen derecho.

El campo del comando puede contener:

- Mnemónicos del ensamblador o una instrucción de máquina.
- Macromnemónicos.
- Directivas del ensamblador.

V.4.2.3 Campo del operando

Inicia después de un espacio en blanco dejado por el campo del comando. El campo del operando puede contener :

- constantes.
- una cadena de caracteres.
- expresiones.

Este campo no debe extenderse más allá del margen derecho de la expresión fuente. Los símbolos utilizados en el campo de operando deben de ser definidos en el ensamblador, usualmente por etiquetas.

V.4.2.4 Campo de comentarios

Inicia después de un espacio en blanco al terminar el campo del operando o el espacio en blanco al terminar el campo del comando. Este campo puede extenderse hasta el final de la expresión fuente, los comentarios no tienen efectos en el ensamblado. Usualmente se escribe un punto y coma (;) antes del comentario de una expresión fuente, esto permite diferenciar donde empiezan los comentarios de la expresión. Los comentarios pueden contener todo tipo de caracteres y espacios.

V.4.3 Modos de direccionamiento de memoria

El TMS soporta un amplio set de instrucciones para cálculos numéricos intensivos empleados en operaciones de procesamiento de señales y aplicaciones de propósito general, tales como multiprocesamiento y el control a alta velocidad.

A continuación se describen algunas instrucciones del lenguaje ensamblador del TMS, así como los modos de direccionamiento que se han utilizado en la elaboración del programa de procesamiento que ejecuta el TMS:

- Direccionamiento directo.
- Direccionamiento indirecto.
- Direccionamiento inmediato.

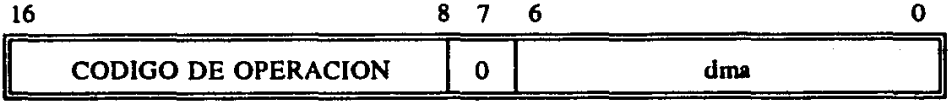
V.4.3.1 Direccionamiento directo

En este modo de direccionamiento, la palabra de instrucción contiene siete bits del direccionamiento de memoria dato (dma). Estos bits son concatenados con los nueve bits del Registro Apuntador de Página (DP), de tal forma que se completa una dirección de memoria dato de 16 bits. El registro DP apunta a una de las 512 páginas posibles que contiene 128 palabras cada una. Los siete bits en la instrucción apuntan a una localización específica dentro de la página. El registro DP, puede ser cargado por las instrucciones[1]:

LDP	carga un valor de memoria dato en el registro DP. Ej. LDP DAT15
LDPK	carga una constante de 9 bits en el registro DP. Ej. LDPK 90
LST	El registro status ST0 es cargado con la dirección de un valor de memoria dato, donde los ocho bits menos significativos (LSB) corresponden al registro DP.

El registro DP no es inicializado cuando se enciende el sistema, por lo que siempre se tiene que inicializar por software.

Formato del direccionamiento directo:



El bit 7 = 0, define el modo de direccionamiento directo, los bits 0 al 6, contienen el direccionamiento de memoria dato.

---

[1] En una instrucción, cuando se refiere a un número hexadecimal, se le antepone el símbolo > cuando se trata de un número decimal no lleva ningún símbolo

V.4.3.2 Modo de direccionamiento indirecto

Para este modo de direccionamiento, se utilizan los Registros Auxiliares (AR), los cuales proveen una gran flexibilidad y un poderoso direccionamiento indirecto. El TMS dispone de ocho registros auxiliares: AR0 al AR7. Para seleccionar cualquiera de ellos, el Apuntador de Registros Auxiliares (ARP), es cargado con un valor de 0 a 7.

El contenido del registro auxiliar puede ser operado por medio de la Unidad Aritmética de Registros Auxiliares (ARAU). La ARAU efectúa operaciones aritméticas sobre los Registros auxiliares en el mismo ciclo de la instrucción.

En el direccionamiento indirecto, cualquier localidad de las 64 Kb en el espacio de Memoria Dato, puede ser accesada por medio de los 16 bits de dirección contenidos en el AR en uso. Estos pueden ser cargados por las instrucciones:

LAR	Carga el registro auxiliar especificado, con el contenido de un dato de memoria direccionada. Ej. LAR AR1,>DAT
LARK	Los ocho bits de una constante positiva son cargadas en el registro auxiliar indicado. Ej. LARK AR4,>10
LRLK	16 bits inmediatos de la instrucción, son cargadas en el registro auxiliar especificado. La constante de 16 bits no debe ser signada. Ej. LRLK AR3,>CTE.
ADRK	Una constante inmediata de 8 bits, es sumada (justificada a la derecha, es decir justificado al LSB) al AR en uso, el valor de AR es reemplazado por el valor de suma. Ej. ADRK >20.
SBRK	Una constante inmediata de 8 bits, es restada (justificada a la derecha) al AR en uso, el valor de AR es reemplazado por el valor de la resta. Ej. SBRK >15.
MAR	Actúa como una no-operación, modifica AR y ARP, cambiando del AR en uso al nuevo indicado. Ej. MAR *,1 ;el registro AR en uso (*) es cambiado a AR1, y ARP cambia al valor de uno.

En las operaciones de direccionamiento indirecto, el AR en uso puede modificarse en el mismo ciclo de instrucción, sumándole ó restándole uno.

V.4.3.2.1 Simbología para direccionamiento indirecto

* El contenido de AR(ARP) es utilizado como el dato de memoria direccionado, es decir el contenido del AR en uso.

*- El contenido de AR(ARP) utilizado como el dato de memoria direccionado, es decrementado después de ser accesado.

*+ El contenido de AR(ARP) utilizado como el dato de memoria direccionado, es incrementado después de ser accesado.

*0- Al contenido de AR(ARP) utilizado como el dato de memoria direccionado, se le resta el contenido de AR0 después de ser accesado.

*0+ Al contenido de AR(ARP) utilizado como el dato de memoria direccionado, se le suma el contenido de AR0 después de ser accesado.

*BR0- El contenido de AR(ARP) utilizado como el dato de memoria direccionado, se le sustrae el contenido de AR0 con la propagación del carry reservado (rc), después de ser accesado.

*BR0+ El contenido de AR(ARP) utilizado como el dato de memoria direccionado, se le suma el contenido de AR0 con la propagación del carry reservado (rc), después de ser accesado.

Existen dos tipos principales de direccionamiento indirecto con indexamiento:

- Direccionamiento indirecto regular con incremento y decremento.
- Direccionamiento indirecto con indexamiento basado en el valor de AR0: Indexado por la suma o resta del contenido de AR0, o indexando por la suma o resta de el contenido de AR0 con la propagación de carry reservado.

En cada uno de los casos el contenido del AR apuntado por ARP es utilizado como la dirección de la memoria dato operando. La ARAU efectúa la operación matemática especificada sobre el AR. Todas las operaciones de indexamiento son ejecutadas sobre el AR utilizado en el mismo ciclo de la instrucción.

El direccionamiento indirecto, puede ser utilizado con todas las instrucciones, excepto con instrucciones de operando inmediato e instrucciones con no operandos.

Formato de este modo de direccionamiento indirecto :

15 — 8	7	6	5	4	3	2 — 0
CODIGO DE OPERACION	1	IDV	INC	DEC	NAR	Y

El bit 7 = 0, define el modo de direccionamiento indirecto, los bits 6 a 0 contienen el control del direccionamiento indirecto.
El bit 6, determina las operaciones de incremento/decremento. Si es 0, el AR en uso es incrementado o decrementado, si es 1, AR0 puede ser sumado o restado del AR en uso.
Los bits, 5 y 4 determinan las operaciones aritméticas a ser ejecutadas por AR(ARP) y AR0, como se indica en la tabla:

BITS 6 5 4	OPERACIONES ARITMETICAS
0 0 0	No operación.
0 0 1	AR(ARP) - 1 ===> AR(ARP)
0 1 0	AR(ARP) + 1 ===> AR(ARP)
0 1 1	Reservado
1 0 0	AR(ARP) - AR0 ===> AR(ARP) [reservado a rc]
1 0 1	AR(ARP) - AR0 ===> AR(ARP)
1 1 0	AR(ARP) + AR0 ===> AR(ARP)
1 1 1	AR(ARP) + AR0 ===> AR(ARP) [reservado a rc]

El bit 3 (NAR), determina si son cargados nuevos valores dentro de ARP, si el bit 3 = 1, el contenido de los bits 2 a 0 (Y = nuevo ARP) es cargado dentro de ARP, si bit 3 = 0, el contenido de ARP permanece sin cambios.

V.4.3.3 Modo de direccionamiento inmediato

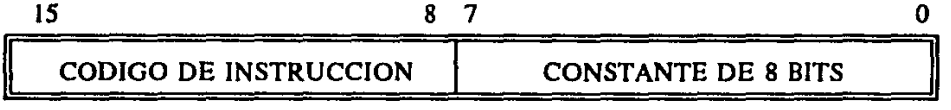
En este modo, utiliza instrucciones que contienen palabras, el valor de la palabra es cargado inmediatamente en el operando. El TMS, tiene la posibilidad de hacer cargas cortas inmediatas para palabras simples (8 bits) y una carga larga inmediata para palabras de 16 bits. En la carga larga inmediata, la palabra que sigue a la instrucción del código de operación es utilizada como el código operando inmediato, esto significa que el código de operación está compuesto de dos palabras de 16 bits.

Las siguientes instrucciones de carga corta inmediata, contienen en el código de operación la palabra, por lo que la carga se ejecuta en el mismo ciclo de instrucción.

ADDK Suma a acumulador una constante corta inmediata.
ADRK Suma al AR en uso una constante corta inmediata.

LACK	Carga al acumulador una constante corta inmediata.
LARK	Carga al AR en uso una constante corta inmediata.
LARP	Cargan al ARP una constante inmediata de 3 bits.
LDPK	Carga al DP una constante inmediata de 9 bits.
MPYK	Multiplicación inmediata del registro T con una constante de 13 bits.
RPTK	Carga al registro de repetición RPTC una constante de 8 bits.
SBRK	Resta del AR en uso una constante de 8 bits.
SUBK	Resta del acumulador una constante corta inmediata de 8 bits.

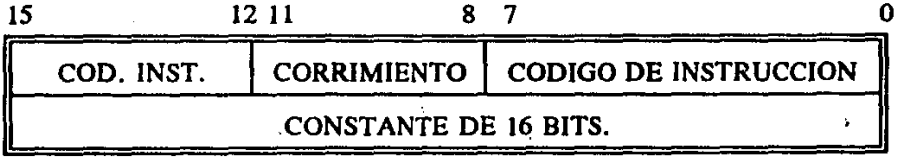
En el formato para direccionamiento inmediato de una instrucción que utiliza una constante de 8 bits, el código de operación queda formado por:



Instrucciones de carga larga inmediata:

ADLK	Suma al acumulador una constante larga inmediata con corrimiento.
ANDK	Efectúa una operación AND inmediata con el acumulador con corrimiento.
LALK	Carga al acumulador una constante larga inmediata con corrimiento.
LRLK	Carga al registro AR en uso una constante larga inmediata.
ORK	Efectúa una operación OR inmediata con el acumulador con corrimiento.
SBLK	Resta del acumulador una constante larga inmediata con corrimiento.
XORK	Efectúa una operación XOR inmediata con el acumulador con corrimiento.

El formato para el modo de direccionamiento inmediato con una constante larga es:



V.4.4 Set de instrucciones

En el anexo E se presenta la lista de instrucciones mnemónicos correspondientes al TMS320C25.

V.5 DESARROLLO DE UN PROGRAMA BASADO EN ALGORITMOS DE PROCESAMIENTO DIGITAL DE IMAGENES (PDI)

Con el objeto de desarrollar un programa de procesamiento capaz de ejecutar el algoritmo de la convolución de una Matriz Imagen con otra matriz plantilla de 3X3 se explica el funcionamiento del algoritmo con las ventajas que ofrece el set de instrucciones del TMS.

V.5.1 Explicación del programa de procesamiento

El programa consta de varios bloques (ver programa en el anexo D):

1) Definición de localidades para el manejo de variables y contantes.
2) Carga de los valores de las plantillas en sus respectivas localidades.
3) Carga de los valores de las constantes de inicio y finalización para la unidad de procesamiento.
4) Identificación de la configuración del módulo de procesamiento.
5) Lectura del procesamiento a realizar.
6) Ejecución del proceso seleccionado.

V.5.1.1 Definición de localidades para el manejo de variables y constantes

Utilizando etiquetas y la instrucción de ensamblador EQU, el ensamblador define las localidades de las variables, constantes y puertos que se están utilizando en el transcurso del programa. Esta definición inicial nos permite hacer cambios directamente en este bloque que se reflejan en todo el programa.

V.5.1.2 Carga de los valores de las plantillas en sus respectivas localidades

Los valores de las cinco plantillas a utilizar (una para cada proceso) son cargados en memoria a partir de la localidad >0020 de la memoria de programa (memoria externa), utilizando las instrucciones de ensamblador: AORG para posicionar la dirección y DATA para cargar valores de constantes en las localidades consecutivas.

V.5.1.3 Carga de los valores de las constantes de inicio y finalización para la unidad de procesamiento

El apuntador de página se ubica en la página 6 para el manejo de localidades de variables, ya que esta página se ubica en el mapa de memoria como el bloque B1 y se utiliza como memoria RAM en la memoria de dato. Las constantes cargadas determinan las veces que se ejecuta el algoritmo, tomando en cuenta el número de unidades que se van a utilizar. Las cargas se realizan haciendo un carga larga inmediata en el acumulador y luego salvando éste en las localidades direccionadas por las constantes de inicio y finalización (CTEIxx y CTEFxx). Estas constantes indican los límites de un bloque de pixels a procesar dentro de una imagen.

V.5.1.4 Identificación de la configuración del módulo de procesamiento

El TMS efectúa una lectura de puerto para detectar con cuantas unidades de procesamiento cuenta el módulo (1,2 o 4) y de qué unidad de procesamiento se trata. Al efectuar la lectura en puerto, lee la programación de los switches S7, S6, S5 y S4, le hace un corrimiento a la derecha al dato, efectúa una operación AND para poner en ceros los bits D3, D2, D1 y D0, que en este momento no interesan, luego se le suma a este resultado un offset de >0300, que corresponde al valor inicial de la página 6 donde se ubican las direcciones de las variables. Las direcciones elegidas de antemano para las constantes iniciales y finales se han calculado de acuerdo a este corrimiento.

V.5.1.5 Lectura del procesamiento a realizar

En este bloque del programa el TMS efectúa de nuevo una lectura en el mismo puerto anterior, efectuando operaciones AND en eliminación de los bits correspondientes a los switches y la determinación del proceso solicitado por el usuario, dependiendo de este proceso el programa efectúa saltos al número de proceso correspondiente etiquetado con PROx, donde efectúa un movimiento de la plantilla X grabada en la memoria de programa a la página 4 de la memoria de datos (>0200) y luego salta a EPRO1 (procesos 1,3,4 y 5) o EPRO2 (proceso 2).

V.5.1.6 Ejecución del proceso seleccionado

Convolución entre una matriz imagen y una plantilla

Una vez que el programa ha seleccionado el proceso a ejecutar, éste se va un bloque de procesamiento. Cada uno de estos bloques se explica en el listado del programa en lenguaje

ensamblador (anexo D) en base a las instrucciones utilizadas. A continuación se explica como se realiza la ejecución del algoritmo por parte del programa, recurriendo a la presentación matricial de una imagen.

Considerando una Matriz Imagen Fuente (A) de (M+1) x (N+1) pixels, con $N+1=M+1=200$ y una matriz plantilla (B) de 3 x 3 (9 valores), entonces, el proceso de convolución consiste en ir sobreponiendo la plantilla de nueve valores a nueve pixeles de la matriz A. Los valores así sobrepuestos (la plantilla con los correspondientes valores de los pixeles) son multiplicados entre sí. Estos productos se suman para obtener el valor calculado correspondiente a la posición central de los nueve valores en la imagen.

El proceso se inicia desde la esquina superior izquierda, hasta la inferior derecha de la matriz A. Cabe señalar que los valores ubicados en las orillas de la imagen no pueden ser evaluados, debido a los efectos de borde. Después de haber evaluado el valor de un pixel, se desplaza la plantilla un pixel a la derecha, para volver a hacer un nuevo cálculo. Cuando se llega al extremo derecho de la imagen, se sitúa la plantilla en un nuevo renglón a partir de la primera columna.

Considerando en general una ventana de 3 x 3 pixeles de la imagen como se aprecia en la figura 5.2, si en la esquina superior izquierda de esta matriz se tiene el pixel a_{ij}, entonces se puede calcular el valor de la operación de convolución para el pixel central de la ventana $a_{i+1,j+1}$. El resultado de esta operación corresponde a un valor resultado para una nueva matriz (C), el cual queda:

Ecuación 1

$$C_{ij} = a_{ij}b_{11} + a_{ij+1}b_{12} + a_{ij+2}b_{13} + a_{i+1j}b_{21} + a_{i+1j+1}b_{22} + a_{i+1j+2}b_{23} + a_{i+2j}b_{31} + a_{i+2j+1}b_{32} + a_{i+2j+2}b_{33}$$

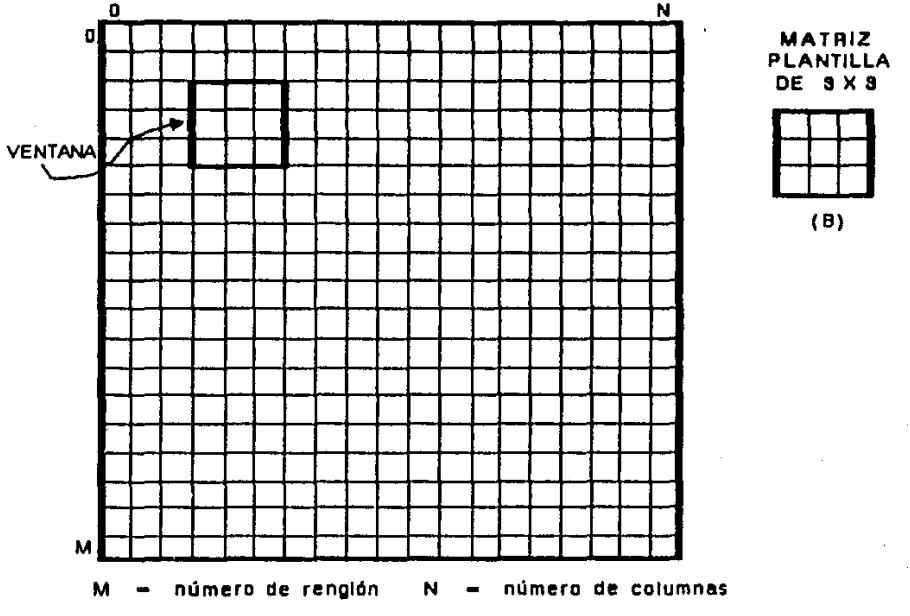

MATRIZ IMAGEN FUENTE A de M+1 X N +1 pixels
0
N
0
M
VENTANA
MATRIZ PLANTILLA DE 3 X 3
(B)
M - número de renglón N - número de columnas
CONVOLUCION DE MATRIZ A CON B
b11 b12 b13
b21 b22 b23
b31 b32 b33
ai
ai+1
ai+2
ai+1
ai+1+1
ai+1+2
ai+2
ai+2+1
ai+2+2

FIGURA 5.2

Agrupando los productos que corresponden a un mismo renglón, quedan las sumatorias con variaciones únicamente de una columna a otra:

$$c_{i+1j+1} = \begin{pmatrix} \sum_{J=0}^{J=2} a_{ij+J} b_{1j} + \\ \sum_{J=0}^{J=2} a_{i+1j+J} b_{2j} + \\ \sum_{J=0}^{J=2} a_{i+3j+J} b_{3j} \end{pmatrix}$$

Ecuación 2

Al implementar el algoritmo con una la matriz de 200 x 200 pixeles (40 K pixeles), la convolución implica realizar nueve multiplicaciones y nueve sumas por cada pixel resultado tal como se observa en las sumatorias. Esto significa que para procesar la matriz anterior se tienen que realizar 78,408 multiplicaciones y sumas más las operaciones de E/S de los valores de memoria y el manejo de registros.

Para el procesamiento de esta matriz en el tiempo que dura un cuadro al desplegarse en el margen de video (33 ms), implicaría un tiempo de proceso de 750 ns/pixel. Considerando las 18 operaciones por pixel y un ciclo de máquina de 100 ns, se tendrían entonces siete ciclos de máquina para procesar cada pixel, que resulta insuficiente debido a que no sólo se tienen que efectuar estas operaciones básicas, sino que además se tienen que efectuar 18 lecturas por cada pixel, una escritura, operaciones de control y manejos de memoria para registros, lo que vendría a sumar adicionalmente unas 25 instrucciones para la manipulación de cada pixel.

ESCRITURA DE LA MATRIZ A EN LA MEMORIA MF

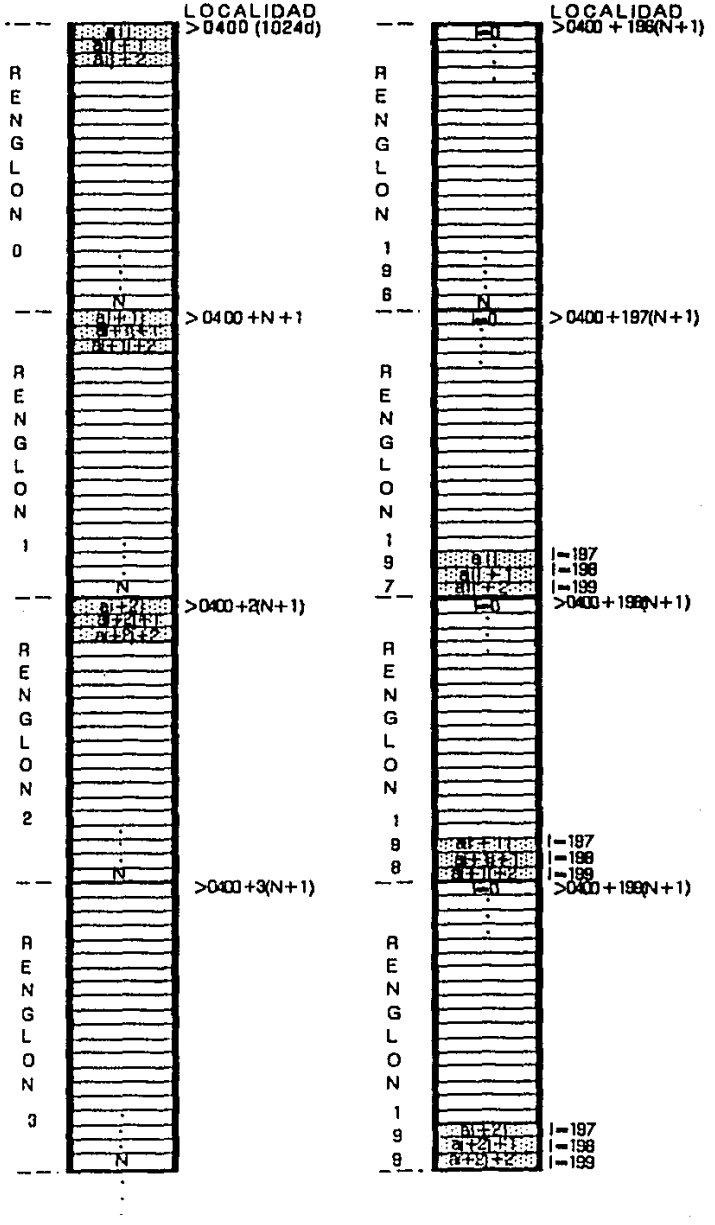


FIGURA 5.3

En la implementación del algoritmo se inició utilizando instrucciones básicas del ensamblador, lo que ocasionaba altos tiempos de procesamiento, sin embargo, el programa se fue depurando al optimizar el algoritmo y los tiempos, hasta lograr la reducción en la cantidad de las instrucciones a manejar, haciendo posible utilizar instrucciones especiales del TMS, donde se aprovecha las características del flujo de datos para el manejo de pipeline.

Para el manejo de la memoria, hay que tener en cuenta el orden en que se han escrito los datos en la memoria MF, ya que en apariencia la implementación del algoritmo desde el punto de vista matricial conllevaría a la aplicación de tres loops anidados y el problema quedaría resuelto, sin embargo, viéndolo desde la escritura en la memoria MF, el algoritmo tiene que tomar en cuenta el modo de almacenamiento de los datos y su interrelación matricial.

