2 ej.
4

Universidad Nacional Autónoma de México

Facultad de Ingeniería

PROCESAMIENTO EN PARALELO

TESIS DE LICENCIATURA

Que para obtener el título de:

INGENIERO EN COMPUTACION

P r e s e n t a n :

Jorge Christen Gracia

Sergio Rajsbaum Gorodezky

Director de Tesis: Dr. Andrés Buzo

México, D. F. 1986

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Indice

Prefacio

La motivacion original de esta tesis fue un profundo interes tanto por el procesamiento paralelo como por el procesamiento digital de señales. En un principio era un interés teorico, intelectual; que sin embargo, durante un periodo de trabajo con el Dr. Andres Buzo en la DEPFI (UNAM), se materializó en un problema concreto: existen algoritmos para el procesamiento de voz cuyos tiempos de ejecucion los hacen imposibles de ejecutar en los microprocesadores comerciales actuales.

Objetivo.

El objetivo de la tesis es realizar un estudio del material existente en el campo del procesamiento paralelo que
(i) sea una contribucion a la literatura de computacion en castellano, y
(ii) provea el marco de conocimientos necesarios para el diseño de una arquitectura paralela para el procesamiento digital de señales que sea posible de implementar en México.

La tesis se divide en dos partes. En la primera (capitulo I, II, III, IV y V) se hace una revisión bibliográfica del procesamiento en paralelo. En la segunda parte (capitulo VI) se describe el diseño de la arquitectura con ejemplos de programas y una simulacion en software.

CAPITULO I

Introducción

1 - Introducción

I- Procesamiento Paralelo.

Definición.

El Procesamiento Paralelo es una forma eficiente de procesamiento de información que hace énfasis en la explotación de eventos concurrentes del proceso de cómputo. **Concurrencia implica paralelismo, simultaneidad y pipelining.** Los eventos paralelos ocurren en múltiples unidades durante el mismo intervalo de tiempo; Los eventos simultáneos ocurren en el mismo instante; y eventos pipelining ocurren durante intervalos sobrepuestos de tiempo. El procesamiento paralelo está en contraste con el procesamiento secuencial. Es un medio eficiente de mejorar el desempeño de un sistyema mediante actividades concurrentes dentro de la computadora para la solución de un problema.

Cuando hablamos de procesamiento paralelo, no nos referimos a sistemas donde se logra concurrencia mediante software como son los sistemas con multiprogramación, a pesar de que muchos principios usados en estos sistemas también lo son en el procesamiento paralelo. Tampoco nos referimos a redes de computadoras, ya que normalmente son sistemas formados por computadoras independientes que no cooperan para resolver una misma tarea. Sin embargo, existen sistemas distribuidos que sí cumplen con nuestra definicion de procesamiento paralelo; no los consideramos debido a que caen fuera del alcance de este trabajo. Tratamos en general con principios y técnicas básicas así como con aplicaciones de las distintas formas que se conocen de implementar procesamiento paralelo.

II- Estructuras para Procesamiento Paralelo.

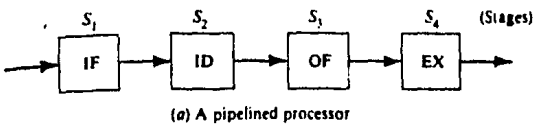
Las **computadoras paralelas** son aquellos sistemas que enfatizan, de alguna manera, el procesamiento paralelo. Dividimos a las computadoras paralelas en:

+ Pipeline.
+ Procesadores de Vectores.
+ Procesadores de Arreglos.
+ Procesadores Asociativos.
+ Multiprocesadores.
+ Computadoras Accionadas por Datos
+ Computadoras Accionadas por Demanda.

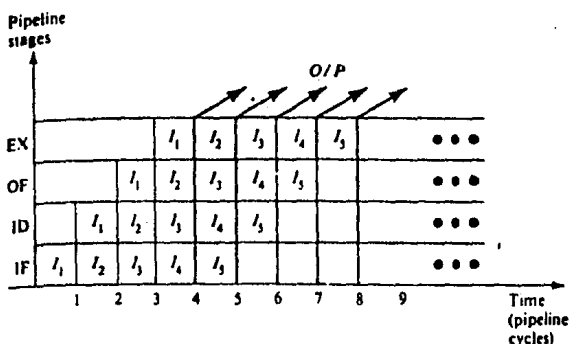
Estas siete categorias de computadoras paralelas no son mutuamente excluventes. Por ejemplo, existen computadoras que utilizan técnicas pipeline, que cuentan con varios procesadores y que además tienen una unidad de procesamiento de arreglos.