La figura 5.3 corresponde a la escritura de la matriz en la memoria fuente. Se observa que la memoria ha sido escrita en orden secuencial desde el renglón cero hasta el renglón 199 (donde i=0 hasta 199 por cada renglón). Para procesar el primer pixel, es necesario tomar los primeros tres valores del renglón cero multiplicándolos por los valores de posición correspondiente en la plantilla, luego saltar a leer los tres primeros valores del renglón uno y para concluir con ese pixel saltar al renglón dos y leer los tres primeros pixeles, el resultado, escribirlo en el primer pixel de la primera línea de la memoria MR.

Una vez terminando con el primer pixel, hay que desplazarse un pixel en cada línea para efectuar el mismo proceso. Una ventaja en los saltos de una línea a la otra, es que el desplazamiento siempre va ser constante y corresponde exactamente a la cantidad de pixeles por renglón.

Por su parte, siempre es necesario que el algoritmo en cada operación de convolución cheque si no ha llegado al borde derecho de la imagen, es decir a tener un valor $a_{ij}=(i,197)$, ya que si esto sucede, sería el último pixel en procesar de este renglón, el algoritmo debe saltarse un renglón e iniciar el proceso de nuevo en la primera columna.

El algoritmo también debe de checar que el proceso no rebase más allá del renglón 199, para no estar fuera de los límites de la imagen, y en el caso que la imagen ocupe toda la memoria (64K) no caer en el error de que el contador de localidades se vaya al sobreflujo, por lo que también cada vez que finalice el procesamiento de un renglón debe checar si el primer valor de la imagen en esa convolución está en $a_{ij}=(198,0)$. De ser así, entonces el procesamiento de la imagen finaliza. Dentro de las consideraciones importantes tomadas en cuenta para la implementación del algoritmo, se tiene la utilización de los registros auxiliares, los modos de direccionamiento y las 544 palabras en memoria RAM que posee el TMS. Estas palabras,

están divididas en tres bloques separados, 288 siempre se utilizan como memoria dato y corresponden a los bloques B1 y B2, mientras que las restantes 256 palabras corresponden al B0, y se pueden configurar como memoria dato o programa. El bloque B1 con 256 palabras, se ubica en las páginas 6 y 7 de la memoria dato en el mapa de memoria y el bloque B2 con 32 palabras se ubica en la parte alta de la página cero de memoria dato.

Cuando el bloque B0 se utiliza como memoria dato, este se ubica en las páginas 4 y 5 del mapa en memoria dato, si el bloque B0 se utiliza como memoria programa, éste se ubica de la localidad >FF00 a >FFFF en el mapa de memoria programa. Mediante la instrucción CNFD, se configura a B0 como memoria dato y con CNFP como memoria programa. Estas dos instrucciones permiten la configuración dinámica del mapa de memoria a través de software. Después de la ejecución de la instrucción CNFP, el bloque B0 es mapeado dentro de la memoria programa, lo que permite utilizar la potencialidad de la instrucción MAC (multiplicación acumulación), la cual multiplica un valor en memoria dato por un valor en memoria programa, sumando el producto previo al acumulador con un corrimiento definido por el bit de status PM.

Con la utilización simultánea de la instrucción RPTK (repita la instrucción siguiente como lo especifica la constante inmediata más una vez), la instrucción MAC se puede hacer más eficiente todavía, al efectuar la operación en un ciclo de instrucción una vez que la repetición de pipeline ha sido instalada, lo que implica la reducción del número de instrucciones/pixel a utilizar. Con operaciones pipeline actuando para 3 pixeles consecutivos, se logra la reducción del proceso de multiplicación acumulación a una instrucción/pixel.

Esta ventaja es utilizada en el algoritmo, que junto con el orden en que han sido escrito los datos, ha servido para implementar la tres sumas de productos de la ecuación 2, donde se utiliza la repetición de tres operaciones similares, y se aprovecha además la instrucción MAC para el direccionamiento simultáneo de los dos datos a operar. La forma de realizarlo es la siguiente: la instrucción ZAC, pone al ACC en cero para dar inicio al procesamiento de un pixel. La instrucción MPYK >0 , multiplica el registro T por cero y el resultado lo escribe en el registro P, esto se hace debido a que la instrucción MAC, siempre inicia sumando: ACC + reg. P ---> ACC. En seguida la operación MAC efectúa la operación (dma)*(pma) ---> reg.P. Con la instrucción RPTK >3, se logra que MAC rescate el valor de la tercera multiplicación y la acumule en ACC desechando la cuarta multiplicación que se aprecia en la figura 5.4. Por lo tanto para iniciar otro ciclo de operaciones con MAC, es necesario tener inicialmente al reg. P en cero.

SALTOS DEL ALGORITMO DE PROCESAMIENTO

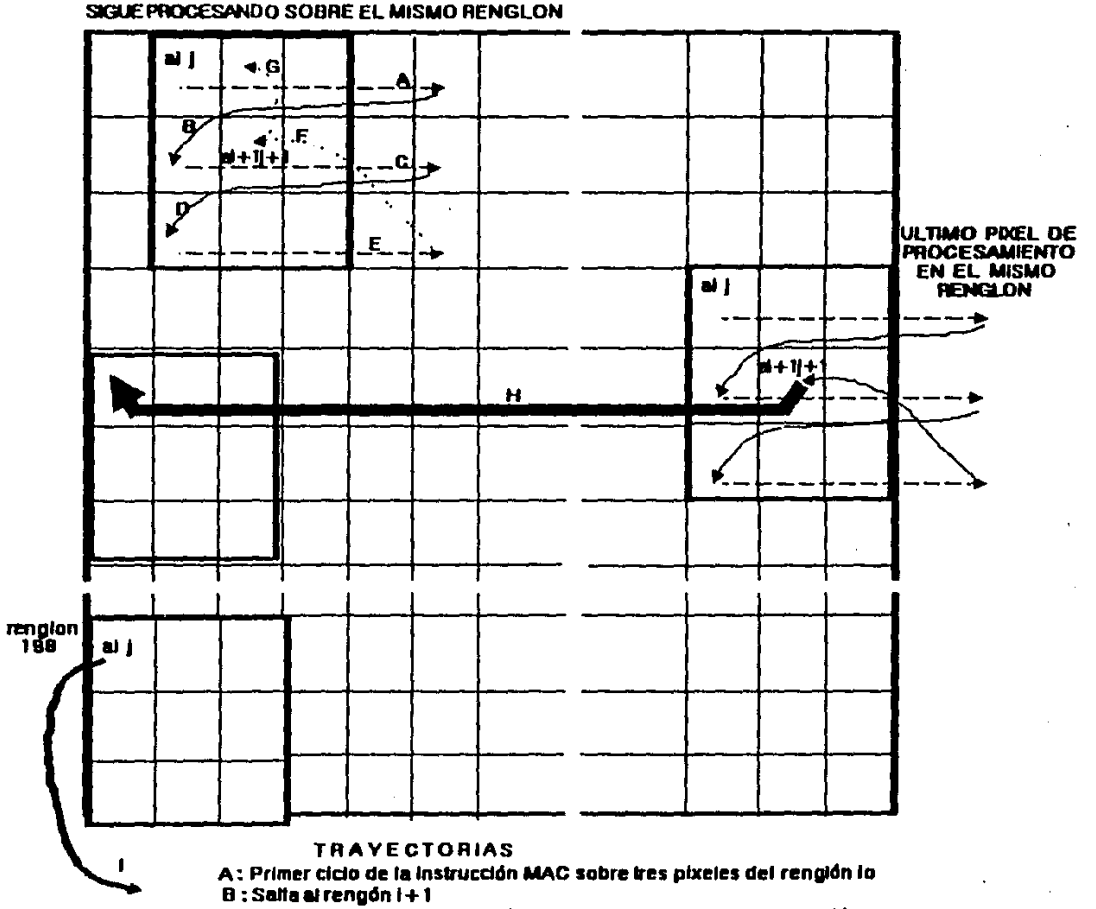


Figura 5.4

VI RESULTADOS Y CONCLUSIONES

INTRODUCCION

En el desarrollo del presente trabajo ha sido incluido un estudio detallado sobre los diversos fundamentos necesarios para llevar a cabo la implementación de una Arquitectura para el Procesamiento Digital de Imágenes. Dicho estudio ha abarcado desde el conocimiento de aspectos de naturaleza física relacionados con la visualización de un objeto, dispositivos electrónicos para la adquisición de las imágenes digitales, procesamiento de imágenes (hardware y software), hasta el manejo de algoritmos matemáticos. Todos estos elementos interrelacionados por medio de una aplicación específica dentro del área de procesamiento de imágenes, han permitido el desarrollo e implementación de la arquitectura propuesta.

La importancia de la utilización de un procesador para señales digitales, tal como el TMS320C25, se justifica primordialmente por sus grandes posibilidades en la ejecución de operaciones matemáticas intensivas, la utilización de la vectorización de operaciones (pipeline) y su posibilidad de trabajar en arquitecturas de múltiples procesadores.

Este capítulo presenta los resultados y conclusiones obtenidos de la evaluación de la arquitectura propuesta en el presente trabajo.

VI.1 RESULTADOS SOBRE EL PROCESAMIENTO DE IMAGENES

En la obtención de las imágenes resultado que se presentan a continuación, se emplearon imágenes capturadas a través de una tarjeta digitalizadora desarrollada en el IIMAS UNAM [IIMAS, 1990], mismas que representan las imagenes fuentes dentro de arquitectura APDI (figuras 6.1 y 6.7). Estas imágenes fuentes han sido utilizadas para demostrar el funcionamiento de la arquitectura APDI y los diversos procesos implementados. Entre estos procesos se cuenta con: Filtro Paso Altas, Filtro Paso Altas, Filtro Diferenciador Vertical, Operador Detector de Bordes de Sobel y Operador Laplaciano. Las imágenes resultado así logradas son mostradas en las páginas 128 y 129, comentándose posteriormente sobre cada una de ellas así como del tipo de proceso asociado.

Resultados

Figura 6.1 : Original

Figura 6.2 : Filtro Paso Altas

Figura 6.3 : Filtro Paso Bajas

Figura 6.4 : Filtro Diferenciador Vertical

Figura 6.5 : Operador de Sobel

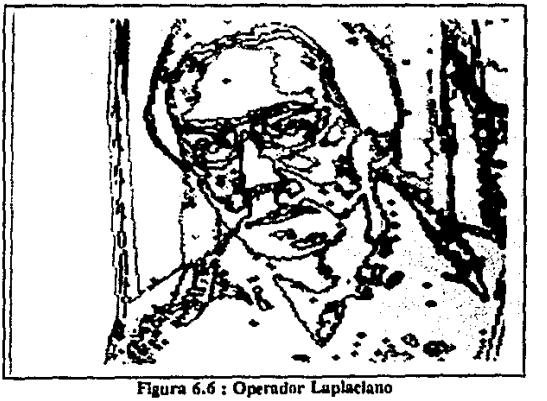


Figura 6.6 : Operador Laplaciano

Resultados

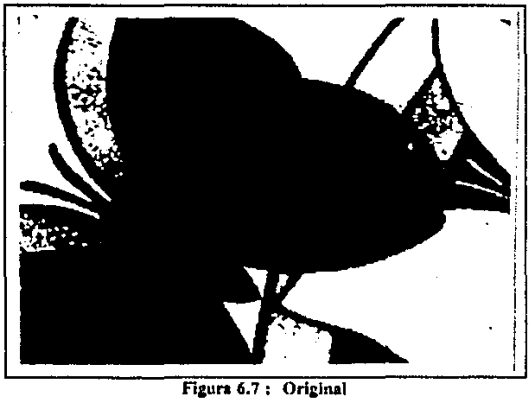


Figura 6.7 : Original

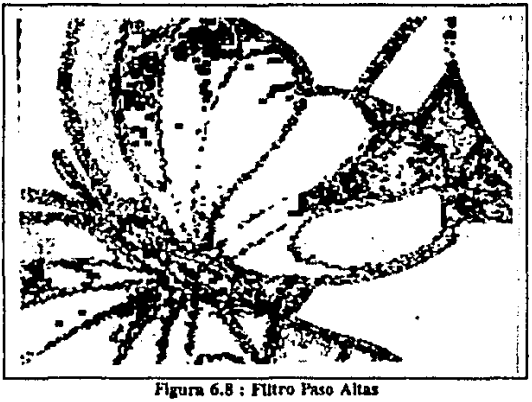


Figura 6.8 : Filtro Paso Altas

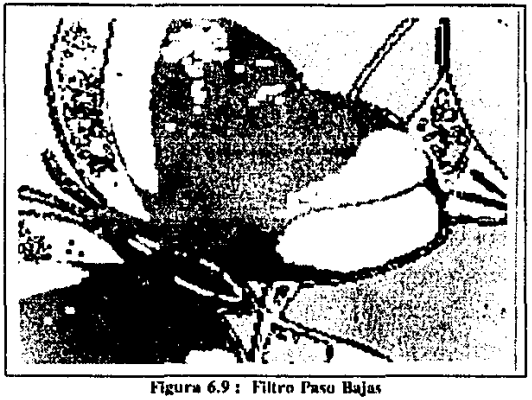


Figura 6.9 : Filtro Paso Bajas

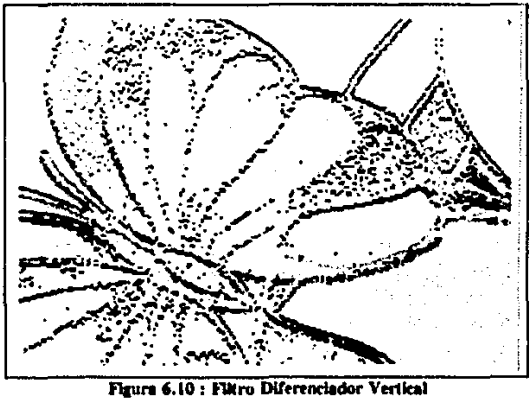


Figura 6.10 : Filtro Diferenciador Vertical

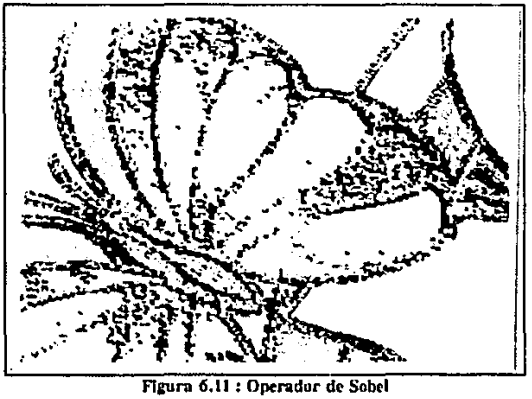


Figura 6.11 : Operador de Sobel

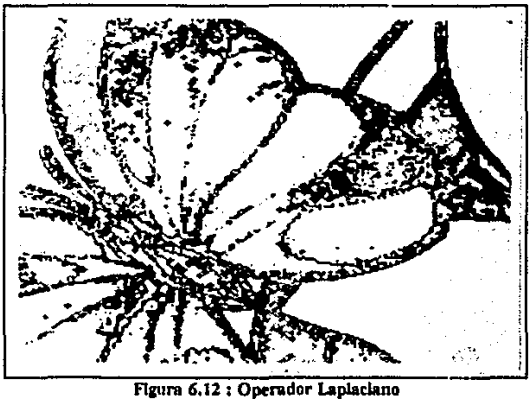


Figura 6.12 : Operador Laplaciano

Filtro Paso Altas

Con el empleo de este filtro se obtienen en las imágenes resultado (Figs 6.2 y 6.8) una atenuación en las componentes de baja frecuencia de las imágenes originales y un realce en las componentes que determinan los bordes de las imágenes. Se puede destacar que este tipo de proceso proporciona esencialmente la información referente a los bordes de una imagen.

Filtro Paso Bajas

Los resultados obtenidos al aplicar esta filtro en las imágenes originales son: atenuación de las componentes de alta frecuencia de la imagen y un predominio de las áreas cuyas componentes son de baja frecuencia, en consecuencia existe una pérdida de información en cuanto a los detalles de las imágenes (Figs. 6.3 y 6.9).

Filtro Diferenciador Vertical

En este tipo de procesamiento se obtienen imágenes (Figs. 6.4 y 6.10), en donde sus bordes horizontales son realzados y se atenúan los verticales. La pérdida de información es notable puesto que prácticamente se conserva la información de los bordes horizontales de la imagen.

Operador Detector de Bordes Sobel

Al aplicar este operador en las imágenes originales se obtiene un realce de sus bordes. Esto indica que el operador detecta los cambios bruscos de una imagen tanto en las direcciones vertical como horizontal, atenuando las componentes de baja frecuencia de las imágenes (Figs. 6.5 y 6.11). Una limitante observada es que los bordes obtenidos no presentan un tono de gris homogéneo en todo el contorno de un objeto, el cual depende de la información en la imagen; de esta forma, ciertas secciones del contorno no son apreciables.

Operador Laplaciano

Este operador al ser aplicado en las imágenes originales proporciona los bordes de las mismas de una forma más detallada en comparación a los obtenidos con el operador de Sobel. En los resultados se aprecia el realce de los bordes de las imágenes (Figs. 6.6 y 6.12), los cuales son el resultado de la sensibilidad que tiene el operador a los cambios bruscos en las direcciones tanto vertical como horizontal. Sin embargo, existe la pérdida de información debido a que el operador no puede detectar los cambios posibles en forma diagonal de acuerdo a la plantilla del operador laplaciano.

VI.2 MEDICIONES SOBRE TIEMPOS DE PROCESAMIENTO PARA DIVERSOS ALGORITMOS

El programa de procesamiento desarrollado como elemento de prueba y evaluación del sistema conllevó a optimizar los recursos del TMS, haciendo uso extensivo del pipeline, a través de la utilización de la instrucción MAC (multiplicación acumulación), en la implementación de la convolución de la imagen con una plantilla.

Para la medición de los tiempos de procesamiento se recurrió a la observación del envío de las palabras de control de inicio y fin de procesamiento. Para la salida correspondiente al bit de inicio se midió con la ayuda de un osciloscopio, el tiempo de ejecución del algoritmo, a través del programa de procesamiento modificado para enviar en forma repetitiva las señales de inicio y fin de procesamiento. Después de varias pruebas de verificación se observó que para el algoritmo de convolución de una matriz imagen de 200 x 200 pixels con una plantilla de 3x3, se generaban ciclos con un tiempo promedio de procesamiento de:

$$\mathbf{Tpi1 = 0.464\ s}$$

Esto significa que la APDI es capaz de procesar un promedio de dos imágenes por segundo, lo cual se puede considerar un resultado satisfactorio. En el caso de emplear dos unidades de procesamiento el tiempo de procesamiento se reduce a:

$$\mathbf{Tpi2 = 0.232\ s}$$

y la cantidad de imágenes a procesar en un segundo se incrementa a cuatro. Estos tiempos de procesamiento Tpi1 y Tpi2 son el producto de trabajar con una o dos unidades de procesamiento respectivamente, sin embargo, en el caso de procesar la señal de video en tiempo real (adquisición y despliegue de 30 imágenes/s), implicaría la utilización del procesamiento en paralelo con al menos 16 unidades de este tipo, las cuales utilizarían el mismo programa de procesamiento sobre bloques de información de menores dimensiones. Para una imagen de 256X256 pixels (con dos unidades) se tendría un tiempo:

$$\mathbf{Tpi3 = 0.371\ s}$$

Respecto a este número de unidades trabajando en paralelo, es importante señalar que estas cantidades (tiempo y unidades) se pueden reducir al eliminar el estado de espera que ha sido incluido de acuerdo al tipo de componentes utilizados, es decir que Tpi1 se reduciría a la mitad. Por su parte si se consideran dos unidades de procesamiento y una imagen de 200X200, se tendría un tiempo de procesamiento igual a:

$$\mathbf{Tpi4 = 0.116\ s}$$

La eliminación de este estado de espera depende de la utilización de componentes electrónicas más rápidas (compuertas y memorias). Bajo esta consideración podría pensarse en alcanzar una optimización a 8 unidades trabajando en paralelo.

V1.3 COMPARACION ENTRE LA IMPLEMENTACION DEL ALGORITMO BASADO EN EL LENGUAJE ENSAMBLADOR DEL TMS320C25 Y EL LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR INTEL 8086

Nuestro programa de procesamiento, tal como se explicó en el capítulo anterior, consta de un bloque de inicialización, otro de reconocimiento del número de proceso a realizar, un tercero de configuración de la tarjeta y finalmente el bloque principal de procesamiento. Estos tres primeros bloques del programa son ejecutadas sólo una vez cuando se procesa una imagen, no así el bloque principal de procesamiento, el cual se efectúa una vez por cada pixel de la imagen. Dentro del bloque principal existen tres loops por médio de los cuales se efectúa la convolución y constituyen el loop principal de procesamiento. Este loop principal consume la mayor parte del tiempo de procesamiento en una imagen, ya que si se contabilizan el número de ciclos que consume la parte del programa fuera del loop principal, se tienen 295 ciclos de máquina que en tiempo corresponde al 0.0127% de Tpi1 y el loop principal consume el 99.987% de Tpi1.

Tomando en cuenta este loop principal, se hace una comparación en términos de los ciclos de máquina utilizados por la APDI y un microprocesador de uso general como es el 8086 de INTEL, utilizando el microprocesador 8086-1 que trabaja con un reloj de 10 Mhz [INTEL 1190], con un ciclo de máquina de 100 ns que es el mismo del TMS, lo que da lugar a una comparación más precisa de procesador a procesador.

V1.3.1 Bloque del programa de procesamiento que efectúa la convolución Ciclo de máquina utilizado por el TMS

```
                        ciclos
SALT11  ZAC             2
        LARP AR0        2
        LAR AR0,CONT    3
*
        MPYK >0         2
        RPTK 3          2   ----------------┐ Loop 1
        MAC >FF00,*+    14  ----------------┘
        ADRK >C4        2
*
        MPYK >0         2
        RPTK 3          2   ----------------┐ Loop 2
        MAC >FF03,*+    14  ----------------┘
        ADRK >C4        2
*
        MPYK >0         2
        RPTK 3          2   ----------------┐ Loop 3
        MAC >FF06,*+    14  ----------------┘
```

La comparación de la implementación del algoritmo se puede reducir a la utilización de un sólo loop de los tres marcados, ya que se repiten las mismas operaciones sólo que con diferentes datos cuando se procesa un mismo pixel. Bajo esta consideración se implementa en el ensamblador del 8086 únicamente la rutina correspondiente a la parte marcada como Loop 1 para convolución.

V1.3.2 Implementación del Loop para convolución en ensamblador del microprocesador 8086

Para hacer la comparación de los ciclos de máquina utilizados por cada procesador en sus respectivas implementaciones del algoritmo, se ha tomado en cuenta la optimización de las instrucciones para que realicen las operaciones pertinentes. Debido a esta consideración, en el caso del 8086 se utilizan datos de 8 bits, los cuales son suficientes para la manipulación de los datos de la imagen.

En la implementación del Loop 1 para convolución en lenguaje ensamblador, se utilizan instrucciones de carga de memoria a registros, entre registros y de registros a memoria con la finalidad de direccionar los valores empleados en la multiplicación, así como acumulación del resultado en una localidad de memoria. Las características de operación del microprocesador 8086 no permiten direccionar en una misma instrucción dos localidades de memoria a la vez, en comparación al TMS que sí lo permite por medio de la instrucción MAC, lo que ocasiona que el programa para el 8086 requiera más instrucciones.

Para efectuar la multiplicación y acumulación del resultado entre tres pares de datos (3 pixels seguidos de la imagen y 3 datos consecutivos de la plantilla), se utiliza un bucle que se repite tres veces. En el TMS simplemente se recurre a la repetición de la instrucción MAC que efectúa la multiplicación y acumulación del resultado.

La siguiente rutina realiza la multiplicación de tres pares de datos previamente almacenados en memoria, los resultados de cada multiplicación son sumados y almacenados en una localidad de memoria.