Pipeline.

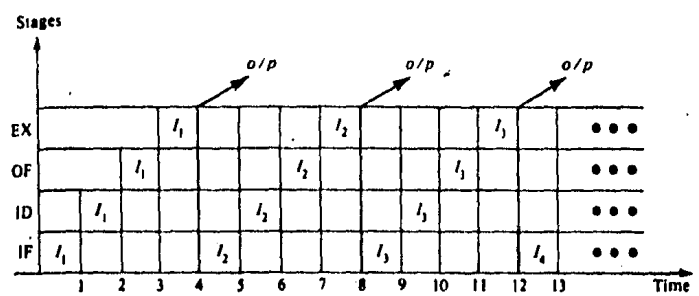
El concepto de pipeline se refiere a la siguiente manera de resolver un problema P. Se divide a P en N subproblemas P_i tales que al resolver cada uno de ellos uno tras de otro, en orden, es equivalente a resolver P. Si es posible encontar el conjunto de los P_i para un problema dado, entonces utilizando N procesadores, cada uno trabajando sobre uno de los subproblemas de tal forma que la entrada a un subproblema P_i es la salida del subproblema anterior P_j $(j<i)$ y su salida va a dar al siguiente subproblema P_k $(k>i)$, la entrada al primer subproblema es la entrada de datos original y la salida del último subproblema es la respuesta del problema, entonces, en teoría, se puede resolver el problema N veces más rápido que si se usara un solo procesador. Este concepto es análogo al utilizado en las líneas de ensamble de algunas fábricas. El concepto de pipeline se puede implementar a nivel de instruccion, de intrainstrucción o de procesador. En la figura 1.1.1 se ilustra el caso de pipeline de instrucciones.



(a) A pipelined processor



(b) Space-time diagram for a pipelined processor



(c) Space-time diagram for a nonpipelined processor

Figura 1.1.1 Pipeline de Instrucciones.[1]

Procesadores de vectores.

El surgimiento de computadoras dedicadas al procesamiento de vectores comenzó con la introducción de dos computadoras conocidas como la CDC-Star y la TI-ASC. La primera generación de los procesadores de vestores fue marcada por estas dos máquinas así como por la Iliac-IV en los años 60's; la segunda generación comenzó con procesadores como la Cray-1, la Fujitsu VP-200 y las series Cyber-200.

La mayoría de los procesadores de vectores tienen una estructura pipeline debido a caraterísticas de las instrucciones para vectores como es la invocación repetida de idénticos procesos, que son eficientemente implementadas con esta filosofía.

Procesadores de Arreglos.

Un procesador de arreglos (array processor) es un sistema que contiene múltiples ALU's llamadas elementos procesadores (PE) operando en paralelo bajo una sola unidad de control que los sincroniza para que ejecuten la misma función simultáneamente. El sistema debe proveer un mecanismo de comunicación entre los PE's. La estructura típica de un procesador de arreglos se puede ver en la figura 1.1.2.

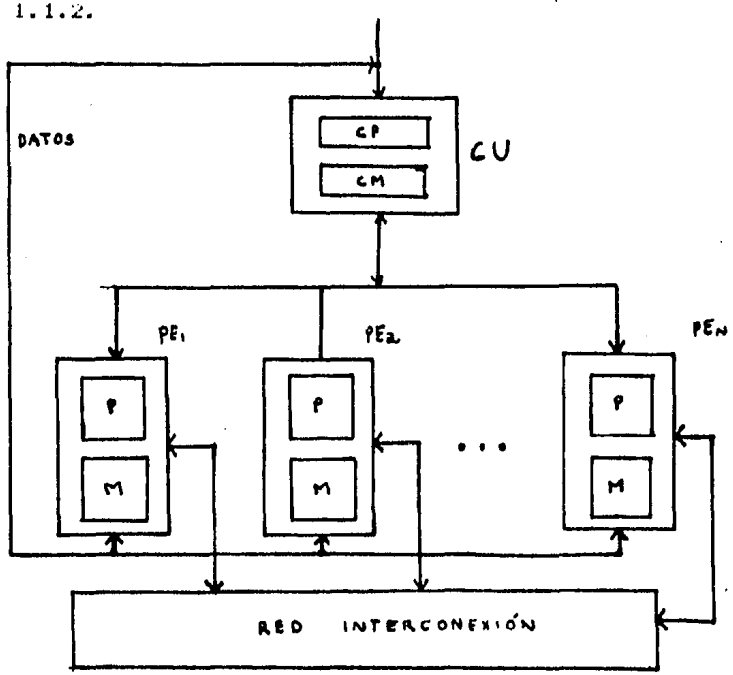


Figura 1.1.2. Estructura funcional de un procesador de arreglos.[1]