V1.3.3 Ciclos de máquina utilizado por el microprocesador 8086

En esta rutina, a los registros empleados se les asigna una función predeterminada: el registro DS, que sirve para almacenar los desplazamientos de los datos (pixeles), puede almacenar dos desplazamientos: uno igual a 9000H para indicar el inicio de la plantilla y resultados, y otro igual a 8000H para el de datos. El registro BP es utilizado para almacenar el desplazamiento correspondiente a la plantilla y resultados. Los registros SI y DI almacenan los desplazamientos empleados para en el direccionamiento (registro indirecto) de los datos y constantes a multiplicar. El registro AX es utilizado para almacenar constantes y transferirlas a los registros BP, DI, SI y DS, además en la multiplicación/acumulación de los datos. El

registro BL interviene para almacenar el multiplicador. Finalmente, el registro CX almacena el valor de repetición del bucle para obtener las tres multiplicaciones y la suma de estos productos necesarias en la implementación del Loop para convolución.

```
                          ciclos        comentario
    mov ax,9000H          ;             Carga de Dir Inicial de Plant en Mem en Reg AX
    mov bp,ax             ;             Almacenamiento de Dir Inicial de Plant en Reg BP
    mov ax,0000H          ;             Carga de Desplazamiento Inicial en Reg AX
    mov si,ax             ;             Almacenamiento de Desplazamiento Inicial de Datos en Reg SI
    mov di,ax             ;             Almacenamiento de Desplazamiento Inicial de Plant en Reg DI
    mov cx,0003H          ; 4           Carga del Número de Repeticiones del bucle en Reg CX
@et1: mov ax,8000H        ; 4           Carga de Dir Inicial de Datos en Mem en Reg AX
    mov ds,ax             ; 2           Carga de Dir Inicial de Datos en Reg DS
    mov al,[si]           ; 13          Carga de Dato de Mem en AL <= = Dir (DS+SI)
    mov ds,bp             ; 2           Carga de Dir Inicial de Plant en Mem en Reg DS
    mov si,di             ; 2           Carga de Desplazamiento de Plant en Reg SI
    mov bl,[si]           ; 13          Carga de Cte en Mem en BL <= = Dir (DS+SI)
    imul bl               ; 98          Mult Signada AX <= = ALxBL
    add al,[10H]          ; 15          Adición del Resultado de la Mult con un Dato en Mem
                          ;             AL <= = AL + Dir (DS + 10H)
    mov [10H],al          ; 15          Carga de la Mul/Adición en Mem Dir (DS+10H)
    inc si                ; 2           Incrementa Reg SI <= = SI + 1
    inc di                ; 2           Incrementa Reg DI <= = DI + 1
    loop @et1             ;7/12         Dec CX <= = CX - 1, Si CX >0 CP <= = Dir ;

        Total : 181x3 -12 + 4 = 535 ciclos
```

Del listado anterior el total de ciclos utilizados en la implementación del loop en ensamblador, es de 535 ciclos de máquina. Como se observa, esta rutina realiza las funciones que en el TMS ejecuta únicamente con dos instrucciones (para direccionar los datos y efectuar las tres multiplicaciones y almacenar la suma de los respectivos resultados), con un total de 16 ciclos de máquina.

De este modo, el tiempo de proceso del TMS resulta ser 33 veces más rápido con respecto al microprocesador 8086-1. Por otra parte, es suficiente hacer una comparación contra el número de ciclos utilizados por la instrucción IMUL (98 ciclos) para tener una idea aproximada de la eficiencia del TMS sobre el 8086-1 en este tipo de operaciones.

VI.4 COMPARACION CON OTRAS TARJETAS EN EL MERCADO

La marca **Data Translation Inc. [Data Translation 1992]** ha introducido al mercado una familia de tarjetas para el procesamiento digital de imágenes, las cuales involucran la realización de operaciones de convolución, sustracción, adición, la creación de histogramas, promediación de imágenes y operaciones lógicas. Esta tarjeta efectúa estas operaciones de 10 a 100 veces más rápido que si lo efectuara una PC tipo AT. Según reporta la mencionada compañía, la tarjeta DT2868 ejecuta 10 millones de adiciones o multiplicaciones por segundo y 3.9 millones de divisiones por segundo. Esto significa que puede efectuar la convolución de una matriz de 3 X 3 sobre una imagen de 512 X 512 en 0.56s, o efectuar promedios con una precisión de 16 bits a razón de 0.46s.

Teniendo en consideración que la arquitectura APDI ha sido evaluada con imágenes de 200 X 200 (40 Kb procesadas en 0.464 s para el algoritmo de convolución), que en cantidad de información es casi la séptima parte de la que procesa la tarjeta comercial, tenemos que en términos de tiempos la tarjeta DT2868 tardaría en procesar la misma imagen con la misma plantilla, en un tiempo de 0.0854s, es decir, el 18.4% del Tpi1, o bien el 73.6% del Tpi4.

VI.5 CONCLUSIONES

A continuación se resumen las conclusiones más importantes obtenidas en base a los resultados del desarrollo, prueba e implementación de la Arquitectura de Procesamiento Digital de Imágenes:

- Se ha demostrado la realización de funciones de procesamiento digital básico sobre imágenes, a partir de las cuales se pueden realizar otras tareas de procesamiento con un tiempo de proceso sensiblemente menor con respecto a una computadora de tipo personal.

- Se ha realizado el manejo de altos volúmenes información aplicando algoritmos matemáticamente intensivos en tiempos relativamente bajos.

- Se ha expuesto la importancia del procesamiento en paralelo y el diseño modular, que junto con un software apropiado puede dar soluciones adecuadas a problemas de procesamiento de señales digitales.

- Se ha demostrado una implementación optimizada por hardware de la convolución en dos dimensiones, que ha sido de la arquitectura APDI.

- La arquitectura APDI aunada al ensamblador y simulador del TMS, puede servir como un Módulo de Evaluación para el TMS320C25.

VII BIBLIOGRAFIA

[CASTLEMAN, 1979]

CASTLEMAN, KENNETH R. Digital Image Processing, New Jersey, Prentice-Hall Inc., 1979. 429 p.

[CONTE y DEL CORSO, 1985]

CONTE, Gianni y DEL CORSO, Dante. Multi-Microprocessor Systems for Real/Time Applications. Boston, D. Reidel Publishing Company, 1985. 299 p.

[CYPRESS, 1989]

CYPRESS Semiconductor. CMOS BiCMOS Data Book. San José California, 1989.

[DATA TRANSLATION, 1992]

DATA TRANSLATION Inc. 1992 Product Handbook. Vol. 2 No. 1. Marlboro, Ma U.S.A., January 1992.

[DAVIS y O'KEEFE, 1989]

DAVIS, Ruth M. y O'KEEFE, Robert M. Simulation Modelling With Pascal. Prentice Hall, Great Britain, 1989. 302 p.

[DE GIUSTI, 1991]

DE GIUSTI, Armando E. Descripción y Validación de Hardware, Aplicaciones en Tiempo Real. Rfo de Janeiro, Brasil, R. Vieria Grafica E. Editora LTDA (EBA I), 1991.

[ENSLOW, 1974]

ENSLOW, Philip H. Ed. Multiprocessors and Parallel Processing. New York, John Wiley & Sons, 1974. 340 p.

[GALBIATI, 1990]

GALBIATI, Louis J. Jr. Machine Vision and Digital Image Processing Fundamentals. New Jersey, Prentice-Hall Inc., 1990, 164p.

[GODFRED 1991]

GODFRED, J. Terry. Lenguaje ensamblador para microcomputadoras IBM, para principiantes y avanzados. Prentice Hall, México 1991. 517p.

[GROB 1975]

GROB, Bernard. Basic Televition, principles and servicing. MacGra-Hill, U.S.A., 1975. 732p.

[HALL, 1986]

HALL, Douglas V. Microprocessors and Interfacing Programming and Hardware. Singapore, Mc Graw-Hill, 1986. 554 p.

[HWANG y DEGROOT, 1989]

HWANG, Kai y DEGROOT, Douglas. Parallel Processing for Supercomputers and Artificial Intelligence. New York, Mc Graw-Hill Publishing Company, 1989. 673 p.

[IIMAS, 1990]

IIMAS-UNAM. Tarjeta Digitalizadora de Imágenes de Video, Manual Técnico. Departamento de Electrónica y Automatización. Instituto de Investigaciones y en Matemáticas Aplicadas y en Sistemas (IIMAS), UNAM, México, 1990.

[INTEL, 1990]

INTEL. Microprocessors. U.S.A., 1990.

[JAIN, 1989]

JAIN, Anil K. Fundamentals of Digital Image Processing, New Jersey, Prentice-Hall, 1989. 569 p.

[KRISHNAMURTHY, 1989]

KRISHNAMURTHY, E. V. Parallel Processing Principles and Practice. Singapore, Addison-Wesley Publishing Co., 1989. 332 p.

[O'BRIEN, 1989]

O'BRIEN, Stephen K. Turbo Pascal 5.5 The Complete Reference. Berkely California, Mc Graw-Hill,1989. 917 p.

[OFFEN, 1985]

OFFEN, R.J. VLSI Image Processing. London, William Collins Sons & Co., 1985. 326 p.

[PUTMAN 1988]

PUTMAN, Byron W. Microcomputer Hardware, operation and troubleshooting with IBM PC aplications. Prentice Hall. U.S.A. 1988. 267 p.

[SCHALKOFF, 1989]

SCHALKOFF, Robert J. Digital Image Processing and Computer Vision. Singapore, John Willey & Sons, Inc., 1989. 489 p.

[SINGH & TRIEBEL, 1989]

SINGH Avtar and TRIEBEL, Walter A. The 8088 Microprocessor, Programing, Interfacing, Software and Aplications. New Jersy, Prentice Hall, 1989. 506 p.

[STARGET y SHOEMAKER, 1986]

STARGET III, Murray y SHOEMAKER, Richard L. The IBM Personal Computer from Inside Out. Addison Wesley, Tucson Arizona, 1986.

[TEXAS INSTRUMENTS, 1988]

TEXAS INSTRUMENTS. TTL Logic Standard TTL, Schottky, Low-Power Schotkly. U.S.A., 1988.

[TEXAS INSTRUMENTS, 1988]

Second-Generation TMS320 User's Guide. Texas Instruments, U.S.A. Prentice Hall, T.I. 1988.

[TEXAS INSTRUMENTS, 1986]
TEXAS INSTRUMENTS. ALS/AS Logic Data Book. U.S.A. 1986.

[TEXAS INSTRUMENTS, 1988]
TEXAS INSTRUMENTS. TMS 320C25 User's Guide, U.S.A., 1986.

[TEXAS INSTRUMENTS, 1989]
TEXAS INSTRUMENTS. Digital Signal Proccesing Applications with the TMS320 Family, Theory, Algorithms and Aplications. Vol. 1 y 2. U.S.A., 1989.

[UAM, 1992]
UNIVERSIDAD AUTONOMA METROPOLITANA, IZTAPALAPA. Memorias. II Taller Internacional de Procesamiento Digital de Imágenes y Visión. México, UAM, marzo-abril 1992.

[WEGMAN & DEPRIEST 1986]
WEGMAN, E. J. and DEPRIEST, D. J. Statistical Image Processing and Graphics. Marcel Dekker, Inc. N.Y. 1986.

ANEXO A
EL MICROPROCESADOR TMS320C25

EL TMS320C25
DESCRIPCION Y ARQUITECTURA [Texas Instruments,1988]

La familia de procesadores TMS320 de 16/32 bits de Texas Instruments, empleada para el Procesamiento Digital de Señales (DSP), combina la flexibilidad de un controlador de alta velocidad con la capacidad numérica de un arreglo procesador, que ofrecen una alternativa inmensa para la tecnología VLSI y el multiprocesamiento de señales.

Estas características hacen que esta familia ofrezca alta capacidad de ejecución, una arquitectura de gran funcionalidad y una relación costo/efectividad ideal para solucionar diversos problemas en telecomunicaciones, computación, comercio, industria, en aplicaciones militares, entre otras.

DESCRIPCION GENERAL

La combinación de la arquitectura tipo Harvard de la familia TMS320 (separación de los buses de datos y datos de programa) y su conjunto de instrucciones especiales para el DSP proveen una alta velocidad de ejecución y gran flexibilidad. De esta forma se produce una familia de procesadores capaces de ejecutar 10 MIPS (Millones de instrucciones por segundo), ya que se implementan funciones en hardware, que en comparación a otros procesadores, recurren al software o microcódigos para generarlas.

La segunda generación de la familia TMS320 esta compuesta de dos procesadores el TMS32020 y el TMS320C25, la arquitectura de ambos procesadores se basa en la del TMS23010. El TMS32020 trabaja a 20 MHz y ejecuta el doble de instrucciones del TMS32010, mientras el TMS320C25 incrementa la funcionalidad de la arquitectura en relación al TMS32020.

CARACTERISTICAS DEL TMS320C25

- Ciclo de Instrucción 100 ns.
- 544-Palabras de datos programables en memoria RAM interna.
- 4K-Palabras de programa en memoria ROM interna.
- 128K-Palabras de espacio total de memoria DATOS y PROGRAMA.
- ALU/Acumulador de 32 bits.
- Multiplicador paralelo de 16x16 bits con producto de 32 bits.
- Instrucciones multiplicación/acumulación ejecutadas en un ciclo simple de instrucción.
- Instrucciones de repetición para uso eficiente del espacio de programa y mejor ejecución.
- Movimiento de bloques para el manejo de DATOS/PROGRAMA.
- Timer integrado para operaciones de control.
- Ocho registros auxiliares con una propia unidad aritmética.

- Un stack por hardware de ocho niveles.
- Dieciséis canales de entrada o salida.
- Un registro de corrimiento paralelo de 16 bits.
- Posibilidad de generar tiempos de espera para comunicación a periféricos o memorias de respuesta lenta.
- Un puerto serie para interface directa a codecs.
- Entrada de sincronización para sincronía en configuraciones de múltiples procesado res.
- Interface de memoria de datos global.
- Compatibilidad de códigos fuente del TMS32010.
- Concurrencia con DMA usando una operación de espera extendida.
- Instrucciones para implementar filtros, la Transformada Rápida de Fourier, y aritmética de precisión extendida.
- Generador de reloj interno.
- Suministro de potencia 5 Volts.

ARQUITECTURA

Descripción

La destacada funcionalidad del TMS320C25 (TMS) para el Procesamiento Digital de Señales (DSP), se debe a su tipo de arquitectura (tipo Harvard) que maximiza el procesamiento mediante el establecimiento de dos estructuras de buses de memoria separadas (memoria programa y memoria datos), para incrementar la velocidad de ejecución, contando con instrucciones para realizar transferencia de datos entre ambos espacios.

Externamente, las memorias de programa y datos son multiplexadas sobre el mismo bus externo para maximizar el rango de direccionamiento para dichos espacios mientras se minimizan los pines del dispositivo.

La flexibilidad en el diseño del sistema es incrementada al contar con dos bloques de datos internos en RAM (un total de 544-Palabras), donde uno de ellos puede ser configurado como memoria programa o memoria dato. El TMS es capaz de direccionar externamente 64K-Palabras en un espacio de memoria dato, para facilitar la implementación de algoritmos para DSP.

La memoria interna ROM mascarable de 4K-Palabras puede ser usada para reducir el costo de sistemas. Los programas de 4K-Palabras pueden ser mascarables en la memoria ROM interna y de esta forma pueden ser ejecutados a alta velocidad desde este espacio de memoria. Externamente el espacio de memoria de programa direccionable es de 64K-Palabras.

El TMS funciona con una aritmética en modo dos complemento, empleando una ALU y un Acumulador de 32 bits. La ALU es una unidad aritmética de propósito general que opera

con palabras de 16 bits provenientes de la RAM de datos o derivadas de instrucciones inmediatas o por el empleo del registro resultado (producto) del multiplicador de 32 bits. La ALU puede efectuar operaciones booleanas. El Acumulador almacena los resultados de la ALU y a su vez es una segunda entrada a la ALU. La longitud total del Acumulador es de 32 bits, la cual es dividida en dos partes una alta (bits 31 al 16) y otra baja (bits 15 al 0), y para el manejo de datos se tiene instrucciones de almacenamiento para cada una de las partes (alta y baja) del Acumulador.

El multiplicador realiza operaciones de 16x16 bits en modo dos complemento con un resultado de 32 bits en un solo ciclo de instrucción. El multiplicador está compuesto de tres elementos: el registro T (de 16 bits) para almacenar temporalmente el multiplicado, el registro P (de 32 bits) para almacenar el producto, y un arreglo multiplicador. Los valores del multiplicador provienen de la memoria dato, memoria programa (cuando se emplean las instrucciones MAC/MACD), o son derivados de una instrucción inmediata MPYK (multiplicación inmediata). La rapidez del multiplicador integrado permite la ejecución de operaciones fundamentales en el DSP como la convolución, correlación y filtrado.

El registro de corrimiento del TMS tiene una entrada de 16 bits y está conectado al bus de datos, mientras que su salida de 32 bits está conectada a la ALU. Este registro proporciona corrimientos a la izquierda de 0 a 16 bits sobre el datos de entrada y son programados mediante instrucción. Adicionalmente, se tiene la capacidad de que el procesador funcione con escalamiento numérico, extracción de bits, aritmética extendida y prevención de sobreflujo.

La interfase local de la memoria del TMS consiste de un bus de datos paralelo de 16 bits (D15-D0), un bus de direcciones de 16 bits (A15-A0), tres pines para selección de memoria dato/programa o espacio de entrada/salida (/DS,/PS e /IS), y varias señales de control del sistema. La señal R/w controla el sentido de transferencia del dato, y la señal /STRB provee un tiempo válido para la transferencia de información. Cuando de emplea las memorias ROM, RAM internas o memoria de programa externa de alta velocidad, el TMS ejecuta instrucciones a la máxima velocidad sin tiempos de espera. De otro modo, el empleo de la señal READY es para generar tiempos de espera para comunicarse con memorias externas lentas.

El stack por hardware de ocho niveles es empleado para almacenar el contenido de contador de programa durante interrupciones y llamadas de subrutinas. Las instrucciones PUSH y POP permiten salvar y recuperar información contenida en el stack. Las interrupciones empleadas en el dispositivo son mascarables.

Las operaciones de control son proporcionadas en el TMS por un timer interno de 16 bits mapeado en memoria, un contador de repetición, tres interrupciones externas mascarables, y una interrupción interna generada por el puerto serie o el timer.

Un puerto serie interno full-duplex provee comunicación directa con dispositivos seriales como codecs, convertidores seriales A/D, y otros dispositivos seriales. Los dos registros del puerto serial mapeados en la memoria (registros de transmisión/recepción de dato) pueden operar en modo byte (ocho bits) o modo word (16 bits). Cada registro tiene una entrada

de reloj externa, una entrada de sincronía y registros de corrimiento. La comunicación serial puede ser usada entre procesadores en aplicaciones de múltiples procesadores.

El TMS para aplicaciones de múltiples procesadores tiene la capacidad de distribuir el espacio global de memoria de dato y de comunicación con este espacio mediante las señales /BR (requerimiento de bus) y READY. El registro de distribución de la memoria global de dato (GREG) de ocho bits especffica hasta 32K-Palabras de memoria dato como memoria global externa. El contenido del registro determina el tamaño del espacio global de memoria. Si la instrucción corriente direcciona un operando dentro de este espacio, /BR es insertado para requerir el control del bus. La extensión del ciclo de lectura/escritura de memoria es controlado con la línea READY.

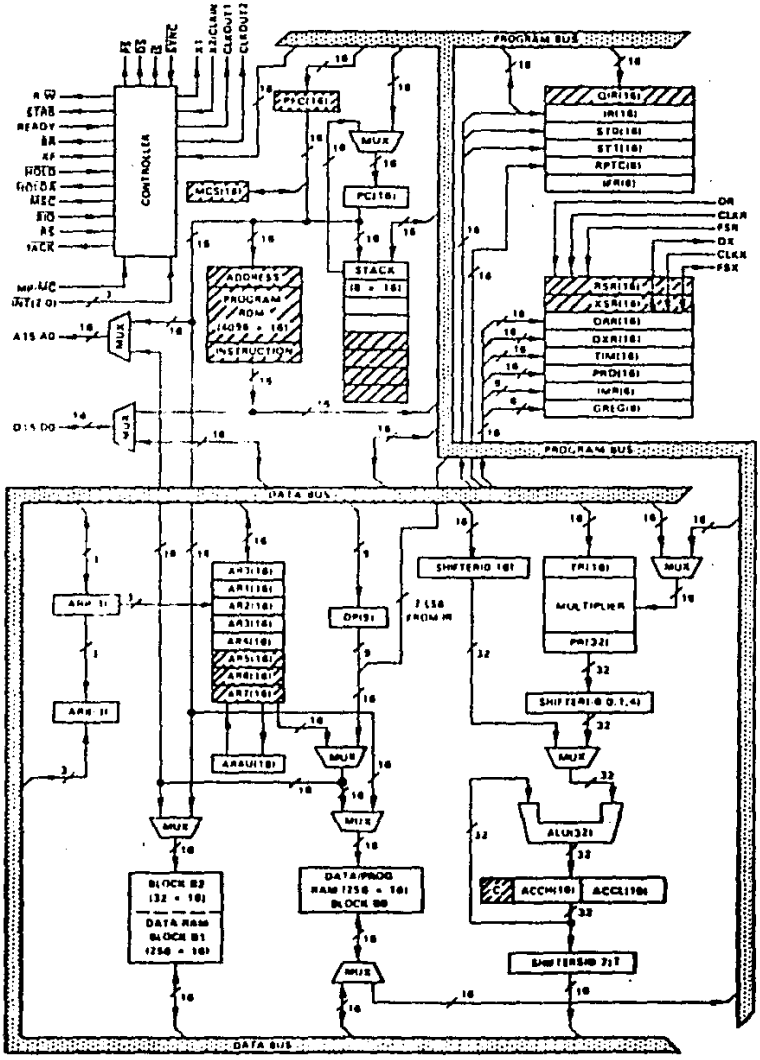
El procesador TMS soporta Acceso Directo a Memoria (DMA) para su memoria externa dato/programa empleando las señales /HOLD y /HOLDA. Otro procesador puede tomar por completo el control de la memoria externa del TMS por medio de la señal /HOLD (activa baja), esta señal provoca que el TMS mantenga en estado de alta impedancia sus líneas de direccionamiento, datos y control. Sin embargo, de manera concurrente al modo DMA, el TMS continuará ejecutando su programa si éste opera con la RAM y ROM internas.

DIAGRAMA DE BLOQUES FUNCIONAL

La arquitectura del TMS320C25 está construída alrededor de dos buses: el de programa y el de datos. El bus de programa proporciona el código de instrucción y operandos inmediatos de la memoria de programa. El bus de datos interconecta varios elementos, como lo son la Unidad Aritmética Lógica Central (CALU) y los registros auxiliares a los datos RAM. Tanto el bus de programa como el de datos, pueden transferir datos de memoria dato RAM interna y de la memoria programa interna o externa al multiplicador en un ciclo de instrucción para operaciones de multiplicación/acumulación.

El TMS320C25 tiene un alto grado de paralelismo, es decir, que mientras está operando sobre la CALU, puede implementar operaciones aritméticas en la Unidad Aritmética de Registros Auxiliares (ARAU). Tal paralelismo resulta en un poderoso conjunto de operaciones aritméticas, lógicas y manipulación de bits que se ejecutan en un solo ciclo de máquina.

DIAGRAMA DE BLOQUES FUNCIONAL



[1] Shifters on TMS32020 (0, 1, 4)
NOTE: Shaded areas are for TMS320C25 only

ORGANIZACI0N DE LA MEMORIA

El TMS320C25 posee un total de 544-Palabras de 16 bit de RAM interna, de las cuales 288-Palabras son siempre memoria dato y las 256 restantes pueden ser configuradas como memoria programa o dato. También provee 4k-Palabras mascarables en programa ROM.