Estos sistemas son muy útiles precisamente para procesar matrices, debido a que ejecutan una misma instrucción sobre varios datos a la vez logrando así lo que se conoce como "paralelismo espacial" a diferencia del "paralelismo temporal" que explota la filosofía pipeline.

Procesadores Asociativos.

Un procesador asociativo, en general, es una máquina que tiene las siguientes dos propiedades:

1. Los datos de la memoria pueden ser accesados a partir de su contenido en lugar de mediante su dirección.
2. Se pueden realizar transformaciones de datos, tanto aritméticas como lógicas, sobre muchos datos con una sola instrucción.

Debido a estas características de procesamiento paralelo, los procesadores asociativos pueden resolver muy eficientemente problemas de procesamiento de información como son los de accesos y búsquedas en bases de datos que cambian rápidamente, realización de operaciones aritméticas y lógicas en grandes conjuntos de datos, procesamiento de señales de radar, entre otros. Se han construido dos procesadores asociativos: El Goodyear Aerospace STARAN y el Parallel Element Processing Ensamble (PEPE).

Multiprocesadores.

La necesidad de lograr velocidad, confiabilidad, flexibilidad y disponibilidad ha motivado el desarrollo de los sistemas de multiprocesadores. Estos sistemas se caracterizan por tener dos o mas procesadores de aproximadamente la misma capacidad y que pueden cooperar y comunicarse a diatintos niveles con el objetivo de resolver una misma tarea. La comunicación se realiza mediante memorias compartidas o mediante una red de interrupciones. Dependiendo del grado de interaccion entre los procesadores pueden ser: débilmente acoplados (loosely coupled) o fuertemente acoplados (tightly coupled). Algunos ejemplos de multiprocesadores son el Cm*, Cyber-170, Honeywell 60/66 y PDP-10.

Computadoras accionadas por Datos y por Demanda.

Las computadoras convencionales son "accionadas por control" (control driven) debido a que las instrucciones se ejecutan secuencialmente de la forma indicada por un registro (PC). Para tratar de explotar el máximo paralelismo de un programa se han propuesto dos nuevas filosofías diferentes de la von Neumann: Flujo de datos o acción por datos (data flow o data driven) y reducción o accionadas por demanda (reduction o demand driven). La computadora M.I.T Data- Flow pertenece al primer grupo en tanto que la GMD Reduction Machine pertenece al segundo.

El concepto básico de flujo de datos es que una instrucción se ejecute en el momento en el que tiene todos sus operandos disponibles. En acción por demanda el concepto es el opuesto. Una instruccion provoca la ejecución de las instrucciones que le proporcionaran los datos que ella necesite. Por lo tanto, en ninguna de las dos filosofías se requiere de un PC; la secuencia de ejecución esta determinada por las dependencias entre los

datos. Se piensa que arquitecturas distintas de las de von Neumann como estas dos son las que van a predominar en las computadoras de la quinta generacion.

III- Desempeño de Computadoras Paralelas.

En aplicaciones científicas existen problemas que ni siquiera los procesadores secuenciales más veloces pueden resolver en tiempo razonable. La idea es que, al poner a trabajar mas procesadores en paralelo, el tiempo para resolverlo se reduzca en forma proporcional al número de procesadores iguales P. El factor de speedup S es una medida comun del desempeño (performance) de sistemas paralelos y se define como el tiempo raquerido para completar el problema utilizando P procesadores dividido entre el tiempo requerido para completar el problema usnado uno de los procesadores.

$$S = T_p / T_1$$

En general $1 \leq S \leq P$. En los promeros días del procesamiento paralelo Minsky hizo la siguiente conjetura pesimista:

$$S = \log P$$

Aunque en ocaciones se observa este comportamiento, muchas veces se ha logrado obtener mejores resultados.

Una estimación más optimista se puede obtener como una especie de promedio entre los distintos modos de operar de un multiprocesadoe quedando una curva acotadaq por:

$$S \leq N / Ln\ N$$

que se puede obtener de la siguiente manera.

Consideremos un problema que se puede resolver por un solo procesador en tiempo $T_1=1$. Sea F_i la probabilidad de asignar el mismo problema a i procesadores trabajando con la misma carga $d_i = 1/i$ por procesador. Si la probabilidad de cada uno de los modos i, $i=1,2,\ldots,N$ es igual a $F_i = 1/N$, entonces el tiempo promedio para resolver el problema usando N procesadores es:

$$T_N = \sum_{(i=1,N)} F_i \times d_i = \frac{\sum_{(i=1,N)} 1/i}{N}$$

y

$$S = T_1 / T_N = \frac{N}{\sum_{(i=1,N)} 1/i} \leq N / Ln\ N$$

Debido a estos resultados es comun encontrar multiprocesadores con dos o cuatro procesadores únicamente, aunque los hay con 16 y aun mas. De hecho, la MPP que está siendo construida actualmete construida en la NASA cuenta con 16384 procesadores.[1] En la figura 1.1.3 estan graficadas varias curvas de speedup.

Existen otras conjeturas como la de Grosh que supone que la ley de Amdhal (N / Ln N) es optimista para N grande (mayor que 10^3): Kuck y sus colegas obtienen 30% de N mediante codigos paralelizados en forma semiautomatica en FORTRAN, y Fox y Seitz han logrado aplicaciones con 94% de N para N=32, 64 y 128.[13]

Por otro lado, está la segunda ley de Amdhal que considera el problema de que muchas veces hay partes del código que se deben ejecutar en forma secuencial necesariamente. Esta ley, sugerida en 1967, dice que si una computadora tiene dos modos de operación, el más lento dominará el desempeño aún en el caso de que el modo más veloz sea infinitamente rápido.[13]

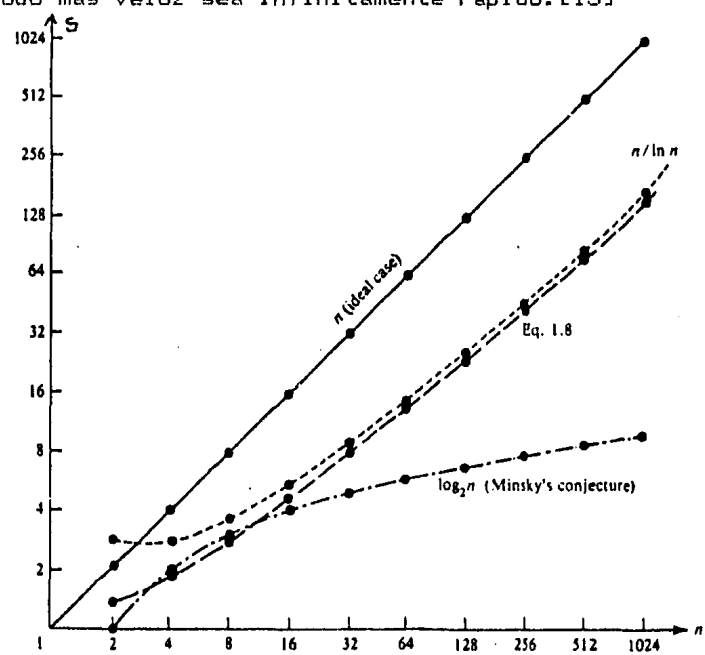


Figura 1.1.3. Varias estimaciones del speedup de un multiprocesador con N procesadores.[1]

Para medir el desempeño real de una computadora, no se pueden ignorar factores como costo y facilidad de programación; además, el desempeño de una computadora varía de acuerdo al tamaño y al tipo de problema que esté resolviendo.

Si comparamos los diversos sistemas de cómputo podemos ver que los uniprocesadores con pipeline son los que dominan el mercado tanto en aplicaciones científicas como en comerciales. Los procesadores de arreglos, la mayoría de las veces son

diseñados específicamente para las necesidades de un usuario, y para estas aplicaciones son muy eficientes. Sin embargo, programar en un procesador de arreglos es muy difícil debido a su rígida arquitectura. Los multiprocesadores son más flexibles en aplicaciones de proposito general. Los multiprocesadores con pipeline representan el estado del arte en el procesamiento paralelo. El factor costo mas que el de velocidad es a veces el que impulsa el desarrollo de un sistema paralelo. El bajo costo por MIPS de los microprocesadores hace atractivo el diseño de sistemas basados en multiples procesadores pequeños en lugar de utilizar un solo procesador de muy alta capacidad.[1],[21],[22].