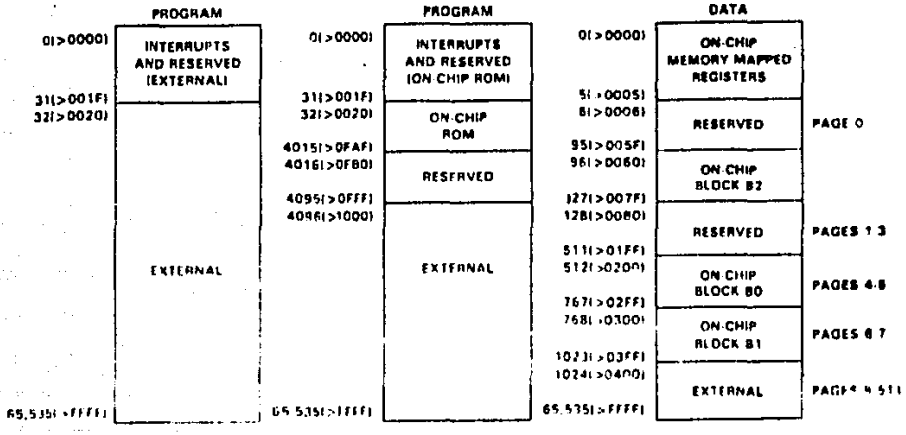
Las 544-Palabras de RAM interna se dividen en tres bloques separados: B0, B1 y B2. El bloque B0 de 256-Palabras es configurado como memoria programa o dato por medio de instrucciones (CNFP o CNPD). Como memoria dato, B0 reside en las páginas 4 y 5 del mapa de memoria de datos, y como memoria de programa de >FF00 a >FFFF, pudiéndose utilizar la instrucción BLKP (movimiento de bloque de memoria programa a memoria dato) para cargar la información del programa al bloque B0 cuando éste es configurado como RAM de datos. Las 288-Palabras (bloques B1 y B2) siempre son configuradas como memoria dato. B1 está localizado en las páginas 6 y 7 abarcando 256 localidades mientras que B2 está en las treinta y dos localidades superiores de la página 0.

La memoria dato interna y las localidades reservadas para registros son mapeadas dentro de las primeras 1024-Palabras del espacio total de memoria dato (configuración con la instrucción CNFD).

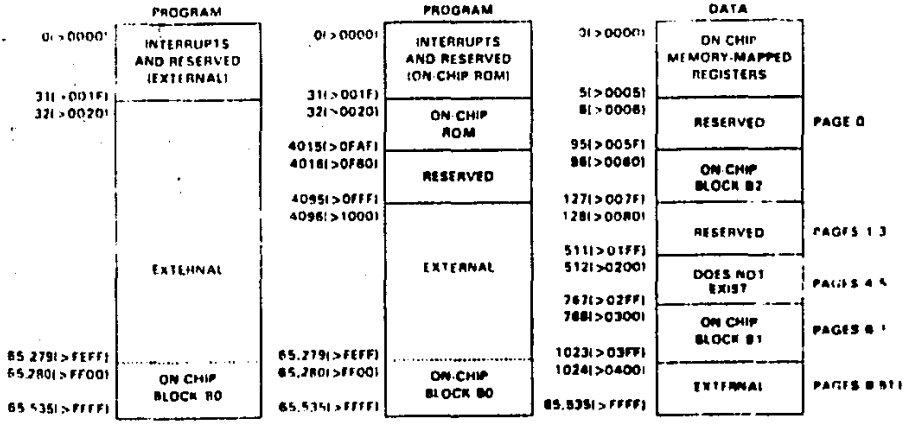
Las 4K-Palabras de ROM interna pueden ser un programa mascarable. Esta área de memoria permite la ejecución de programas a máxima velocidad sin necesidad de memorias externas veloces para almacenar el programa. El mapeo de estas 4K palabras es hecho por medio del pin de selección MP/MC (Microprocesador/microcomputadora). Manteniendo MP/MC en alto mapea este bloque de memoria como externo, y MP//MC en bajo lo mapea en ROM interna.

El TMS provee tres espacios de direccionamiento, para memoria programa, memoria dato e I/O. Estos espacios son distinguidos externamente por medio de las señales /PS, /DS e /IS (programa, dato e I/O). Las señales /PS, /DS, /IS Y /STRB son activadas sólo cuando un espacio externo de memoria está siendo direccionado. Durante un direccionamiento interno estas señales permanecen en estado inactivo alto para prevenir conflictos de direccionamiento cuando el B0 es configurado como memoria programa.

MAPA DE MEMORIA



(a) MEMORY MAPS AFTER A CNFD INSTRUCTION



(b) MEMORY MAPS AFTER A CNFP INSTRUCTION

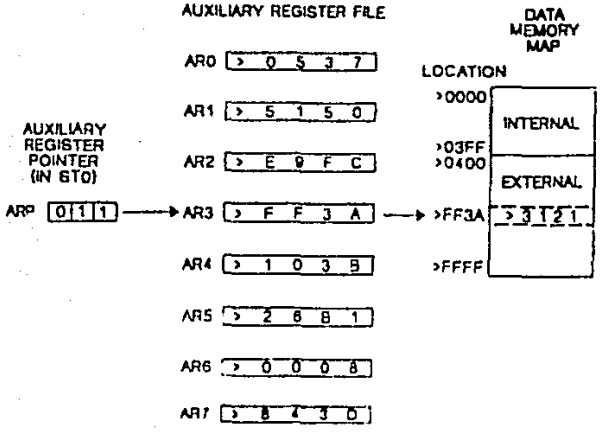
Registros mapeados en memoria

En el mapa de memoria, los registros pueden ser accesados de igual manera como un dato localizado en la memoria dato, con la excepción de que la instrucción BLKD no puede ser ejecutada en las localidades de memoria de los registros.

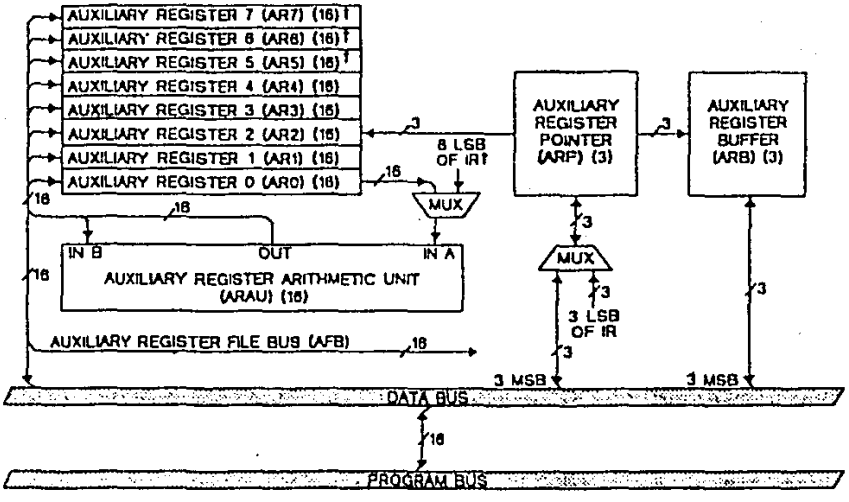
REGISTRO	LOCALIZACION	DEFINICION
DRR (15-0)	0	Receptor de dato en Puerto Serie
DXR (15-0)	1	Trasmisor de dato en Puerto Serie
TIM (15-0)	2	Timer
PRD (15-0)	3	Periodo
IMR (5-0)	4	Interrupciones mascarables
GREG(7-0)	5	Localización global de memoria

Registros Auxiliares

El TMS posee 8 Registros Auxiliares: AR0-AR7, los cuales pueden ser utilizados para direccionamiento indirecto del memoria dato, como se observa en la figura, o almacenamiento temporal de datos. Además permite posicionar una dirección de memoria de dato de un operando de una instrucción en un registro auxiliar. Estos registros son seleccionados por un Apuntador de Registros Auxiliares de tres bits (ARP), cargándose con los valores de 0 a 7 para designar desde AR0 a AR7 respectivamente.



Estos registros auxiliares están conectados a una Unidad Aritmética de Registros Auxiliares (ARAU). Esta unidad puede autoindexar el registro actual mientras que la localización del dato está siendo direccionada. Como resultado la CALU no accesa a tablas de información para manipuleo de direcciones, de este modo efectúa libremente otras operaciones.



† TMS320C25 specific.

ARAU efectúa las siguientes operaciones:

AR(ARP) + AR0 ==> AR(ARP)
AR(ARP) - AR0 ==> AR(ARP)
AR(ARP) + 1 ==> AR(ARP)
AR(ARP) - 1 ==> AR(ARP)
AR(ARP) ==> AR(ARP) AR(ARP) sin cambios
AR(ARP) + IR(7-0) ==> AR(ARP) Suma 8 bits inmediatos a AR(ARP)
AR(ARP) - IR(7-0) ==> AR(ARP) Resta 8 bits inmediatos a AR(ARP)
AR(ARP) + rcARO ==> AR(ARP) Suma AR0 con la propagación de carry reservado.
AR(ARP) - rcARO ==> AR(ARP) Resta AR0 con la propagación de carry reservado.

Aunque la ARAU es útil para la manipulación de direccionamiento en paralelo con otras operaciones, ésta puede servir como una unidad aritmética adicional de propósito general ya que los registros auxiliares pueden comunicarse directamente con la memoria de datos. La ARAU implementa aritmética no signada de 16 bits en comparación a la CALU que implementa aritmética modo dos complemento de 32 bits.

MODOS DE DIRECCIONAMIENTO DE MEMORIA

El TMS puede direccionar un total de 64K-Palabras de memoria de programa y 64K-Palabras de memoria de datos. Los 16 bits del bus de direcciones (DAB) direcciona la memoria de datos de dos formas:

1. Direccionamiento directo de bus usando modo de direccionamiento directo.
2. Por los registros auxiliares usando el modo de direccionamiento indirecto.

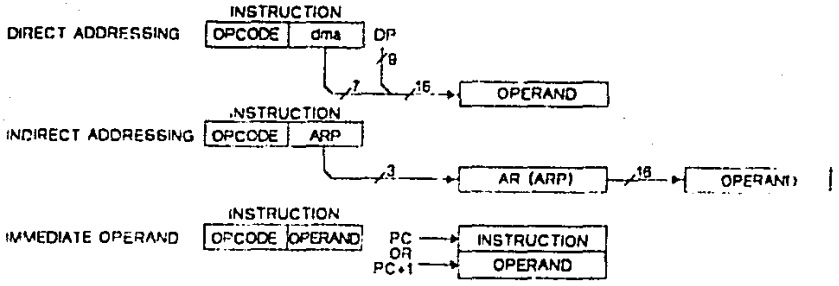
Los operandos también son direccionados por el contenido del contador de programa en el modo de direccionamiento inmediato.

Modo de direccionamiento directo

Utiliza 9 bits del apuntador de página DP, el cual puede apuntar 512 páginas de 128 palabras cada una. El dato de memoria direccionado es especificado por los 7 bits menos significativos de la instrucción para apuntar la palabra deseada dentro de la página.

Modo de direccionamiento indirecto

Los 16 bits de la dirección son seleccionados por el registro auxiliar en uso, direccionando el dato de memoria a través del bus de registro auxiliar file (AFB).



Modo operando inmediato

Cuando un operando inmediato es empleado, la palabra de la instrucción contiene el operando, en el caso de operandos inmediatos de 16 bits, éstos son la siguiente palabra del código de instrucción.

MOVIMIENTO DE MEMORIA A MEMORIA

Las instrucciones que permiten el movimiento de datos y programa, nos permiten utilizar eficazmente la configuración de la RAM interna. La instrucción BLKD mueve un bloque dentro de la memoria dato y BLKP mueve un bloque de programa a memoria de datos. Para el empleo eficiente de estas instrucciones, se utilizan las instrucciones de repetición RPT y RPTK.

La instrucción DMOV permite operar una palabra de la dirección actual en la memoria dato en RAM interna a la próxima localización alta mientras que el dato de la dirección de localización está siendo operado en el mismo ciclo por CALU. Una operación de la ARAU también puede ejecutarse en el mismo ciclo cuando se usa el modo de direccionamiento indirecto. La función de DMOV es útil en la implementación de algoritmos donde se utilizan operadores de retardo Z^{-1}, tales como convolución y filtrado digital donde el dato es pasado a través de una ventana de tiempo. La función de movimiento de dato puede ser utilizada dentro

de los bloques B0, B1 y B2. Las instrucciones TBLR y TBLW (tabla de lectura y escritura) permiten transferir palabras entre el espacio de programa y de dato.

UNIDAD ARITMETICA LOGICA CENTRAL (CALU).

Una Unidad Aritmética Lógica Central (CALU) contiene un registro de corrimiento de 16 bits, un multiplicador paralelo de 16x16 bits, una Unidad Aritmética Lógica de 32 bits, un acumulador de 32 bit (ACC) el cual está dividido en parte alta ACCH y parte baja ACCL, corrimientos de salida para el acumulador y el multiplicador. Las instrucciones SFL y SFR indican los corrimientos a la izquierda y derecha respectivamente.

Pasos en la ejecución de una instrucción típica en la ALU:

1) El dato es buscado de RAM a través del bus de datos.
2) El dato es pasado a través del registro de corrimiento y en la ALU se ejecuta la operación.
3) El resultado es movido dentro del acumulador.

Una entrada a la ALU es siempre dada por el acumulador y la otra puede ser transferida del Registro Producto (PR), del multiplicador, o del registro de corrimiento que es cargado de la memoria dato. Después de ejecutar la operación aritmética o lógica, la ALU almacena el resultado en el acumulador.

El TMS soporta operaciones de punto flotante requeridas para grandes rangos dinámicos. La instrucción NORM (normalizar) es utilizada para normalizar puntos fijos en el número del acumulador por ejecución de corrimientos a la izquierda. La instrucción LACT desnormaliza un número de punto flotante por corrimiento aritmético a la izquierda de la mantisa a través de la entrada del registro de corrimiento.

El modo de saturación de sobreflujo en el acumulador puede ser programado a través de la instrucciones SOVM y ROVM. Cuando el acumulador está en modo de saturación de sobreflujo y un sobreflujo ocurre, la bandera de sobreflujo es fijada y el acumulador es cargado con cada uno de los números más positivos o negativos, dependiendo de la dirección de sobreflujo. (>7FFFFFFF para positivo, y >80000000 para negativo).

Se pueden implementar una variedad de instrucciones de salto a una dirección específica dependiendo del estatus de la ALU y del acumulador. El acumulador tiene asociado un acarreo (carry) el cual puede ser puesto es un uno (set) o cero (reset) dependiendo de las operaciones a implementar. También existen instrucciones que permiten hacer corrimientos y rotamientos del acumulador a la izquierda o a la derecha.

Registro de Corrimiento

Puede producir corrimientos de 0 a 16 bits a un dato de entrada según sea programado en la instrucción. Los bits menos significativos se llenan de ceros y los más significativos pueden ser llenados con ceros o signo extendido, dependiendo del status programado en el modo de signo extendido SXM.

Multiplicador, Registros T y P

El TMS utiliza un hardware capaz de efectuar multiplicaciones de 16x16 bits en un ciclo de máquina. Todas las instrucciones de multiplicación exceptuando MPYU (multiplicación no signada) efectúan operaciones de multiplicación signada en el multiplicador. Es decir, que dos números pueden ser multiplicados siendo tratados como números en modo dos complemento y el resultado es un número de 32 bits complementado a dos. Los registros asociados a la multiplicación son:

- TR. Registro temporal de 16 bits, que mantiene uno de los operandos a ser multiplicado.
- PR. Registro producto de 32 bits, que mantiene el producto.

La salida del registro producto puede ser corrida a la izquierda 1 ó 4 bits, esto es útil para la implementación de aritmética fraccionaria o justificación de productos fraccionales. La salida PR también puede ser corrida a la derecha en 6 bits habilitando la posibilidad de ejecución de 128 multiplicaciones/acumulación sucesivas sin sobreflujo. El registro T normalmente es cargado con la instrucción LT, la cual le provee uno de los operandos (del bus de datos), y la instrucción MPY (multiplicación) provee el segundo operando también del bus de datos. Una multiplicación también puede ser ejecutada con un operando inmediato utilizando la instrucción MPYK. En cada uno de los dos casos el producto es obtenido cada dos ciclos.

Las instrucciones de multiplicación/acumulación (MAC y MACD) utilizan totalmente el ancho de banda de cálculo de la multiplicación permitiendo que ambos operandos sean procesados simultáneamente.

Las instrucciones SQRA (cuadrado/suma) y SQRS (cuadrado/resta) pasan el mismo valor a ambas entradas de la multiplicación para elevar al cuadrado un valor de la memoria dato.

La instrucción MPYU permite multiplicaciones no signadas, las cuales facilitan grandemente la precisión de operaciones aritméticas.

Son disponibles cuatro modos de corrimiento a través del Registro Producto (PR), los cuales son útiles cuando se ejecutan operaciones de multiplicación/acumulación, aritmética fraccional o justificación fraccional de productos. PM ocupa dos bits en el registro estatus ST1 especificando el modo del producto (PM) como se indica:

PM MODO DE CORRIMIENTO	
PM	RESULTADO
00	No hay corrimiento
01	Corrimiento a la izquierda de 1 bit
10	Corrimiento a la izquierda de 4 bit
11	Corrimiento a la derecha de 6 bit

SISTEMA DE CONTROL

Está dado por el contador de programa, el stack, la señal de reset externo, las interrupciones, los registros status, el timer interno y el contador de repetición.

El Contador de Programa y el Stack

El contador de programa (CP) consta de 16 bits, lo que permite direccionar hasta 64K direcciones de memoria programa externas o internas en la búsqueda de las instrucciones. El stack consta de 8 localidades de 16 bits para almacenar el CP durante las interrupciones o las transferencias a subrutinas.

El CP direcciona de la memoria programa vía el Bus de Dirección de Programa (PAB). A través del PAB, una instrucción es buscada de la memoria programa y cargada en el Registro de Instrucción (IR). Cuando el IR es cargado, el CP esta listo para iniciar un nuevo ciclo de búsqueda. El CP puede direccionar la RAM interna del bloque B0 cuando éste ha sido configurado como memoria de programa o la ROM interna del TMS. El CP también direcciona la memoria de programa cuando es memoria externa a través del bus externo de direcciones A15-A0 y el bus externo de datos D15-D0. Al inicio del nuevo ciclo de búsqueda, el CP es cargado con CP+1 o con una dirección de salto. En el caso de no tomar el salto el CP es incrementado una vez más de la localización de la instrucción de salto.

También el TMS tiene una característica, la cual le permite ejecutar en el próximo ciclo simple N+1 veces una instrucción, donde N es definido por la carga de 8 bits en el contador de repeticiones RPTC con la instrucción RPTK.

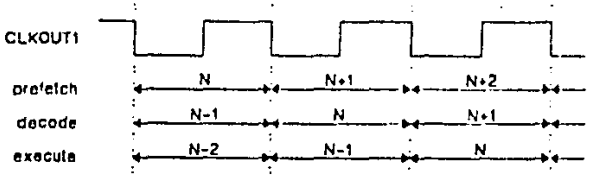
El stack es accesado a través de las instrucciones PUSH y POP. Las instrucciones adicionales de PSHD y POPD permiten utilizar el stack para el almacenamiento temporal de la memoria de datos.

La operación Pipeline

Una instrucción de Pipeline consiste en una secuencia de operaciones externas de bus que ocurren durante la ejecución interna de la instrucción. El pipeline de prebúsqueda-decodificación-ejecución es esencialmente transparente al usuario, excepto en algunos casos donde el pipeline debe ser interrumpida con instrucciones tales como saltos dentro del programa (B, BNZ,etc.). En la operación de pipeline, las operaciones prebúsqueda, decodificación y ejecución son independientes. Durante algún ciclo dado, dos o tres diferentes instrucciones pueden ser activadas, cada una a un estado diferente de completez.

Los diferentes niveles en pipeline no necesariamente afectan la velocidad de ejecución, pero sí la secuencia búsqueda/decodificación. Más instrucciones son ejecutadas en igual número de ciclos.

OPERACION DE PIPELINE A TRES NIVELES



En los tres niveles de Pipeline, un contador de prebúsqueda contiene la dirección de la próxima instrucción a ser prebuscada. Una vez que una instrucción es prebuscada, entonces es cargada en un Registro de Instrucción (IR), al menos que IR contenga la instrucción en ejecución, en cuyo caso la instrucción prebuscada es cargada en un Registro de Instrucción a la Cola (QIR). El contador de prebúsqueda es incrementado y después la instrucción actual completa la ejecución, la instrucción en QIR es cargada en IR para ser ejecutada.

El CP contiene la dirección de la próxima instrucción a ser ejecutada, sin ser usada directamente en las operaciones de búsqueda, pero sirve como un apuntador de referencia para la posición actual dentro del programa. El CP es incrementado cada vez que una instrucción se ejecuta. Cuando ocurre una interrupción o una llamada de subrutina, el contenido del CP es guardado en el stack para preservar su contenido al regreso de una interrupción o subrutina.

La prebúsqueda, decodificación y ejecución de la operación de pipeline son independientes, permitiendo así la ejecución de operaciones traslapadas. Durante algún ciclo, tres instrucciones diferentes pueden ser activadas cada una a diferente estado de completez.

La operación básica de pipeline es de 3.25 ciclos de duración, donde la secuencia de un dispositivo sobre algún ciclo dado es buscada una tercera instrucción, decodificada una segunda y ejecutada una primera. Si la instrucción es buscada en la RAM interna, el pipeline es acortado a 2.5 ciclos, dado que TMS puede buscar la instrucción en la mitad del ciclo, contrario a lo que ocurre cuando se busca en la ROM externa que requiere de un ciclo completo para la búsqueda.

Cuando existen estados de espera, sólo retardan directamente la operación de pipeline. Cada estado de espera insertado durante la operación de búsqueda contribuye a agregar ciclos adicionales de máquina en la ejecución de pipeline de la instrucción.

Acceso externo de memoria de programa o datos

La visibilidad de pipeline requiere el monitoreo de señales /MSC, /STRB, /PS y /DS. La señal /MSC indicada en la caída de CLKUOT2 ya sea o no el ciclo de una nueva instrucción de búsqueda, es decir, que /MSC activa indica la terminación de una instrucción y la adquisición de otra instrucción. /PS indica que el bus de datos está siendo utilizado para la búsqueda de una instrucción (aunque no necesariamente la activación de esta señal indica la búsqueda de una instrucción). El monitoreo de la búsqueda de memoria dato, debe hacerse seleccionando /DS en conjunto con /STRB.

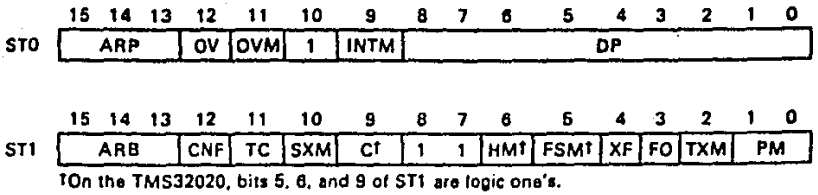
Reset (/RS)

Es una interrupción externa no mascarable, con la más alta prioridad sobre las demás interrupciones. Puede ser usada en cualquier instante y es aplicada típicamente en el encendido cuando los estados de máquina son aleatorios.

La señal /RS causa que el TMS termine la ejecución y force al contador de programa ir a cero afectando varios registros y los bits de status. Para la correcta operación del sistema, la señal de reset debe ser insertada baja por lo menos tres ciclos de reloj. El bus de datos es puesto en alta impedancia, y las señales de control se van a alto mientras dura el reset.

Registros de Status

Los registros de status ST0 y ST1, contiene el status de varias condiciones y modos. Los registros status pueden ser almacenados dentro de la memoria dato y cargados de memoria dato, permitiendo así que el status de la máquina sea salvado y restaurado para interrupciones y subrutinas. Todos los bits de status pueden ser escritos y leidos utilizando las instrucciones LST/LST1 y SST/SST1 (con excepción de INTM, el cual puede ser cargado por la instrucción LST).



†On the TMS32020, bits 5, 6, and 9 of ST1 are logic one's.

DONDE:

ARB: Bufer Apuntador de Registros Auxiliares.
ARP: Apuntador de Registros Auxiliares.
C: Bit de Carry.
CNF: Bit de control de configuración de RAM interna.
DP: Apuntador de página de Memoria Dato.
FO: Bit de Formato, cada puerto serie.
FSM: Bit de Modo de Estructura de Sincronización.
HM: Bit de Modo Hold.
INTM: Bit de modo de interrupción, 0 = habilitado, 1 = deshabilitado.
OV: Bit de bandera de sobreflujo.
OVM: Bit de modo de sobreflujo.
PM: Modo de corrimiento de producto.
SXM: Bit de modo de signo extendido.
TC: Bandera de control de prueba de bit.
TMX: Bit de modo de transmisión.
XF: Bit de estatus del pin XF.

Operación del Timer

El TMS posee un Registro Timer (TIM) y un Registro Período (PRD) de 16 bit. El timer cuenta descendentemente en la caída de CLKOUT1. Los registros TIM y PRD son fijados a su máximo valor >FFFF en el reset. Una interrupción de Timer (TINT) es generada cada vez que el TIM llega a cero, el TIM es recargado con el valor de PRD en el próximo ciclo. Esta característica es útil para operaciones de control y sincronización de muestreos o escritura de periféricos.

Si el timer no es utilizado, TINT debe ser mascarable o todas las interrupciones deben ser deshabilitadas por la instrucción DINT.

Contador de repetición (RPTC)

Es de 8 bits y cuando es cargado con un número N, causa que la siguiente instrucción sea ejecutada N+1 veces. El RPTC puede ser cargado con un número e 0 a 255 usando las instrucciones RPT o RPTK. Es utilizado en las instrucciones de multiplicación/acumulación, movimiento de bloques, transferencias de I/O y tablas de lectura/escritura.

Memoria externa e interfase I/O

La interfase local de memoria consiste de:

- Bus de datos paralelos de 16 bit D15-D0
- Bus de direcciones A15-A0
- Señales de selección de espacio programa, dato y I/O, /PS, /DS y /IS, respectivamente.
- Varias señales del sistema de control.

La señal de lectura/escritura R//W provee la dirección de transferencia y la señal /STRB el tiempo de transferencia de control.

El espacio I/O consiste de 16 puertos de entrada y 16 de salida, con 16 bit para datos de entrada o salida. Para entrada o salida se utilizan las instrucciones de IN y OUT respectivamente, que típicamente ocupan dos ciclos; sin embargo, con el contador de repeticiones la operación se convierte en un ciclo simple.

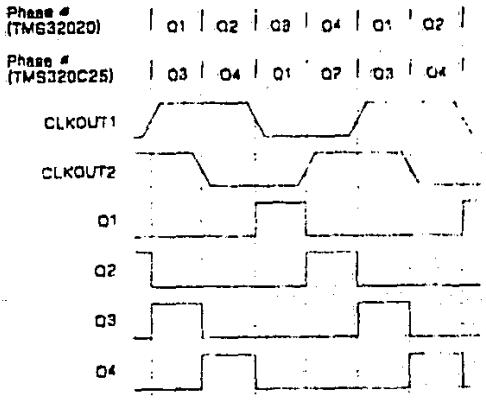
COMBINACIÓN DE MEMORIA

La secuencia exacta de las operaciones ejecutadas de una instrucción dependen de las áreas de memoria donde la instrucción y los operandos son localizados. Las seis posibles combinaciones de programa y dato, dada la información pueden ser localizadas en memoria RAM interna, memoria ROM externa o interna:

PI/DI: Programa interno RAM/dato interno.
PI/DE: Programa interno RAM/dato externo.
PE/DI: Programa externo/dato interno.
PE/DE: Programa externo/dato externo.
PR/DI: Programa interno ROM/dato interno.
PR/DE: Programa interno ROM/dato externo.

Relaciones de tiempo de reloj interno

El cristal o la fuente de frecuencia externa es dividida para producir cuatro fases de reloj. Las cuatro fases son definidas CLKOUT1 y CLKOTU2.



Interrupciones

El TMS tiene tres interrupciones externas mascarables (/INT2, /INT1, /INT0), disponibles para dispositivos externos que interrumpan al microprocesador. Dos interrupciones internas que son generadas por el puerto serie (RINT y XINT), una por el timer (TINT) y una por software a través de la instrucción TRAP. Aunque esta última no es priorizable incluye su vector de interrupción.

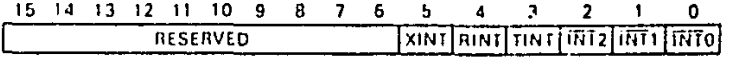
INTERRUPT NAME	MEMORY LOCATION	PRIORITY	FUNCTION
$\overline{RS}$	0	1 (highest)	External reset signal
$\overline{INT0}$	2	2	External user interrupt #0
$\overline{INT1}$	4	3	External user interrupt #1
$\overline{INT2}$	6	4	External user interrupt #2
	8-23		Reserved locations
TINT	24	5	Internal timer interrupt
RINT	26	6	Serial port receive interrupt
XINT	28	7 (lowest)	Serial port transmit interrupt
TRAP	30	N/A	TRAP instruction address

Operación de las interrupciones

Cada dirección de interrupción ha sido espaciada cada dos localidades, de tal modo que se puedan acomodar instrucciones para realizar saltos cuando así se desea. Cuando una

interrupción ocurre, esta es almacenada en un Registro de Bandera de Interrupción de 6 bits (IFR). Este registro es fijado por el uso de interrupciones externas /INT(2-0) y las interrupciones RINT, XINT y TINT. Cada interrupción es almacenada en IFR hasta que ésta es reconocida y automáticamente borrada por el Reconocimiento de Interrupciones (/IACK) o el reset (/RS). /RS no es almacenado en IFR. Ninguna instrucción puede leer o escribir en IFR.

El TMS tiene una mapa de memoria para Registro de Interrupciones Mascarables (IMR) para mascarar las interrupciones internas y externas.



IMR habilita la correspondiente interrupción colocando un cero. El IMR es accesible para operaciones de lectura y escritura excepto cuando se usa BLKD. IMR no es afectado por reset.

El modo de interrupción INTM en el registro de status ST0 habilita o deshabilita todas las interrupciones mascarables (INTM = 0 habilitado y INTM = 1 deshabilitado). INTM es fijado a uno por la señal /IAK, la instrucción DINT (deshabilita interrupciones) y por reset, y en cero por la instrucción EINT (habilita interrupciones).

Si una interrupción ocurre durante un ciclo múltiple de instrucciones, la interrupción no puede ser procesada hasta que la instrucción es completada, este mecanismo de protección también es aplicado a instrucciones de ciclo múltiple debido a la señal de READY. Tampoco se permite que se procesen interrupciones cuando una instrucción está siendo repetida por medio de RPTK, la interrupción es almacenada en IFR hasta que el ciclo de repetición sea terminado, después la interrupción es procesada. Las interrupciones no pueden ser procesadas entre la instrucción EINT y la próxima instrucción en la secuencia del programa. También una interrupción no es reconocida cuando el /HOLD está activado.

Multiprocesamiento y acceso directo de memoria (DMA)

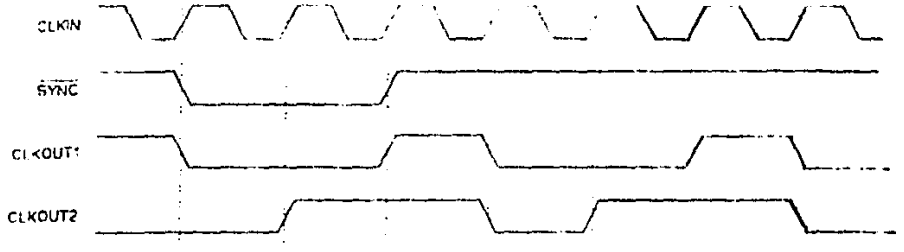
El TMS puede ser configurado para satisfacer amplios rangos de requerimiento de sistemas. Algunas configuraciones son:

- Sistema estándar (proceso simple).
- Multiprocesamiento con dispositivos en paralelo.
- Multiprocesos Huésped/Esclavo con espacio de memoria dato compartida.
- Proceso a periféricos interfaseados utilizando señales de control de procesos para otros dispositivos.

Estas configuraciones son posibles utilizando las señales externa de entrada /SYNC, la interfase de memoria global y la implementación de la función hold con los pines de /HOLD y /HOLDA.

Sincronización

Esta señal externa de entrada puede ser utilizada con gran facilidad entre procesos, causando que cada TMS en el sistema se sincronice con su reloj interno, permitiendo que el proceso corra amarrado a los pasos de la operación. La transición negativa de /SYNC fija a cada proceso a un cuarto de fase interna (Q1). Esta transición debe ocurrir sincronizada con la bajada del reloj CLKIN. En el TMS hay dos ciclos de retardo CLKIN que permiten que el ciclo en el cual /SYNC vaya abajo, por lo que ocurre la sincronización Q1.



Normalmente /SYNC es aplicado mientras /RS está activado, para evitar situaciones no predecibles del procesador.

MEMORIA GLOBAL

Para aplicaciones de multiprocesamiento, el TMS tiene la capacidad de localizar un espacio de memoria global de datos y comunicarse con este espacio a través de /BR (requerimiento de bus) y la señal de control READY.

La memoria global es compartida por más de un procesador, por lo tanto para su acceso debe ser arbitrado. Cuando se utiliza memoria global, el espacio de direccionamiento del procesador es dividido en secciones local y global. La sección local es usada por el procesador para ejecutar funciones individuales y la sección global para comunicarse con otros procesadores.

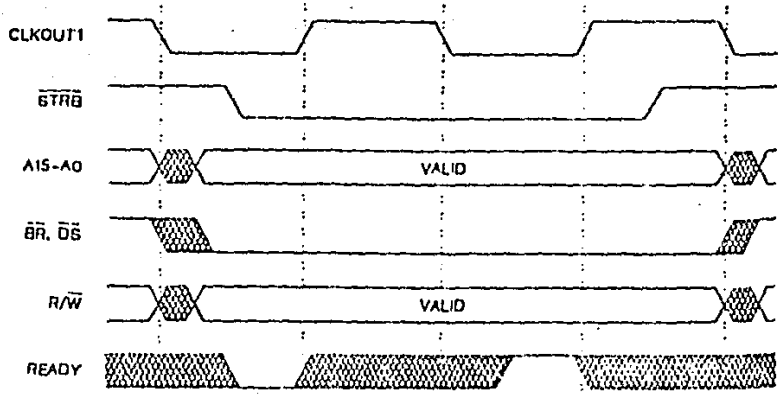
El Registro de Mapeo de Memoria Global (GREG) especifica la parte de memoria de datos del TMS como memoria global externa. GREG es mapeada como memoria de datos en la localización 5, es un registro de 8 bit conectados con los 8 bits menos significativos del bus de datos.

El contenido de GREG determina el tamaño del espacio de memoria global. El valor de GREG y el correspondiente espacio es indicado en la tabla:

GREG VALUE	LOCAL MEMORY RANGE	# WORDS	GLOBAL MEMORY RANGE	# WORDS
000000XX	>0 - >FFFF	65,536	----------	0
10000000	>0 - >7FFF	32,768	>8000 - >FFFF	32,768
11000000	>0 - >BFFF	49,152	>C000 - >FFFF	16,384
11100000	>0 - >DFFF	57,344	>E000 - >FFFF	8,192
11110000	>0 - >EFFF	61,440	>F000 - >FFFF	4,096
11111000	>0 - >F7FF	63,488	>F800 - >FFFF	2,048
11111100	>0 - >FBFF	64,512	>FC00 - >FFFF	1,024
11111110	>0 - >FDFF	65,024	>FE00 - >FFFF	512
11111111	>0 - >FEFF	65,280	>FF00 - >FFFF	256

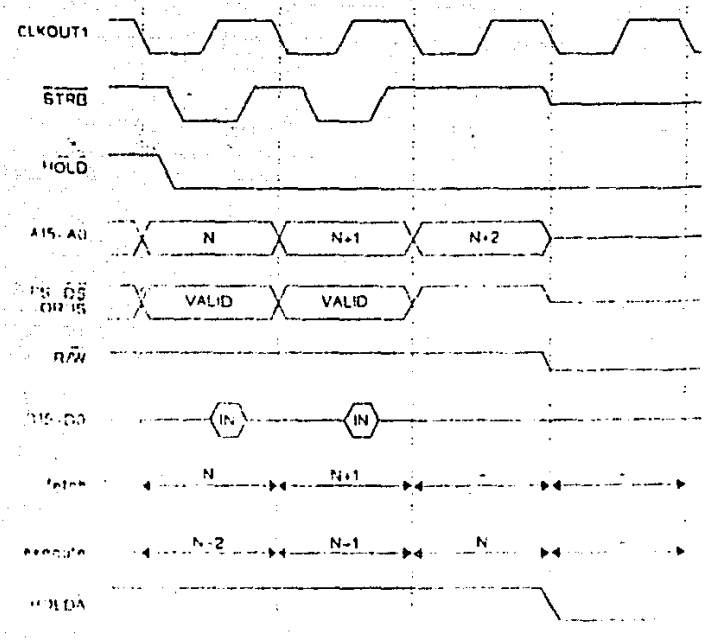
Cuando un dato de memoria es direccionado, cada uno directa o indirectamente, correspondiente a direccionamiento de memoria global (definido por GREG), /BR es acertado bajo con /DS para indicar al procesador que se desea hacer una acceso de memoria global.

La lógica externa arbitra el control de memoria global acertando READY cuando tiene el control.



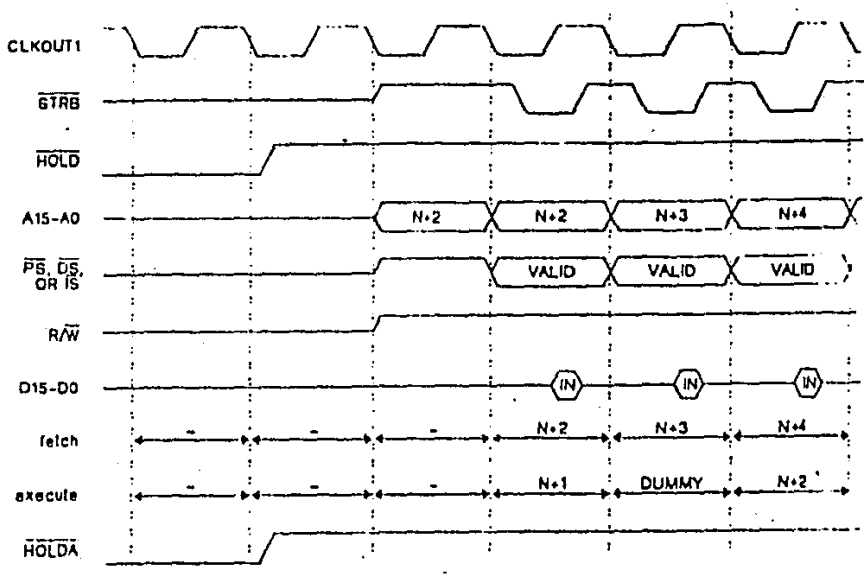
Función HOLD

El TMS soporta Acceso Directo de Memoria (DMA) a un programa local, dato ó espacio I/O. Las dos señales /HOLD y /HOLDA permiten a otros dispositivos tomar el control de los buses del procesador. Cuando el procesador recibe una señal de /HOLD de un dispositivo externo, el procesador envía una señal de reconocimiento /HOLDA (activa baja); entonces pone sus buses de direcciones y de datos así como las señales de control (/PS, /DS, /IS, R//W y /STRB) en alta impedancia. Los pines de puerto serie (DX y FSX) no son afectados. /HOLD es muestreado en el tercer cuarto de fase, si es encontrado, éste toma tres ciclos de máquina antes que los buses y señales se vayan a alta impedancia, permitiendo terminar la instrucción que está siendo ejecutada.



NOTES: 1. N is the program memory location for the current instruction.
2. This example only shows the execution of single cycle instructions fetched from external program memory.

/HOLDA es removido asíncronamente por /HOLD.



El modo de operación es seleccionada por HM (modo hold) en el registro de status. Cuando HM = 1, el TMS detiene la ejecución del programa e ingresa al estado de hold directamente. Cuando HM = 0 el procesador puede continuar la ejecución fuera de la memoria de programa interna pero pone la interfase externa en alta impedancia. Si el programa en ejecución es de memoria interna y no se accesa dato externo de memoria, el procesador entra a estado externo de hold, pero el programa sigue ejecutándose internamente. Esto permite una operación más eficiente del sistema, dado que puede continuarse la ejecución mientras que una operación de DMA está siendo ejecutada.

La ejecución de programa cesa hasta que /HOLD es removido si el procesador está en un estado hold con HM = 0 y una ejecución interna requiere un acceso de programa externo, o si la rama del programa va a una dirección externa. Todas la interrupciones son deshabilitadas mientras que /HOLD es activada con HM = 1. Si HM=0 las interrupciones funcionan normalmente.

DIAGRAMA DE TIEMPO DE LECTURA Y ESCRITURA

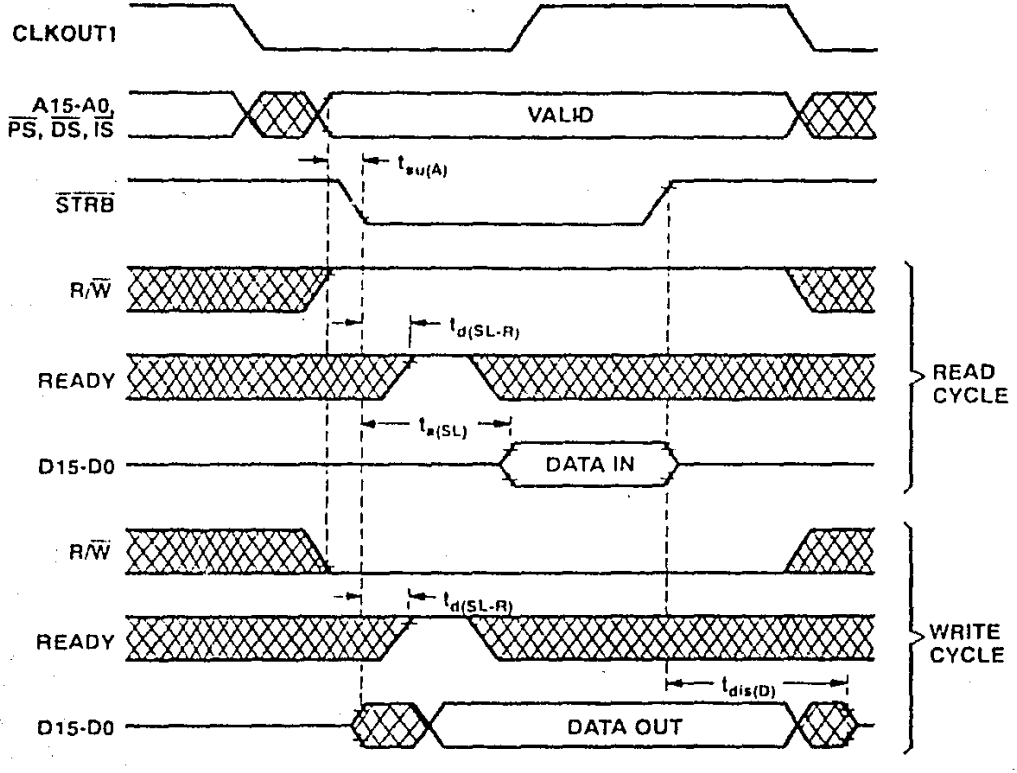
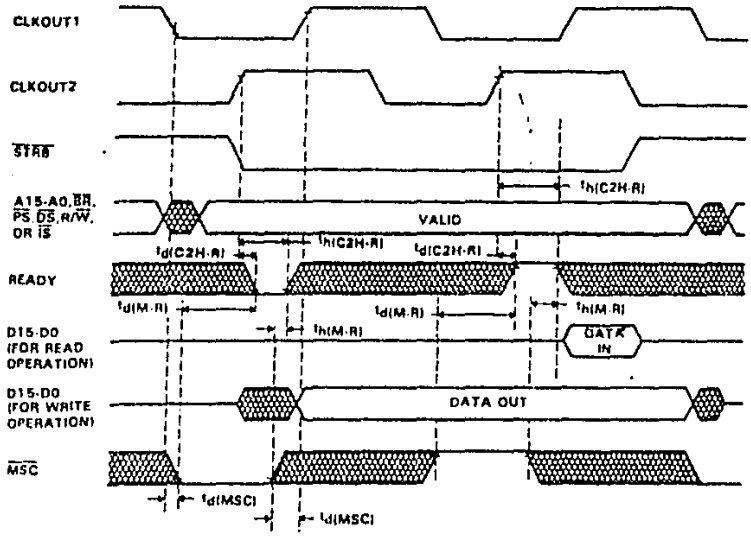
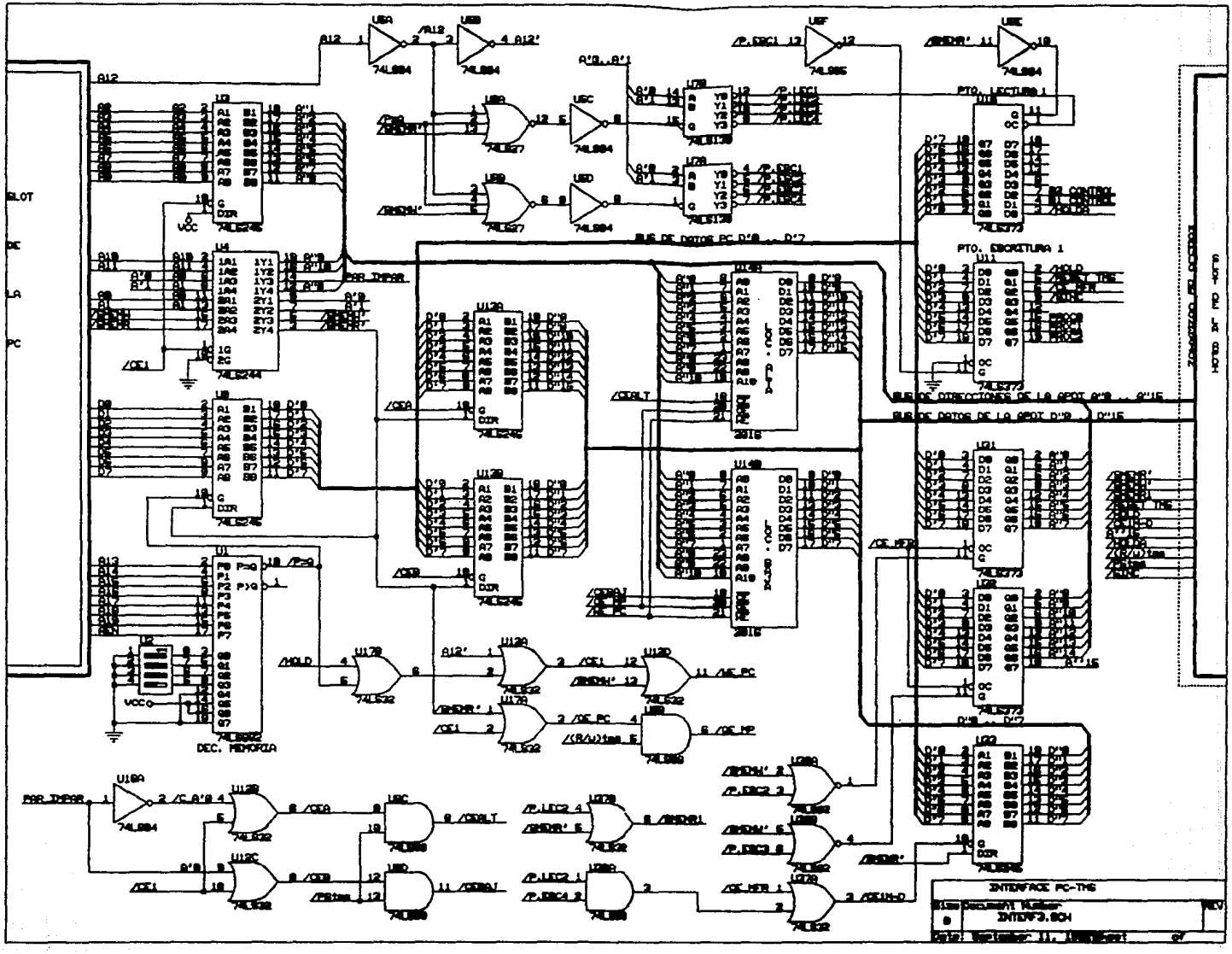
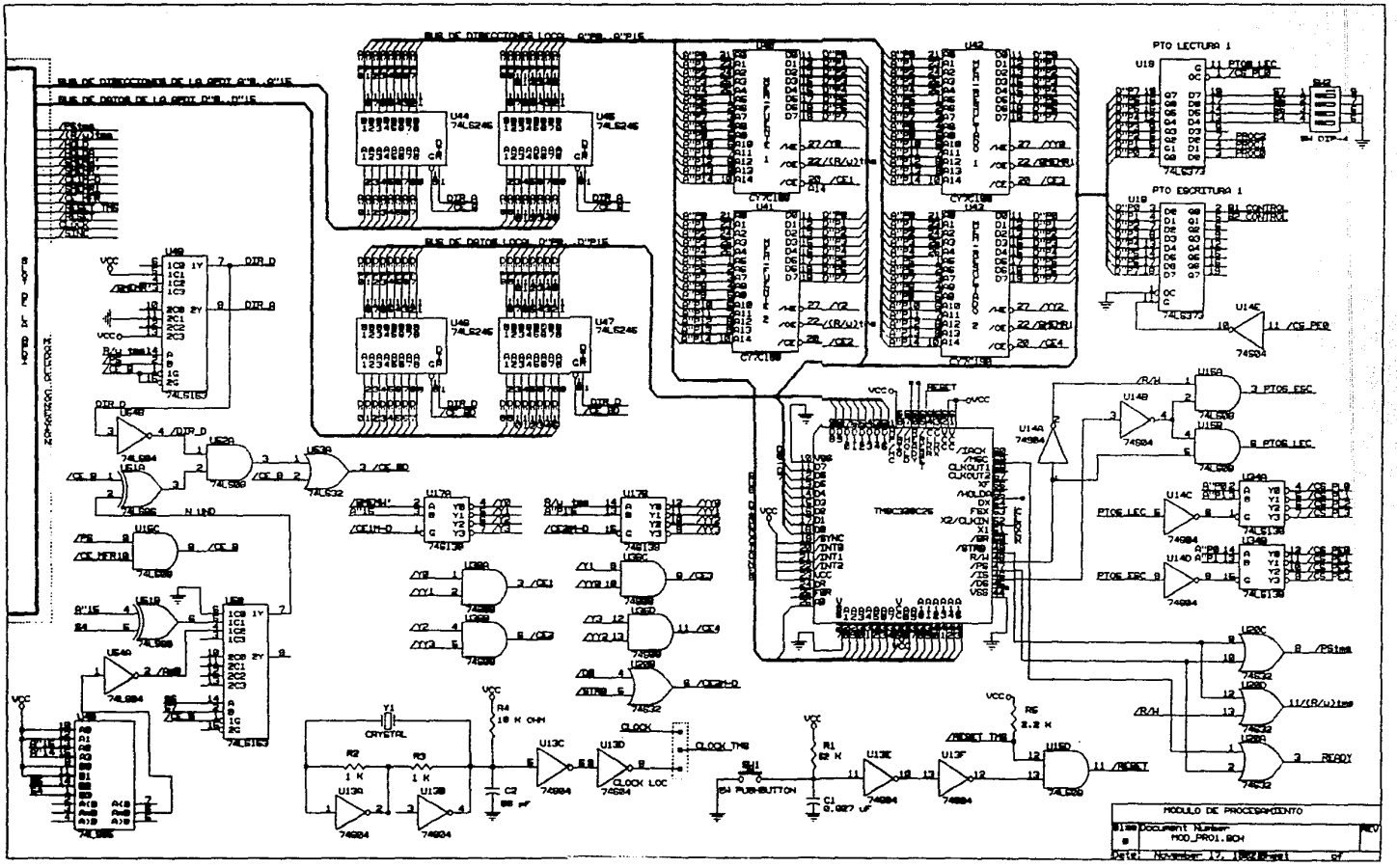


DIAGRAMA DE TIEMPO PARA UN ESTADO DE ESPERA



ANEXO B
DIAGRAMAS ELECTRONICOS


SLOT DE LA PC
A12
/A12
A'8..A'1
PTO. LECTURA 1
PTO. ESCRITURA 1
BUS DE DATOS PC D'8 .. D'7
BUS DE DIRECCIONES DE LA GPDT D'8 .. D'15
BUS DE DATOS DE LA GPDT D'8 .. D'15
PWR THRSH
/OE1
/OEA
/OEB
/OEALT
/OEBALT
/MOLD
/OE_PC
/OE_MP
/OE_MEM
/OE1N-O
DEC. MEMORIA
INTERFACE PC-THS
INTERF3.SCH


BUS DE DIRECCIONES LOCAL A'15..A'00
BUS DE DATOS LOCAL D'15..D'00
BUS DE DIRECCIONES DE LA MEM A'15..A'00
BUS DE DATOS DE LA MEM D'15..D'00
PTO LECTURA 1
PTO ESCRITURA 1
MODULO DE PROCESAMIENTO
MOD_PRO1.SCH
CRYSTAL
CLOCK
CLOCK_TMS
/RESET_TMS
PUSH-BUTTON
/RESET
/READY
/PS1ms
/PTOS_ESC
/PTOS_LEC

ANEXO C
PROGRAMA DE APLICACION EN TURBOPASCAL

```
{**********************************************************
**********************************************************
**                                                      **
**  PROGRAMA DE APLICACION PARA: LA ADMINISTRACION DE LOS **
**  RECURSOS DE LA APDI Y LA EVALUACION DE LA MISMA.     **
**                                                      **
**********************************************************
**********************************************************}

PROGRAM PROGRAMA_DE_APLICACION;
{$I-}

{**********************************************************
* EMPLEO DE LA UNIDADES DE SERVICIO DE: DOS, MONITOR Y GRAFICOS *
**********************************************************}
USES DOS,CRT,GRAPH;

{**********************************************************
**  LISTA DE LAS VARIABLES EMPLEADAS EN EL PROGRAMA      **
**********************************************************}

VAR

 Cond_esc          { Reg aux de escritura de imagen      }
                : INTEGER;

 Bandera,Ban_fin,
 Holda,Holda2,     { Variables auxiliares de control     }
 Proceso           { Numero de proceso a realizar        }
                : BYTE;

 Hora_ini,
 Min_ini,
 Seg_ini,
 Cseg_ini,
 Hora_fin,
 Min_fin,
 Seg_fin,          { Variables utilizadas en el calculo del  }
 Cseg_fin,         { tiempo de duracion del procedimiento    }
 Cont              { Regs apunt en lec y escr de Imagenes    }
                : WORD;

 Archivo_Img       { Nombre de la Imagen Fuente          }
                : STRING;

 Lect,             { Variables para despliegue o almacena-  }
```

```
 Resp1,Resp2      ( miento de datos resultados          )
                : STRING[1];

 Cond_salida      ( Reg de condicion de salida          )
                : BOOLEAN;

{*************************************************************
**      LISTA DE CONSTANTES EMPLEADAS EN EL PROGRAMA          **
*************************************************************}

CONST
  Dir_Mem_base = $D000;  ( Direc Base de Memoria desde PC     )
  Dir_Pto_Base = $D100;  ( Direc Base de Puertos desde PC     )
                 ( Programables en microswitches              )

{*************************************************************
**                                                           **
**  PROCEDIMIENTO DE CONVERSION DE LOS DATOS PERTENECIENTES   **
**  A LOS ARCHIVOS DE EXTENSION .MPO, DE UN FORMATO TIPO      **
**  TEXTO A UNO TIPO BYTE. PARA SU POSTERIOR TRANSFERENCIA A  **
**  LA MEMORIA DE PROGRAMA EN LA APDI.                        **
**                                                           **
*************************************************************}

PROCEDURE CONVERSION(Dig:string;var Val:byte);
BEGIN
   if Dig = '0'then Val := $00; if Dig = '1'then Val := $01;
   if Dig = '2'then Val := $02; if Dig = '3'then Val := $03;
   if Dig = '4'then Val := $04; if Dig = '5'then Val := $05;
   if Dig = '6'then Val := $06; if Dig = '7'then Val := $07;
   if Dig = '8'then Val := $08; if Dig = '9'then Val := $09;
   if Dig = 'A'then Val := $0A; if Dig = 'B'then Val := $0B;
   if Dig = 'C'then Val := $0C; if Dig = 'D'then Val := $0D;
   if Dig = 'E'then Val := $0E; if Dig = 'F'then Val := $0F;
END;

{*************************************************************
**                                                           **
**  PROCEDIMIENTO DE ESCRITURA DEL PROGRAMA EN LENGUAJE       **
**  ENSAMBLADOR A LA MEMORIA DE PROGRAMA, A PARTIR DE UN      **
**  ARCHIVO GENERADO POR EL PROGRAMA ENSAMBLADOR TMS32025.    **
**  LA TRANSFERENCIA SE REALIZA AL ESPACIO CORRESPONDIENTE     **
**  A LA EXTENSION DE MEMORIA DE LA PC, DONDE ESTA UBICADA    **
**  LA MEMORIA DE PROGRAMA (MP) DE LA APDI.                   **
**                                                           **
*************************************************************}
```

```
PROCEDURE ESCRITURATMS;

{**********************************************************
**      LISTA DE LAS VARIABLES EMPLEADAS EN EL PROCEDURE      **
**********************************************************}

VAR

  Dig1b,          { Reg aux en la lectura del archivo MPO   }
  Dig2b,          {  "    "   "  "   "    "        }
  Dig3b,          {  "    "   "  "   "    "        }
  Dig4b,          {  "    "   "  "   "    "        }
  DireccM,        { Direc Alta de dato archivo  OBJ       }
  DireccL,        {  "  Baja "  "     "    "         }
  ValorM,         { Valor parte Alta de dato archivo OBJ    }
  ValorL,         {  "    "   Baja "  "     "    "     }
  Valor           { Reg de lectura de datos en arch tipo OBJ }
                : BYTE;

  Cont1           { Regs apunt en escr de memoria MP       }
                : WORD;

  Dig1d,          { Reg aux  en la escritura a mem MP      }
  Dig2d,          {  "    "   "  "   "    "        }
  Dig3d,          {  "    "   "  "   "    "        }
  Dig4d,          {  "    "   "  "   "    "        }
  Guia            { Variable guia en conversion MPO a OBJ   }
                : STRING[1];

  Linea           { Variable aux en lectura archivo MPO     }
                : STRING[5];

  Nombre_Archivo    { Nombre archivo tipo MPO               }
                : STRING[30];

  Archivo_Fuente    { Archivo fuente tipo MPO               }
                : TEXT;

  Cond_sal        { Reg de condicion de salida              }
                : BOOLEAN;

  Informa         { Variable de informacion de archivos     }
                : PATHSTR;
```

```
BEGIN
  Clrscr;
  REPEAT
   Write( 'Nombre del archivo fuente ( ARCHIVO.MPO ) : ');
   Readln(Nombre_Archivo);
   Informa := FSearch(Nombre_Archivo,GetEnv('PATH'));
   IF Informa = '' THEN Writeln(' NO SE ENCONTRO EL ARCHIVO ');
   Writeln;
  UNTIL Informa <> '';

{  Nombre_Archivo := 'C:\tms25\proconvo.mpo';
}  Assign(Archivo_Fuente,Nombre_Archivo);
  Reset(Archivo_Fuente);

{***** RESET A MEMORIA DATOS *****}
  Cont1:=$00;
  REPEAT
   Mem[Dir_Mem_base:CONT1]:=$00;
   Inc(Cont1);
  UNTIL Cont1=$FA0;
  Cond_sal := True;
  Clrscr;
  Writeln(' ESCRIBIENDO EN LA MEMORIA DE PROGRAMA ');
  WHILE  Cond_sal = True DO
  BEGIN
   Read(Archivo_Fuente,Guia);
   IF Guia = '9' THEN
   BEGIN
    Read(Archivo_Fuente,Dig1d,Dig2d,Dig3d,Dig4d);
    CONVERSION(Dig1d,Dig1b); CONVERSION(Dig2d,Dig2b);
    CONVERSION(Dig3d,Dig3b); CONVERSION(Dig4d,Dig4b);
    DireccM := Dig1b shl 4 + Dig2b;
    DireccL := Dig3b shl 4 + Dig4b;
   END
   ELSE IF Guia = 'B' THEN
      BEGIN
       Read(Archivo_Fuente,Dig1d,Dig2d,Dig3d,Dig4d);
       CONVERSION(Dig1d,Dig1b); CONVERSION(Dig2d,Dig2b);
       CONVERSION(Dig3d,Dig3b); CONVERSION(Dig4d,Dig4b);
       ValorM := Dig1b shl 4 + Dig2b;
       ValorL := Dig3b shl 4 + Dig4b;
       Cont1 := DireccM shl 8 + DireccL;
```

```
{*****************************************************************
******      ESCRITURA DEL DATO EN MEMORIA PROGRAMA              ******
*****************************************************************}
        MEM[Dir_Mem_base:Cont1 * 2] := ValorL;
        MEM[Dir_Mem_base:(Cont1 * 2 + $01)]: = ValorM;

        IF DireccL = $FF THEN
        BEGIN
          DireccL := $00;
          DireccM := DireccM + $01;
        END
        ELSE DireccL := DireccL + $01;
       END
       ELSE IF Guia = '7' THEN Readln(Archivo_Fuente,Linea)
          ELSE IF Guia = ':' THEN Cond_sal := False;

  END; { fin while }

{*****************************************************************
******      VERIFICACION LA ESCRITURA DE DATOS                  ******
*****************************************************************}
{  Cont1: = $00;
   REPEAT
    Valor := Mem[Dir_Mem_base:Cont1];
    Writeln('DIRECCION  =  ',Cont1,'  DATO  =  ',Valor);
    Delay(100);
    Inc(Cont1);
   UNTIL Cont1 = $AA;
}
  Close(Archivo_Fuente);
  Writeln;
  Write(' FINALIZA ESCRITURA EN MEMORIA, ENTER para continuar ');
  Readln;
END;


{*****************************************************************
**                                                              **
**  PROCEDIMIENTO DE DESPLIEGUE DE IMAGENES EN PANTALLA,        **
**  PREVIAMENTE GRABADAS EN UNA ARCHIVO CON UN TAMANO DE        **
**  200x200 BYTES.                                              **
**                                                              **
*****************************************************************}

PROCEDURE DESPLIEGUE(Imagen_Fuente:string);
```

```
VAR
  Archivo_B    { Tipo del archivo de la Imagen Fuente IF      }
          : file of byte;

  Mode,        { Modo de configuracion del monitor            }
  Driver,      { Tipo del monitor                             }
  I,J          { Regs auxiliares en el despliegue de imagenes }
          : integer;

  Var1         { Valor de pixel a desplegar                   }
          : byte;

BEGIN
  Driver := CGA;
  Mode := 0;
  InitGraph(driver,mode,'c:\TP500');
  Assign(archivo_B,Imagen_fuente);
  Reset(archivo_B);
  FOR I := 1 TO 200 DO
  BEGIN
    FOR J := 1 TO 200 DO
    BEGIN
      Read(Archivo_B,Var1);
      PutPixel(J,I,Var1);
    END;
  END;
  Delay(1000);
  SetColor(2);
  SetTextStyle(0,0,1);
  OutTextXY(210,160,'ENTER');
  OutTextXY(210,170,'PARA');
  OutTextXY(210,180,'CONTINUAR');
  Readln;
  Close(Archivo_B);
  Closegraph;
END;

{**************************************************************
**                                                           **
** PROCEDIMIENTO DE CREACION DE UN MENU DE SELECCION, CON    **
** EL OBJETO DE PROPORCIONAR LA IMAGEN A PROCESAR Y EL       **
** PROCESO A EFECTUAR EN LA MISMA. ADEMAS DE DETERMINAR LAS  **
** PALABRAS DE CONTROL PARA LA EJECUCION DEL PROCESAMIENTO.  **
**                                                           **
**************************************************************}
```

```
PROCEDURE MENU(var Proceso_selec:byte;Cond:integer;
          var Imagen_selec:string;var Cont:word);

{**************************************************************
**    LISTA DE LAS VARIABLES EMPLEADAS EN EL PROCEDURE        **
**************************************************************}

VAR

 Lectura,Resp      { Regs auxs en la eleccion de imagenes     }
               : STRING[1];


 Infor_imagen      { Variable de informacion de imagenes      }
               : PATHSTR;

{**************************************************************
**    LISTA DE LAS CONSTANTES EMPLEADAS EN EL PROCEDURE       **
**************************************************************}

CONST
    VALFINAL = 41024;  { TAMANO DE LA IMAGEN A PROCESAR      }



BEGIN

{**************************************************************
*****  ENVIO DE LA SENAL DE /HOLD PARA MANTENER AL TMS25     ****
*****  EN ESTADO DE ESPERA Y SUS BUSES EN ALTA IMPEDANCIA.   ****
*****  ADEMAS SE ENVIAN UNOS A LOS BITS QUE INFORMAN         ****
*****  EL PROCESO A REALIZAR EN LA IMAGEN.                   ****
**************************************************************}
  MEM[Dir_Pto_base:0000] := $FE;
  Delay(5);
  Clrscr;
  IF Cond = 0 THEN
    REPEAT
      Write('PROCESAR A LA MISMA IMAGEN  S/N : ');
      Readln(Lectura);
      Resp := UpCase(Lectura[1]);
    UNTIL ( (Resp='N') or (Resp='S') );
  IF Resp = 'S' THEN Cont := VALFINAL;
```

```
IF ( (Resp <> 'S') or (Cond = 1) ) THEN BEGIN
  REPEAT
    Writeln(' NOMBRE DE LA IMAGEN A PROCESAR (INCLUIR PATH)');
    Readln(Imagen_selec);
    Infor_imagen := FSearch(Imagen_selec,GetEnv('PATH'));
    IF Infor_imagen = '' THEN Writeln(' NO SE ENCONTRO LA IMAGEN ');
    Writeln;
  UNTIL Infor_imagen <> '';
  Cond := 1;
  END;
REPEAT;
  Clrscr;
  Writeln;
  Writeln;
  Writeln;
  Writeln( ' M E N U   D E   P R O C E S A M I E N T O S');
  Writeln;
  Writeln;
  Writeln(' 1) FILTRO PASO ALTAS  ');
  Writeln;
  Writeln(' 2) FILTRO PASO BAJAS  ');
  Writeln;
  Writeln(' 3) FILTRO DIFERENCIADOR VERTICAL ');
  Writeln;
  Writeln(' 4) OPERADOR DE SOBEL ');
  Writeln;
  Writeln(' 5) OPERADOR LAPLACIANO ');
  Writeln;
  Writeln(' 6) DESPLEGAR IMAGEN A PROCESAR ');
  Writeln;
  Writeln(' 7) FINALIZAR ');
  Writeln;
  Writeln;
  Write(' PROCESO A REALIZAR  NUM : ');
  Readln(proceso_selec);
  IF IOResult =106 THEN
  Begin
    Writeln('ERROR EN EL FORMATO');
    Writeln;
     Proceso_selec := 0;
    Delay(1000);
  END;
  Writeln;
```

```
{**********************************************************
**                                                       **
**  SEGUN EL PROCESO ELEGIDO,  SE ASIGNA UN VALOR QUE SERA **
**  ENVIADO MEDIANTE UNA PALABRA DE CONTROL AL PUERTO DE  **
**  ESCRITURA NUMERO 1, DE ESTE MODO EL TMS25 PODRA OBTENER **
**  LA INFORMACION SOBRE EL PROCESO A EFECTUAR.           **
**                                                       **
** PROCESO_SELEC = $3F => PROCESO ELEGIDO FILTRO PASO ALTAS **
** PROCESO_SELEC = $5F => PROCESO ELEGIDO FILTRO PASO BAJAS **
** PROCESO_SELEC = $7F => PROCESO ELEGIDO FILTRO DIF VERTICAL **
** PROCESO_SELEC = $9F => PROCESO ELEGIDO OPERADOR SOBEL  **
** PROCESO_SELEC = $BF => PROCESO ELEGIDO OPERADOR LAPLACIANO **
** PROCESO_SELEC = $DF => FINALIZAR PROGRAMA DE APLICACION **
**                                                       **
**********************************************************}
   CASE Proceso_selec OF
     1 : Proceso_selec := $3F;
     2 : Proceso_selec := $5F;
     3 : Proceso_selec := $7F;
     4 : Proceso_selec := $9F;
     5 : Proceso_selec := $BF;
     6 : DESPLIEGUE(Imagen_selec);
     7 : Proceso_selec := $DF;
   END;
  UNTIL Proceso_selec IN [$3F, $5F, $7F, $9F, $BF,$DF];
END;


{**********************************************************
**                                                       **
**  PROCEDIMIENTO DE TRANSFERENCIA DE LA IMAGEN FUENTE DESDE **
**  UN ARCHIVO ALMACENADO EN LA PC A LA MEMORIA FUENTE DE LA **
**  APDI, UNA VEZ QUE SE TIENEN LAS CONDICIONES NECESARIAS **
**  PARA REALIZARLA.                                     **
**                                                       **
**********************************************************}

PROCEDURE ESC_IMAGEN(Imagen_procesar:string;var Tam_img:word);

{**********************************************************
**     LISTA DE VARIABLES EMPLEADAS EN EL PROCEDURE       **
**********************************************************}

VAR
 ArchivoIMG      { Tipo del archivo de Imagen Fuente      }
            : FILE OF BYTE;
```

```
 Valor_pixel,      { Valor del Pixel de la Imagen          }
 DirH,DirL         { Variables de Direc en escritura de IF  }
                : BYTE;

{**************************************************************
**     LISTA DE CONSTANTES EMPLEADAS EN EL PROCEDURE          **
**************************************************************}

CONST
    DIR_INICIAL = $0400; { DIR INICIAL DE ESCRITURA DE LA IF }

BEGIN

{**************************************************************
**    SE ENVIA LA PALABRA $FA PUERTO DE ESCRITURA 1 PARA      **
**    MANTENER EL TMS EN ALTA IMPEDANCIA Y PODER HABILITAR    **
**    LOS BUFFERS DE LA MEMORIA MF PARA TRANSFERIR LA IMAGEN  **
**    FUENTE A LA MEMORIRA FUENTE (MF) EN LA APDI.            **
**************************************************************}
  MEM[Dir_Pto_base:0000] := $FA;
  Assign(ArchivoIMG,Imagen_procesar);
  Reset(ArchivoIMG);
  Tam_img := DIR_INICIAL;
  WHILE not EOF(ArchivoIMG) DO
  BEGIN
    Read(ArchivoIMG,Valor_pixel);
    DirL := Tam_img;
    DirH := Tam_img shr 8;
    MEM[Dir_Pto_base:0001] := DirL;  {Dir memoria A0-A7}
    MEM[Dir_Pto_base:0002] := DirH;  {Dir memoria A8-A15}
    MEM[Dir_Pto_base:0003]:= Valor_pixel; {Valor almacenado en DIRH,DIRL}
    Inc(Tam_img);
  END;  {While}
  Close(ArchivoIMG);

{**************************************************************
*****  ENVIO DE LA SENAL DE /HOLD PARA MANTENER AL TMS25     ****
*****  EN ESTADO DE ESPERA Y SUS BUSES EN ALTA IMPEDANCIA    ****
*****  ADEMAS SE ENVIAN UNOS A LOS BITS QUE INFORMAN         ****
*****  EL PROCESO A REALIZAR EN LA IMAGEN.                   ****
**************************************************************}
  MEM[Dir_Pto_base:0000] := $FE;

END;
```

```
{**********************************************************
**                                                       **
** PROCEDIMIENTO PARA LA CREACION DE UN ARCHIVO CUYO CONTE-**
** NIDO ES LA IMAGEN RESULTADO, O BIEN EFECTUAR EL DESPLIE-**
** GUE DE LA MISMA EN EL MONITOR DE LA PC. LA FUENTE DE   **
** INFORMACION EN AMBOS CASOS ES LA MEMORIA RESULTADO (MR)**
** DE LA APDI.                                           **
**                                                       **
**********************************************************}

PROCEDURE DESPCREAMEM(Imagen_resultado:string; Cont_desp:word;
                      Desp:string; Almacen:string);

{**********************************************************
**    LISTA DE LAS VARIABLES EMPLEADAS EN EL PROCEDURE   **
**********************************************************}

VAR

 Driver,Mode,       { Regs de configuracion del monitor      }
 I,J,N,M,           { Registros contadores                   }
 Iden               { Reg aux para identificacion de imagenes }
                    : INTEGER;

 Valor_pixel,       { Valor de Pixel de la Imagen            }
 DirHDes,DirLDes    { Variables de Direc para lectura datos  }
                    : BYTE;

 ArchivoIMGRES      { Archivo de Imagen Resultado            }
                    : FILE OF BYTE;

 Apunt              { Regs apunt en lec y escr de Imagenes   }
                    : WORD;

 Archivo_ImgRes     { Nombre de la Imagen Resultado          }
                    : STRING;

 Exten              { Extension del archivo de imagen resul  }
                    : STRING[2];

 Aux                { Reg aux en nombre del archivo resultado }
                    : STRING[3];


 Infor_num          { Variable de informacion de archivos    }
                    : PATHSTR;
```

```
{**********************************************************
**    LISTA DE LAS VARIABLES EMPLEADAS EN EL PROCEDURE     **
**********************************************************}

CONST
   VAL_INICIAL = $0400;

BEGIN

{**********************************************************
**   SE ENVIA LA PALABRA $FA PUERTO DE ESCRITURA 1 PARA    **
**   MANTENER EL TMS EN ALTA IMPEDANCIA Y PODER HABILITAR  **
**   LOS BUFFERS DE LA MEMORIA MR PARA LEER LOS RESULTADOS **
**   DEL PROCESO.                                          **
**********************************************************}
  MEM[Dir_Pto_base:0000] := $FA;
  Delay(5);

  IF Desp = 'S' THEN
  BEGIN
   Driver := CGA;
   Mode := 0;
   Initgraph(Driver,Mode,'C:\TP500');
  END;

  IF Almacen = 'S' THEN
  BEGIN
   Iden := Length(Imagen_resultado) - 3;
   Exten :='00';
   I := 1;
   Archivo_ImgRes := Copy(Imagen_resultado,1,Iden) + 'p'+ Exten;
   Infor_num :=  FSearch(Archivo_ImgRes,GetEnv('PATH'));
   WHILE Infor_num <> '' DO
   BEGIN
    N := 100 + I;
    Str(N,Aux);
    Exten := Copy(Aux,2,3);
    Archivo_ImgRes :=Copy(Imagen_resultado,1,Iden) +'p'+ Exten;
    Infor_num := FSearch(Archivo_ImgRes,GetEnv('PATH'));
    inc(I);
   END;
   Assign(ArchivoIMGRES,Archivo_ImgRes);
   Rewrite(ArchivoIMGRES);
  END;
  Apunt := VAL_INICIAL;
  I := 1;
```

```
  J := 1;
  WHILE Cont_desp > $0400 DO
  BEGIN
   DirLDes := Apunt;
   DirHDes := Apunt shr 8;
   Mem[Dir_Pto_base:0001] := DirLDes; { Dir memoria A0-A7 }
   Mem[Dir_Pto_base:0002] := DirHDes; { Dir memoria A8-A15}
   Valor_pixel := Mem[Dir_Pto_base:0001];
                          { Valor leido en DIRH,DIRL }
   IF ( (Desp = 'S') and (I>1) and (I<=199) and (J>1) and (J<=199) )
                THEN PutPixel(J,I,Valor_pixel);
   IF Almacen = 'S' THEN Write(ArchivoIMGRES,Valor_pixel);
   Inc(J);
   IF J > 200 THEN
   BEGIN
    Inc(I);
    J := 1;
   END;
   Inc(Apunt);
   Dec(Cont_desp);
  END;
  Delay(2000);
  IF Almacen = 'S' THEN Close(ArchivoIMGRES);
  IF Desp = 'S' THEN
  BEGIN
   SetColor(2);
   SetTextStyle(0,0,1);
   OutTextXY(210,160,'ENTER');
   OutTextXY(210,170,'PARA');
   OutTextXY(210,180,'CONTINUAR');
   Readln;
   Closegraph;
  END;

{*************************************************************
***** ENVIO DE LA SENAL DE /HOLD PARA MANTENER AL TMS25   *****
***** EN ESTADO DE ESPERA Y SUS BUSES EN ALTA IMPEDANCIA. *****
***** ADEMAS SE ENVIAN UNOS A LOS BITS QUE INFORMAN       *****
***** EL PROCESO A REALIZAR EN LA IMAGEN.                 *****
*************************************************************}
  MEM[Dir_Pto_base:0000] := $FE;
END;
```

```
{*************************************************************
*+************************************************************
**                                                           **
**      PROGRAMA PRINCIPAL  DEL PROGRAMA DE APLICACION       **
**                                                           **
*************************************************************
*************************************************************}

BEGIN
  ClrScr;

{*************************************************************
** SE ENVIA LA SENAL DE /HOLD (PALABRA DE ESCRITTURA =$FE)   **
** AL TMS PARA QUE SUS BUSES Y SENALES DE CONTROL PERMANEZ-  **
** CAN EN ALTA IMPEDANCIA, INMEDIATAMENTE SE MANDA UN /RESET **
** (PALABRA = $FC )POR SOFTWARE PARA EL TMS, MANTENIENDO     **
** ACTIVADO EL HOLD. SE VERIFICA QUE ESTE PRESENTE EN EL     **
** TMS25 LA SENAL DE /HOLDA ACTIVO BAJO, CON ESTAS CONDICIO- **
** NES SE ESTA EN POSIBILIDAD DE TRANSFERIR INFORMACION.     **
*************************************************************}

{*************************************************************
***** ENVIO DE LA SENAL DE /HOLD PARA MANTENER AL TMS25      ****
***** EN ESTADO DE ESPERA Y SUS BUSES EN ALTA IMPEDANCIA     ****
*************************************************************}
  Mem[Dir_Pto_Base:$0000] := $FE;
  Delay(5);

{*************************************************************
***** ENVIO DE LA SENAL DE /RESET_COM PARA ACTIVAR EL        ***
***** RESET DEL TMS25 SIN QUITAR /HOLD, $FC.                 ***
*************************************************************}
  Mem[Dir_Pto_Base:$0000] := $FC;
  Delay(5);

{*************************************************************
***** ENVIO DE LA SENAL DE /SINC EN CASO DE TENER INSTALADAS ***
***** DOS O CUATRO UNIDADES DE PROCESAMIENTO.                ***
*************************************************************}

{*************************************************************
***** ENVIO DE LA SENAL DE /HOLD PARA MANTENER AL TMS25      ****
***** EN ESTADO DE ESPERA Y SUS BUSES EN ALTA IMPEDANCIA     ****
*************************************************************}
  Mem[Dir_Pto_Base:$0000] := $FE;
  Delay(5);
```

```
{**********************************************************
** RECONOCIMIENTO DE LA SENAL /HOLDA PROVENIENTE DEL TMS25      **
**********************************************************}
 REPEAT
  Holda:= Mem[Dir_Pto_Base:$0000];
  Writeln('CONDICIONES NO DADAS, no puedo leer archivo.tms');
  Delay(1000);
  Holda2 := Holda OR $FE
 UNTIL  Holda2 = $FE;

 WriteLn(' CONDICIONES DE ESCRITURA DE PROGRAMA');

 ClrScr;
 Writeln;
 Writeln;
 Writeln(' UN MOMENTO TRASNFIRIENDO PROGRAMA FUENTE ');

{**********************************************************
*** LECTURA DEL ARCHIVO.MPO PARA SU ESCRITURA EN LA MEMO-      ***
*** RIA DE PROGRAMA DEL TMS25.                                ***
**********************************************************}
 ESCRITURATMS;
 ClrScr;
 Writeln;
 Writeln('    P R O G R A M A   D E   A P L I C A C I O N');
 Writeln;
 writeln;
 Writeln(' PROCESAMIENTO DIGITAL DE IMAGENES ');
 Writeln;
 Writeln;
 Writeln;
 Cond_esc := 1;
 REPEAT
{**********************************************************
***  DETERMINACION DE LA IMAGEN A PROCESAR Y PROCESO          ***
**********************************************************}
  MENU(Proceso,Cond_esc,Archivo_Img,Cont);
  IF Proceso <> $DF THEN BEGIN
   IF Cond_esc = 1 THEN BEGIN
     ClrScr;
     Writeln;
     Writeln;
     Writeln(' TRANSFIRIENDO IMAGEN FUENTE A LA APDI ');
     ESC_IMAGEN(Archivo_Img,Cont);
     Cond_esc := 0;
     END;
```

```
    ClrScr;
    Writeln(' INICIA PROCESO EN LA IMAGEN ');

{*****************************************************************
***   SE DESHABILITA LA SENAL DE /HOLD PARA QUE EL TMS25          ***
***   PUEDA INICIAR LA EJECUCION DE SU PROGRAMA, ADEMAS DE        ***
***   ADICIONAR LA INFORMACION SOBRE EL PROCESO A REALIZAR        ***
*****************************************************************}
    Mem[Dir_Pto_Base:$0000] := Proceso;

    REPEAT
      Bandera := Mem[Dir_Pto_base:0000];
      Ban_fin := Bandera And $02;
    UNTIL (Ban_fin = 2);
    Gettime(Hora_ini,Min_ini,Seg_ini,Cseg_ini);
    Writeln;
    Writeln('ESPERANDO LA SENAL DE FINALIZACION DEL PROCESO ' );
    Writeln;
    REPEAT
      Bandera := Mem[Dir_Pto_base:0000];
      Ban_fin := Bandera AND $04
    UNTIL (Ban_fin = $04);

    GetTime(Hora_fin,Min_fin,Seg_fin,Cseg_fin);
    Seg_fin := Seg_fin - Seg_ini;
    Cseg_fin := Cseg_fin- Cseg_ini;

    Writeln('FINALIZO PROCESAMIENTO DE LA IMAGEN ');
    Writeln,
    Write('DURACION DEL PROCESO ( SEG : ',Seg_fin,'  CSEG : ',Cseg_fin);



{*****************************************************************
***   SE ENVIAN UNOS A LOS BITS QUE INFORMAN SOBRE EL PRO-        **
***   CESO A EFECTUAR EN EL PUERTO DE ESCRITURA 1                 **
*****************************************************************}
   Mem[Dir_Pto_base:0000] := $FF;
    Writeln;
    REPEAT
      Write('DESEAS DESPLIEGUE DE LA IMAGEN (S/N)? ');
      Readln(lect);
      Resp1 := UpCase(Lect[1]);
      Writeln;
    UNTIL ( (Resp1 = 'S') or (Resp1 = 'N') );
```

```
    REPEAT
      Write('DESEAS CREAR UNA ARCHIVO CON LA IMAGEN RESULTADO (S/N)? ');
      Readln(Lect);
      Resp2 := UpCase(Lect[1]);
    UNTIL ( (Resp2 = 'S') or (Resp2 = 'N') );
    IF ( (Resp1 = 'S') OR (Resp2 = 'S') ) THEN
            DESPCREAMEM(Archivo_Img,Cont,Resp1,Resp2);
  END;

{**************************************************************
***   SE ENVIAN UNOS A LOS BITS QUE INFORMAN SOBRE EL PRO-     **
***   CESO A EFECTUAR EN EL PUERTO DE ESCRITURA 1              **
**************************************************************}
  Mem[Dir_Pto_base:0000] := $FF;

  UNTIL Proceso = $DF;


{**************************************************************
***   SE ENVIALA INFORMACION DE FINALIZACION A LOS BITS QUE    **
***   INFORMAN SOBRE EL PROCESO A EFECTUAR EN EL PUERTO DE     **
***   ESCRITURA 1  Y SE MANTINE LIBERADO AL PROCESADO          **
**************************************************************}
  Mem[Dir_Pto_base:0000] := $DF;

  Delay(500);
  ClrScr;
  Writeln(' FINALIZACION DE PROGRAMA DE APLICACION ');
  Delay(1000);
END.
{$I+}
```

ANEXO D
PROGRAMA DE PROCESAMIENTO EN LENGUAJE ENSAMBLADOR

```
*****************************************************
**** PROGRAMA DE PROCESAMIENTO DE IMAGENES DIGITALES ****
**** MEDIANTE EL EMPLEO DE CONVOLUCION ****************
*
* NOMBRE : PROCONV2.ASM (Utiliza multiplicacion acumulacion)
*
*****************************************************
*
*       DEFINICION DE LOCALIDADES PARA USO DE VARIABLES Y CTES.
*
*****************************************************
*
PESCR   EQU  >1   ; PUERTO DE ESCRITURA DEL TMS32025
PLECT   EQU  >0   ; PUERTO DE LECTURA DEL TMS32025
CTEI11  EQU  >0   ; DIR. VALOR INICIAL PARA MOD_PRO_UNI1/1
CTEF11  EQU  >1   ; DIR. VALOR FINAL PARA MOD_PRO_UNI1/1
CTEI12  EQU  >20  ; DIR. VALOR INICIAL PARA MOD_PRO_UNI1/2
CTEF12  EQU  >21  ; DIR. VALOR FINAL PARA MOD_PRO_UNI1/2
CTEI22  EQU  >28  ; DIR. VALOR INICIAL PARA MOD_PRO_UNI2/2
CTEF22  EQU  >29  ; DIR. VALOR FINAL PARA MOD_PRO_UNI2/2
CTEI14  EQU  >40  ; DIR. VALOR INICIAL PARA MOD_PRO_UNI1/4
CTEF14  EQU  >41  ; DIR. VALOR FINAL PARA MOD_PRO_UNI1/4
CTEI24  EQU  >48  ; DIR. VALOR INICIAL PARA MOD_PRO_UNI2/4
CTEF24  EQU  >49  ; DIR. VALOR FINAL PARA MOD_PRO_UNI2/4
CTEI34  EQU  >50  ; DIR. VALOR INICIAL PARA MOD_PRO_UNI3/4
CTEF34  EQU  >51  ; DIR. VALOR FINAL PARA MOD_PRO_UNI3/4
CTEI44  EQU  >58  ; DIR. VALOR INICIAL PARA MOD_PRO_UNI4/4
CTEF44  EQU  >59  ; DIR. VALOR FINAL PARA MOD_PRO_UNI4/4
CONT    EQU  >5   ; VARIABLE AUXILIAR DEL CONTADOR DE DIR.
RES     EQU  >6   ; VARIABLE RESULTADO DE MULTIPLICACIONES
APUNT   EQU  >7   ; VARIABLE DE CONTEO en la misma fila
BANDI   EQU  >8   ; DIR. BANDERA DE INICIO
BANDF   EQU  >9   ; DIR. BANDERA DE FINALIZACION Y RESET
PROCE   EQU  >A   ; VECTOR DE INFORMACION DEL PROCESO
INICIO  EQU  >B   ; DIR VALOR INICIAL DEL CICLO DE PROC.
FINAL   EQU  >C   ; DIR VALOR FINAL DEL CICLO DE PROC.
*
*****************************************************
*
*       UBICACION DE LAS PLANTILLAS EN MEMORIA PROGRAMA
*
*****************************************************
*
*
        AORG >0020      ; DIRECCION PLANTILLA 1 EN MEM. PROG.
        DATA -1,-1,-1,-1,8,-1,-1,-1,-1
*
```

```
        AORG >0030      ; DIRECCION PLANTILLA 2 EN MEM. PROG.
        DATA >0E,>0E,>0E,>0E,>0E,>0E,>0E,>0E,>0E
*
        AORG >0040      ; DIRECCION PLANTILLA 3 EN MEM. PROG.
        DATA 0,1,0,0,0,0,0,-1,0
*
        AORG >0050      ; DIRECCION PLANTILLA 4 EN MEM. PROG.
        DATA 0,1,2,-1,0,1,-2,-1,0
*
        AORG >0060      ; DIRECCION PLANTILLA 5 EN MEM. PROG.
        DATA 0,-1,0,-1,4,-1,0,-1,0
*
*
*****************************************************************
*
*       INICIO DEL PROGRAMA
*
*****************************************************************
*
        AORG >0000      ; DIRECCION INICIAL AL RESET DEL TMS
        B INIT          ; DESPUES DE RESET VA A INICIO
*
INIT    AORG >0070      ; DIRECCION INICIO DEL PROGRAMA
        SSXM            ; FIJA OPERACIONES MODO SIGNADO
*
        ZAC             ; ACC <---- 0000
        SACL BANDF      ; ACC ----> DIR BANDF(0000)
        OUT BANDF,PESCR ; RESET AL PUERTO DE ESCRITURA,
*                         INDICA QUE NO EXISTE PROCESAMIENTO
*
*
*****************************************************************
*
*       INICIO DEL PROCESAMIENTO
*
*****************************************************************
*
        LDPK >6         ; PAGINA 6 (DIR >300) PARA VARIABLES
*
*****************************************************************
*
* CARGA DE LOS VALORES DE DIRECCIONES INCIALES Y FINALES
* DE PROCESAMIENTO DEPENDIENDO DEL NUMERO DE UNIDADES DE
* EN EL MODULO DE PROCESAMIENTO
*
*****************************************************************
```

```
*
        LALK >0400      ; PIXEL MAS BAJO EN MEMORIA
        SACL CTEI11
        SACL CTEI12
        SACL CTEI14
        LALK >9EB0      ; PIXEL A 1/4 DE LA IMAGEN
        SACL CTEF11
        SACL CTEF22
        SACL CTEF44
        LALK >5158      ; PIXEL A 1/2 DE LA IMAGEN
        SACL CTEF12
        SACL CTEI22
        SACL CTEF24
        SACL CTEI34
        LALK >2A48      ; PIXEL A 3/4 DE LA IMAGEN
        SACL CTEF14
        SACL CTEI24
        LALK >7868      ; PIXEL PARTE FINAL IMAGEN
        SACL CTEF34
        SACL CTEI44
*
*****************************************************
*
*       IDENTIFICACION DE LA CONFIGURACION DEL MODULO
*       DE PROCESAMIENTO, LECTURA DE PUERTO
*
*****************************************************
*
        IN PROCE,PLECT  ; LEE EN PUERTO
        LAC PROCE       ; Acc <--- PROCE
        SFR             ; CORRE EL Acc UN BIT A LA DERECHA
        ANDK >78        ; FILTRA LA LECTURA DE SWITCH 7-5
        ADLK >0300      ; Acc + VALOR DE SWITCHES (CORRIDOS)
        SACL INICIO     ; Acc ---> INICIO
        LARP AR4        ; USO DE REG. AUXILIAR AR4
        LAR AR4,INICIO  ; CARGA AR4 CON DIR. INICIO
        LAC *+          ; INCREMENTA AR4
        SACL INICIO     ; DIR. INICIO A INICIO
        LAC *           ; Acc <--- AR4
        SACL FINAL      ; DIR. AR4 A FINAL
*
*****************************************************
*
*       LECTURA DEL PROCESO A REALIZAR EN LA IMAGEN
*
*****************************************************
*
```

```
LECPRO  CNFD            ; CONFIGURA BLOQUE B0 EN MEM. DATO
        IN PROCE,PLECT  ; A TRAVES DEL PUERTO DE LECTURA
        LAC PROCE       ; Acc <--- PROCE
        ANDK >08        ; DETECTA PROCESO 1
        BZ PRO1         ; VA A REALIZAR PROCESO 1
        LAC PROCE
        ANDK >05        ; DETECTA PROCESO 2
        BZ PRO2         ; VA A REALIZAR PROCESO 2
        LAC PROCE
        ANDK >04        ; DETECTA PROCESO 3
        BZ PRO3         ; VA A REALIZAR PROCESO 3
        LAC PROCE
        ANDK >03        ; DETECTA PROCESO 4
        BZ PRO4         ; VA A REALIZAR PROCESO 4
        LAC PROCE
        ANDK >02        ; DETECTA PROCESO 5
        BZ PRO5         ; VA A REALIZAR PROCESO 5
        LAC PROCE
        ANDK >01        ; DETECTA FINALIZACION
        BZ FIN          ; SE VA A FINALIZACION
        B LECPRO        ; REGRESA A LEER PUERTO
*
*
*****************************************************
*
*       TRASLADO DE PLANTILLAS DE ACUERDO
*       AL PROCESO A EJECUTAR
*
*****************************************************
*
*
PRO1    LALK >21        ; PALABRA QUE INDICA PROCESO 1
        SACL BANDI
        OUT BANDI,PESCR ; SACA LA PALABRA EN EL PUERTO
*
        LALK >20
        LARP AR6
        LRLK AR6,>200
        RPTK >08
        TBLR *+         ; COPIA PLANTILLA 1 EN MEM DATO
        B EPRO1
*
*
PRO2    LALK >30        ; PALABRA QUE INDICA PROCESO 2
        LARP AR6
        LRLK AR6,>210
        RPTK >08
```

```
        TBLR *+         ; COPIA PLANTILLA 2 EN MEM. DATO
        B EPRO2
*
*
PRO3    LALK >61        ; PALABRA QUE INDICA PROCESO 3
        SACL BANDI
        OUT BANDI,PESCR
*
        LALK >40
        LARP AR6
        LRLK AR6,>200
        RPTK >08
        TBLR *+         ; COPIA PLANTILLA 3 EN MEM. DATO
        B EPRO1
*
*
PRO4    LALK >81        ; PALABRA QUE INDICA PROCESO 4
        SACL BANDI
        OUT BANDI,PESCR
*
        LALK >50
        LARP AR6
        LRLK AR6,>200
        RPTK >08
        TBLR *+         ; COPIA PLANTILLA 4 EN MEM. DATO
        B EPRO1
*
*
PRO5    LALK >A1        ; PALABRA QUE INDICA PROCESO 5
        SACL BANDI
        OUT BANDI,PESCR
*
        LALK >60
        LARP AR6
        LRLK AR6,>200
        RPTK >08
        TBLR *+         ; COPIA PLANTILLA 5 EN MEM. DATO
        B EPRO1


*****************************************************
*
*    EJECUCION  DEL PROCESO 1
*
*****************************************************
*
```

```
EPRO1   CNFP            ; configure bloque B0 en Mem. Prog
        LAC INICIO      ; Acc <--- DIRECCION INICIAL
        SACL CONT       ; CONT<--- (INICIO)
        ZAC             ; Acc<--- 0
        SACL APUNT      ; APUNT = 0, control de PIXEL/linea
        LARP AR0        ; UTILIZA REG. AUXILIAR AR0
        LAR AR0,CONT    ; AR0 <--- VALOR DIR CONT(INICO)
SALT11  ZAC             ; ACC <--- 00, inicia a proc. pixel
        LARP AR0        ; UTILIZA AR0
        LAR AR0,CONT    ; AR0 <= = Pixel
        MPYK >0         ; REG. P <--- (REG. T)*(0)
*
        RPTK 3          ; CICLO DE 4 MULTIPLICACIONES
        MAC >FF00,* +   ; Acc=Acc+Bi*Pixel (Bi DE PLANTILLA)
        ADRK >C4        ; SALTA A OTRO RENGLON AR0<--AR0+196
*
        MPYK >0         ; REG. P <--- 0
        RPTK 3          ; CICLO DE 4 MULTIPLICACIONES
        MAC >FF03,* +   ; Acc = Acc + K(i+3)*Pixel
        ADRK >C4        ; SIGUIENT RENGLON AR0<---AR0+196
*
        MPYK >0         ; REG. P <--- 0
        RPTK 3          ; CICLO DE 4 MULTIPLICACIONES
        MAC >FF06,* +   ; Acc = Acc + K(i+6)*Pixel
*
        SBRK >CB        ; AR0 ---> DIR. (Pixel Resultado)
        SACL *          ; Pixel Resultado-->DIR. Resultado
        SAR AR0,CONT    ; AR0 ---> CONT
*
        LAC APUNT       ; Acc <--- APUNT (contador de Col.)
        ADDK >1         ; Acc <--- Acc + 1
        SACL APUNT      ; APUNT <--- Acc
        LACK >C6        ; Acc <--- 198
        SUB APUNT       ; Acc  = 198 - APUNT
        BZ SALT12       ; Acc = 0 a SALT12 (A otro RENGLON)
*
        LARP AR0        ; Acc <> 0 continua en mismo renglon
        LAR AR0,CONT    ; AR0 <--- CONT
        SBRK >C8        ; AR0 <--- AR0 - 200 (otro pixel)
        SAR AR0,CONT    ; CONT <--- AR0
        B SALT11        ; Regresa a procesar otro Pixel de
*                       le misma linea
*
*
SALT12  LARP AR0        ;INICIA EN primer pixel OTRO RENGLON
        LAR AR0,CONT    ;AR0<---CONT
        SBRK >C6        ;AR0 <--- AR0 - 198 (nuevo renglon)
```

```
        SAR AR0,CONT    ;CONT <--- AR0
        LAC FINAL       ; valor inicial de penultimo renglon
        SUB CONT        ; Acc <--- FINAL - AUXCT
        BZ SALT13       ; SI Acc = 0 = = = >  finaliza
        ZAC             ; Acc = 0
        SACL APUNT      ; APUNT = 0, conteo en otr renglon
        B SALT11        ; Regresa a iniciar otro reglon
*
*
SALT13  LALK >01        ; Acc <--- >01
        ADD BANDI       ; Acc <--- Acc + (BANDI)
        SACL BANDF      ; Acc ----> DIR BANDF
        OUT BANDF,PESCR ; BANDERA DE FINALIZACION A PUERTO
        B NEWSAL        ; VA A SENSAR OTRO PROCESO
*
*

**********************************************************
*
*           EJECUCION DEL PROCESO 2
*
* En el PROCESO 2 existe una pequeña diferencia respecto a
* los demas, ya que se introduce un corrimiento de 7 bits a
* la derecha del resultado en cada pixel convolucionado,
* para lograr que la suma de coeficientes de la plantilla
* sume uno, para que cumpla con un FPB.
*
**********************************************************
*
EPRO2   CNFP
        LALK >41
        SACL BANDI
        OUT BANDI,PESCR ; SALIDA POR PUERTO, PROCESO 2
        LAC INICIO
        SACL CONT
        ZAC
        SACL APUNT      ; APUNT = 0  control de un renglon
        LARP AR0
        LAR AR0,CONT    ; AR0 <--- VALOR DIR CONT(INICIO)
SALT21  ZAC             ; ACC <---- 0
        LARP AR0        ; UTILIZA AR0
        LAR AR0,CONT    ; AR0 <--- Pixel
        MPYK >0
*
        RPTK 3          ; CICLO DE 4 MULTIPLICACIONES
        MAC >FF10,*+    ; Acc = Acc + Bi*Pixel
        ADRK >C4        ; OTRO RENGLON DE PIXELES
```

```
*
        MPYK >0
        RPTK 3          ; CICLO DE 4 MULTIPLICACIONES
        MAC >FF13,*+    ; Acc = Acc + B(i+3)*Pixel
        ADRK >C4        ; AR0 <---- AR0 + 196
*
        MPYK >0
        RPTK 3          ; CICLO DE 4 MULTIPLICACIONES
        MAC >FF16,*+    ; Acc = Acc + B(i+6)*Pixel
*
        SBRK >CB        ; AR0 <--- DIR (Pixel Resultado)
        RPTK >6         ; CICLO DE 7 CORRIMIENTOS
        SFR             ; CORRIMIENTO A LA DERECHA
        SACL *          ; Pixel Resultado ---> DIR.Resultado
        SAR AR0,CONT    ; CONT <--- AR0
*
        LAC APUNT
        ADDK >1
        SACL APUNT
        LACK >C6
        SUB APUNT       ; Acc  = 198 - APUNT
        BZ SALT22       ; Si Acc = 0  a SALT 22 otro renglon
*
        LARP AR0        ; Acc <> 0 continua en la mismo renglon
        LAR AR0,CONT
        SBRK >C8        ; Posiciona a CONT en otro pixel inicial
        SAR AR0,CONT
        B SALT21        ; procesar otro Pixel de la mismo renglon
*
SALT22  LARP AR0
        LAR AR0,CONT
        SBRK >C6        ; inicia nuevo renglon
        SAR AR0,CONT
        LAC FINAL       ; pixel inicial de penultima renglon
        SUB CONT        ; Acc = FINAL - CONT
        BZ SALT23       ; Si Acc = 0, finaliza
        ZAC             ; Acc = 0
        SACL APUNT      ; APUNT = 0,  conteo de otro renglon
        B SALT21        ; Regresa a iniciar otra linea.
*
SALT23  LALK >01        ; ACC <--- 01
        ADD BANDI
        SACL BANDF      ; ACC ---> DIR BANDF
        OUT BANDF,PESCR ; BANDERA DE FINALIZACION
        B NEWSAL
*
*****************************************************************
```

```
*
NEWSAL  IN PROCE,PLECT
        LAC PROCE
        ANDK >07
        SUBK >07
        BZ LECPRO
        B NEWSAL
FIN     ZAC
        SACL BANDF
        OUT BANDF,PESCR
        B FIN
```

ANEXO E

SET DE INSTRUCCIONES DEL MICROPROCESADOR TMS320C25

INSTRUCCIONES DEL TMS320C25

Para cumplir con características de un procesador de señales digitales, este debe ejecutar algoritmos matemáticos intensivos en tiempo real, por lo que requiere alta velocidad de procesamiento y capacidad de la ejecución de operaciones multiplicación/acumulación.

El desempeño de un procesador de señales es medido en términos de los requerimientos apropiados, tales como la velocidad de ejecución de instrucciones individuales, la potencia de su conjunto de instrucciones y la capacidad de entrada/salida. La velocidad está dada como el tiempo del ciclo básico de una instrucción y el número de ciclos requeridos para completar una instrucción.

El TMS320C25 cuenta con un conjunto de 133 instrucciones; 97 de ellas son ejecutadas en un ciclo simple de instrucción; otras 36 requieren ciclos adicionales para su ejecución; 21 involucran saltos, llamadas de rutinas y retornos (las cuales rompen la ejecución de la operación pipeline); otras siete instrucciones son de dos palabras (instrucciones largas inmediatas). Las ocho restantes (IN, OUT, BLKD, BLKP, TBLR, TBLW, MAC y MACD) soportan transferencia de I/O y de datos entre espacio de memoria, o son utilizadas para operaciones en paralelo en el procesador. Adicionalmente estas ocho instrucciones se convierten en un ciclo simple cuando son utilizadas en conjunto con el contador de repeticiones.

Estas instrucciones utilizan los modos direccionamiento del TMS y descritos en el anexo A, además explotan el paralelismo del procesador permitiendo complejos o intensivos cálculos que son implementados con pocas instrucciones.

A continuación se listan en orden alfabético el conjunto de instrucciones del TMS [Texas Instruments, 1988]

TMS320C2x DIGITAL SIGNAL PROCESSOR Programmer's Reference Card

TEXAS INSTRUMENTS

TI Customer Response Center (CRC) Hotline:
(800) 232-3200

TMS320 DSP Hotline:
(713) 274-2320

TMS320 DSP Bulletin Board Service:
(713) 274-2323

Instruction Symbols

Symbol	Meaning
AR	Auxiliary register
ARP	Auxiliary register pointer
B	Bit code
BR	Branch address
D	Data memory address or indirect addressing control bits (see below)
dma	Data memory address
I	Indirect/direct addressing mode 1 = indirect; 0 = direct addressing
ind	Indirect address: {*|*+|*-|*0+|*0-} for '20; {*|*+|*-|*0+|*0-|*BR0+|*BR0-} for 'C25
K	Immediate value
PA	Port address
pma	Program memory address
S	Shift count
< >	User-defined items
[ ]	Optional items

Status Register ST0 Bits

15-13	12	11	10	9	8 7 6 5 4 3 2 1 0
ARP	OV	OVM	1	INTM	DP

Status Register ST1 Bits

ARB	CNF	TC	SXM	C	1	1	HM	FSM	XF	FO	TXM	PM

NOTE: On the TMS32020, bits 5, 6, and 9 of ST1 are one's.

ARP	Auxiliary register pointer
OV	Accumulator overflow flag bit
OVM	Overflow mode bit
INTM	Interrupt mask bit
DP	Data memory page pointer
ARB	Auxiliary register pointer buffer
CNF	On-chip RAM configuration control bit
TC	Test/control flag bit
SXM	Sign-extension mode bit
C	Carry bit
HM	Hold mode bit
FSM	Frame synchronization mode bit
XF	XF pin status bit
FO	Format bit
TXM	Transmit mode bit
PM	Product shift mode bits

Instruction Format Description

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	OPCODE															
2	OPCODE								1	D						
3	OPCODE								1	D						
	BR															
4	OPCODE								I	D						
5	OPCODE				S/AR				I	D						
6	OPCODE				S/PA/B				I	D						
7	OPCODE															K
8	OPCODE														K	
9	OPCODE													K		
10	OPCODE								K							
11	OPCODE				AR				K							
12	OPCODE								K							
13	OPCODE							K								
14	OPCODE				AR			OPCODE								
	K															
15	OPCODE				S			OPCODE								
	K															

Indirect Addressing Control Bits

6	5	4	3	2 1 0
IDV	INC	DEC	NAR	next ARP

IDV	Increment/decrement value
INC	Increment flag; 1 increments auxiliary register
DEC	Decrement flag; 1 decrements auxiliary register
NAR	New auxiliary register control bit; 1 loads new ARP
ARP	Auxiliary register pointer

6	5	4	Operation	6	5	4	Operation
0	0	0	*	1	0	0	*BR0-
0	0	1	*-	1	0	1	*0+
0	1	0	*+	1	1	0	*0+
0	1	1	Not used	1	1	1	*BR0+

Instruction Set Summary

Instr	Description	Cyc†/Wd	Operand Options	Opcode	Format
ABS	Absolute value of accumulator	1/1	None	CE1B	1
ADD	Add to accumulator with shift	1/1	<dma>[,<shift>] <ind>[,<shift>[,<next ARP>]]	0000	6
ADDC	Add to accumulator with carry	1/1	<dma>; <ind>[,<next ARP>]	4300	4
ADDH	Add to high accumulator	1/1	<dma>; <ind>[,<next ARP>]	4800	4
ADDK	Add to accumulator short immediate	1/1	<constant>	CC00	10
ADDS	Add to low accumulator with no sign extension	1/1	<dma> <ind>[,<next ARP>]	4900	4
ADDT	Add to accumulator with shift specified by T register	1/1	<dma> <ind>[,<next ARP>]	4A00	4

†Cycles using full-speed, on-chip, external program memory.

Instruction Set Summary (Continued)

Instr	Description	Cyc†/Wd	Operand Options	Opcode	Format
ADLK	Add to accumulator long immediate with shift	2/2	<constant>[,<shift>]	D002	15
ADRK	Add to auxiliary register short immediate	1/1	<constant>	7E00	10
AND	AND with accumulator	1/1	<dma>; <ind>[,<next ARP>]	4E00	4
ANDK	AND immediate with accumulator with shift	2/2	<constant>[,<shift>]	D004	15
APAC	Add P register to accumulator	1/1	None	CE15	1
B	Branch unconditionally	3/2	<pma>[,<ind>[,<next ARP>]]	FF80	3
BACC	Branch to address specified by accumulator	3/1	None	CE25	1
BANZ	Branch on auxiliary register not 0	3/2	<pma>[,<ind>[,<next ARP>]]	F880	3
BBNZ	Branch if TC bit ≠ 0	3/2	<pma>[,<ind>[,<next ARP>]]	F980	3
BBZ	Branch if TC bit = 0	3/2	<pma>[,<ind>[,<next ARP>]]	F880	3
BC	Branch on carry	3/2	<pma>[,<ind>[,<next ARP>]]	5E80	3
BGEZ	Branch if accumulator ≥ 0	3/2	<pma>[,<ind>[,<next ARP>]]	F480	3
BGZ	Branch if accumulator > 0	3/2	<pma>[,<ind>[,<next ARP>]]	F180	3
BIOZ	Branch on I/O status = 0	3/2	<pma>[,<ind>[,<next ARP]]	FA80	3
BIT	Test bit	1/1	<dma>,<bit code> <ind>,<bit code>[,<next ARP>]	9000	8
BITT	Test bit specified by T register	1/1	<dma>; <ind>[,<next ARP>]	5700	4
BLEZ	Branch if accumulator ≤ 0	3/2	<pma>[,<ind>[,<next ARP>]]	F280	3
BLKD	Block move from data memory to data memory	4/2	<dma1>,<dma2> <dma1>,<ind>[,<next ARP>]	FD00	4
BLKP	Block move from program memory to data memory	4/2	<pma>,<dma> <pma>,<ind>[,<next ARP>]	FC00	4
BLZ	Branch if accumulator < 0	3/2	<pma>[,<ind>[,<next ARP>]]	F380	3
BNC	Branch on no carry	3/2	<pma>[,<ind>[,<next ARP>]]	5F80	3
BNV	Branch on no overflow	3/2	<pma>[,<ind>[,<next ARP>]]	F780	3
BNZ	Branch if accumulator ≠ 0	3/2	<pma>[,<ind>[,<next ARP>]]	F580	3
BV	Branch on overflow	3/2	<pma>[,<ind>[,<next ARP>]]	F080	3
BZ	Branch if accumulator = 0	3/2	<pma>[,<ind>[,<next ARP>]]	F680	3
CALA	Call subroutine indirect	3/1	None	CE24	1
CALL	Call subroutine	3/2	<pma>[,<ind>[,<next ARP>]]	FE80	3
CMPL	Complement accumulator	1/1	None	CE27	1
CMPR	Compare auxiliary register with AR0	1/1	<constant>	CE50	8
CNFD	Configure block as data memory	1/1	None	CE06	1
CNFP	Configure block as program memory	1/1	None	CE08	1
DINT	Disable interrupt	1/1	None	CE01	1
DMOV	Data move in data memory	1/1	<dma>; <ind>[<next ARP>]	5600	4
EINT	Enable interrupt	1/1	None	CE00	1
FORT	Format serial port registers	1/1	<constant>	CE0E	7
IDLE	Idle until interrupt	3/1	None	CE1F	1
IN	Input data from port	2/1	<dma>,<PA> <ind>,<PA>[,<next ARP>]	8000	8
LAC	Load accumulator with shift	1/1	<dma>[,<shift>] <ind>[,<shift>[,<next ARP>]]	2000	8
LACK	Load accumulator immediate	1/1	<constant>	CA00	10
LACT	Load accumulator with shift specified by T register	1/1	<dma> <ind>[,<next ARP>]	4200	4
LALK	Load accumulator long immediate with shift	2/2	<constant>[,<shift>]	D001	15
LAR	Load auxiliary register	1/1	<AR>,<dma> <AR>,<ind>[,<next ARP>]	3000	8

†Cycles using full-speed, on-chip, external program memory

Instruction Set Summary (Continued)

Instr	Description	Cyc†/Wd	Operand Options	Opcode	Format
LARK	Load auxiliary register immediate	1/1	<AR>,<constant>	C000	11
LARP	Load auxiliary register pointer	1/1	<constant>	5588	9
LDP	Load data memory page pointer	1/1	<dma>; <ind>[,<next ARP>]	5200	4
LDPK	Load data memory page pointer immediate	1/1	<constant>	C800	12
LPH	Load high P register	1/1	<dma>; <ind>[,<next ARP>]	5300	4
LRLK	Load auxiliary register long immediate	2/2	<AR>,<constant>	D000	14
LST	Load status register ST0	1/1	<dma>; <ind>[,<next ARP>]	5000	4
LST1	Load status register ST1	1/1	<dma>; <ind>[,<next ARP>]	5100	4
LT	Load T register	1/1	<dma>; <ind>[,<next ARP>]	3C00	4
LTA	Load T register and accumulate previous product	1/1	<dma> <ind>[,<next ARP>]	3D00	4
LTD	Load T register, accumulate previous product, move data	1/1	<dma> <ind>[,<next ARP>]	3F00	4
LTP	Load T register and store P register in accumulator	1/1	<dma> <ind>[,<next ARP>]	3E00	4
LTS	Load T register and subtract previous product	1/1	<dma> <ind>[,<next ARP>]	5B00	4
MAC	Multiply and accumulate	4/2	<pma>,<dma> <pma>,<ind>[,<next ARP>]	5D00	4
MACD	Multiply and accumulate with data move	4/2	<pma>,<dma> <pma>,<ind>[,<next ARP>]	5C00	4
MAR	Modify auxiliary register	1/1	<dma>; <ind>[,<next ARP>]	5500	4
MPY	Multiply (with T register, store product in P register)	1/1	<dma> <ind>[,<next ARP>]	3B00	4
MPYA	Multiply and accumulate previous product	1/1	<dma> <ind>[,<next ARP>]	3A00	4
MPYK	Multiply immediate	1/1	<constant>	A000	13
MPYS	Multiply and subtract previous product	1/1	<dma> <ind>[,<next ARP>]	3B00	4
MPYU	Multiply unsigned	1/1	<dma>; <ind>[,<next ARP>]	3F00	4
NEG	Negate accumulator	1/1	None	CE23	1
NOP	No operation	1/1	None	5500	1
NORM	Normalize contents of accumulator	1/1	<ind>	CE80	2
OR	OR with accumulator	1/1	<dma>; <ind>[,<next ARP>]	4D00	4
ORK	OR immediate with accumulator with shift	2/2	<constant>[,<shift>]	D005	15
OUT	Output data to port	1/1	<dma>,<PA> <ind>,<PA>[,<next ARP>]	E000	6
PAC	Load accumulator with P register	1/1	None	CE14	1
POP	Pop top of stack to low accumulator	1/1	None	CE1D	1
POPD	Pop top of stack to data memory	1/1	<dma>; <ind>[,<next ARP>]	7A00	4
PSHD	Push data memory value onto top of stack	1/1	<dma> <ind>[,<next ARP>]	5400	4
PUSH	Push low accumulator onto stack	1/1	None	CE1C	1
RC	Reset carry bit	1/1	None	CE30	1
RET	Return from subroutine	3/1	None	CE2B	1
RFSM	Reset serial port frame synchronization mode	1/1	None	CE36	1
RHM	Reset hold mode	1/1	None	CE38	1
ROL	Rotate accumulator left	1/1	None	CE34	1
ROR	Rotate accumulator right	1/1	None	CE35	1
ROVM	Reset overflow mode	1/1	None	CE02	1
RPT	Repeat instruction as specified by data memory value	1/1	<dma> <ind>[,<next ARP>]	4B00	4
RPTK	Repeat instruction as specified by immediate value	1/1	<constant>	CB00	10

†Cycles using full-speed, on-chip, external program memory.

Instruction Set Summary (Concluded)

Instr	Description	Cyc†/Wd	Operand Options	Opcode	Format
RSXM	Reset sign-extension mode	1/1	None	CE08	1
RTC	Reset test/control flag	1/1	None	CE32	1
RTXM	Reset serial port transmit mode	1/1	None	CE20	1
RXF	Reset external flag	1/1	None	CE0C	1
SACH	Store high accumulator with shift	1/1	<dma>[,<shift>] <ind>[,<shift>[,<next ARP>]]	5800	5
SACL	Store low accumulator	1/1	<dma> <ind>[,<shift>[,<next ARP>]]	5000	5
SAR	Store auxiliary register	1/1	<AR>,<dma> <AR>,<ind>[,<next ARP>]	7000	5
SBLK	Subtract from accumulator long immediate with shift	2/2	<constant>[,<shift>]	D003	15
SBRK	Subtract from auxiliary register short immediate	1/1	<constant>	7F00	10
SC	Set carry bit	1/1	None	CE31	1
SFL	Shift accumulator left	1/1	None	CE18	1
SFR	Shift accumulator right	1/1	None	CE19	1
SFSM	Set serial port frame synchronization mode	1/1	None	CE37	1
SHM	Set hold mode	1/1	None	CE39	1
SOVM	Set overflow mode	1/1	None	CE03	1
SPAC	Subtract P register from accumulator	1/1	None	CE16	1
SPH	Store high P register	1/1	<dma>; <ind>[,<next ARP>]	7D00	4
SPL	Store low P register	1/1	<dma>; <ind>[,<next ARP>]	7C00	4
SPM	Set P register output shift mode	1/1	<constant>	CE08	8
SQRA	Square and accumulate	1/1	<dma>; <ind>[,<next ARP>]	3900	4
SQRS	Square and subtract previous product	1/1	<dma>; <ind>[,<next ARP>]	5A00	4
SST	Store status register ST0	1/1	<dma>; <ind>[,<next ARP>]	7800	4
SST1	Store status register ST1	1/1	<dma>; <ind>[,<next ARP>]	7900	4
SSXM	Set sign-extension mode	1/1	None	CE07	1
STC	Set test/control flag	1/1	None	CE33	1
STXM	Set serial port transmit mode	1/1	None	CE21	1
SUB	Subtract from accumulator with shift	1/1	<dma>[,<shift>] <ind>[,<shift>[,<next ARP>]]	1000	6
SUBB	Subtract from accumulator with borrow	1/1	<dma> <ind>[,<next ARP>]	4F00	4
SUBC	Conditional subtract	1/1	<dma>; <ind>[,<next ARP>]	4700	4
SUBH	Subtract from high accumulator	1/1	<dma>; <ind>[,<next ARP>]	4400	4
SUBK	Subtract from accumulator short immediate	1/1	<constant>	CD00	10
SUBS	Subtract from low accumulator with no sign extension	1/1	<dma> <ind>[,<next ARP>]	4500	4
SUBT	Subtract from accumulator with shift specified by T register	1/1	<dma> <ind>[,<next ARP>]	4600	4
SXF	Set external flag	1/1	None	5E0D	1
TBLR	Table read	4/1	<dma>; <ind>[,<next ARP>]	5800	4
TBLW	Table write	3/1	<dma>; <ind>[,<next ARP>]	5900	4
TRAP	Software interrupt	3/1	None	CE1E	1
XOR	Exclusive-OR with accumulator	1/1	<dma>; <ind>[,<next ARP>]	4C00	4
XORK	Exclusive-OR immediate with accumulator with shift	2/2	<constant>[,<shift>]	D006	15
ZAC	Zero accumulator	1/1	None	CA00	1
ZALH	Zero low accumulator and load high accumulator	1/1	<dma> <ind>[,<next ARP>]	4000	4
ZALR	Zero low accumulator and load high accumulator with rounding	1/1	<dma> <ind>[,<next ARP>]	7B00	4
ZALS	Zero accumulator, load low accumulator with no sign extension	1/1	<dma> <ind>[,<next ARP>]	4100	4

†Cycles using full-speed, on-chip, external program memory.

ANEXO F
ABREVIATURAS UTILIZADAS

ACC : Acumulador.
A/D : Análogo/Digital.
ALU : Unidad Aritmético Lógica.
ANSI: American National Standar.
APDI: Arquitectura de Procesamiento Digital de Imágenes
AR : Registros Auxiliares.
ARP : Apuntador de Registros Auxiliares.
ARAU: Unidad Aritmética de Registros Auxiliares.
ASCII: American Standard Code for Information Interchange.

CCIR: Sistema de Televisión Europeo Monocromático.
CI: Circuito Integrado.
CF: Código Fuente.
CO: Código de operación.
CP: Contador de programa.
CRT: Tubo de Rayos Catódicos.

DP : Registro apuntador de Página.
DSP: Procesamiento de Señales Digitales.
DMA: Acceso Directo de Memoria.
dma: direccionamiento de memoria de datos (desde el TMS)

E/S: Entrada/Salida.

FPA: Filtro Paso Altas.
FPB: Filtro Paso Bajas.

GND: 0 volts o Tierra.

hex (>) : número en código hexadecimal.
HDTV: Televisión de alta resolución.
IIMAS: Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas.
IF: Imagen Fuente.
IP: Imagen Procesada.

LCA: Celdas de Arreglos Lógicos.
L/E: Lectura/Escritura.
LSB: Bit menos significativo.

LSI: Gran Escala de Integración.
LUT: Look-Up Table.

MF : Memoria Fuente.
MIMD: Múltiple Instrucción, Múltiple Dato.
MP : Memoria Programa.
MR : Memoria Resultado.
ms : mili segundo.
MSB: Bit más significativo.
MFT: Función de Transferencia Modulada.

nm : nano metro $= 10^{-9}$ metros.
ns : nano segundo $= 10^{-9}$ segundos.
NTSC: Sistema de Video estándar para Imágenes a Colores.
PC : Computadora Personal.
PDI: Procesamiento Digital de Imágenes.
PE : Proceso Elemental.
PEs: Procesos Elementales.
PF : Programa Fuente.
PMS: Notación que viene de Processor, Memory and Switch.
pma: direccionamiento de memoria programa (desde el TMS)

s : segundo (unidad de tiempo)
SPDI: Sistema Procesamiento Digital de Imágenes.
SIMD: Simple Instrucción, Múltiple Dato.
SISD: Simple Instrucción, Simple Dato
SVI: Sistema de Visión Digital.
SVH: Sistema de Visión Humano.

TMS: microprocesador TMS32025.
Tpi1: tiempo de procesamiento de una imagen.
TP5: Turbo Pascal 5.

UNAM: Universidad Nacional Autónoma de México.
us : microsegundo.

VLSI: Muy alta escala de integración.
*** :** Registro auxiliar en uso (en lenguaje ensamblador). Símbolo de la operación de convolución.