2 - Esquemas de Clasificación

Los esquemas de clasificacion son procesos de abstraccion de informacion que el hombre inventa para tratar de comprender una área de conocimiento compleja. En las ciencias biologicas, por ejemplo, se clasifican a los seres vivos en animales y vegetales, y luego, a partir de estas dos categorias se van haciendo otras más pequeñas, de tal forma que el estudio de la biologia se apoya fuertemente en este esquema de clasificación. La intencion de esta seccion es: clasificar a los distintos sistemas de procesamiento paralelo para proporcionar un panorama general de esta fascinante área de la computacion, así como proveer un marco básico de trabajo sobre el cual desarrollar el resto de esta tesis.

Sin embargo, debido a lo nuevo de este campo, y a lo rápido que está evolucionando, no existe un esquema de clasificacion aceptado universalmente, que represente a todos los sistemas. A continuacion se exponen varios esquemas, esperando que esto contribuya a dar una idea completa de lo que es el procesamiento paralelo.

Esquema de Murtha y Beadles.

Un intento de clasificación hecho en 1964 por Murtha y Beadles propuso tres amplias clases de procesadores paralelos:

1) Redes de computadoras de proposito general.
2) Redes de computadoras de proposito especifico caracterizadas por paralelismo global.
3) Computadoras no globales donde los módulos son semindependientes (es decir paralelas localmente).

La primera categoria a su vez fue dividida en:

1) Redes paralelas con un control central comun.
2) Redes paralelas con gran cantidad de procesadores identicos que puedan ejecutar instrucciones independientemente.

La segunda categoria fue dividida en:

1) Procesadores de patrones.
2) Procesadores asociativos.

En realidad, la categoria "computadoras no globales" fue usada como un saco donde cupiera todo lo que no se pudo colocar en las otras dos categorias. De todos modos fue un primer intento por diferenciar a las redes de computadoras de los otros sistemas de procesamiento paralelo.

Unidades en paralelo.

Otro enfoque es considerando las cosas que pueden estar en paralelo:

1) Unidades de control.
2) Unidades de procesamiento.
3) Flujos de datos.

En un sistema puede haber varias unidades de control funcionando simultáneamente. De esta manera habría varios flujos de instrucciones que pueden estar operando sobre partes de un solo problema o sobre distintos problemas.

Pueden existir varias unidades de procesamiento, ya sea iguales o diferentes, operando en paralelo sobre un solo flujo de datos o sobre varios flujos de datos diferentes.

Finalmente, puede haber varios flujos de datos sobre los que se realiza una misma operacion (como en las máquinas de arreglos), o sobre los cuales se realizan distintas operaciones simultáneamente.

Una clasificación forzada.

Hobbs y Thesis en su escrito "Survey of Parallel Processor Approaches and Techniques" [2] hacen un intento, más bien desesperado de lograr una clasificacion correcta del procesamiento paralelo. La idea es agrupar a los distintos sistemas que se han construido o propuesto, y luego asignar un título apropiado a cada grupo. El resultado es el siguiente.

1) Multicomputadoras y multiprocesadores.
2) Procesadores asociativos.
3) Procesadores de arreglos.
4) Organizaciones de funciones.

En la tabla 1.1.1 se muestra la categoría de varios sistemas bajo esta clasificacion.

El tipo I incluye a todas las multicomputadoras y multiprocesadores excepto a las máquinas altamente paralelas (es decir, las de más de diez procesadores y cuando los procesadores están integrados de tal forma que pueden trabajar sobre el mismo algoritmo). El tipo IV se refiere a los sistemas que cuentan con un número de módulos funcionales para permitir la ejecución de distintos tipos de operaciones proceder concurrentemente sobre datos diferentes, ya sea dentro del mismo programa o dentro de programas distintos.[2]

Type I *Multicomputers and Multiprocessors*	Type II *Associative Processors*	Type III *Parallel Network of Array Processors*	Type IV *Functional Machines*
IBM 9020[3]	Rosin-Associative Cryogenic Computer[10]	Holland Machine[17]	IBM 360/91[30]
RW-400[4]	Librascope's Associative Parallel Processor[11]	Comfort's-Modified Holland Machine[18]	CDC 6600[31]
Pilot[5]	Lee-Intercommunicating Cells[12]	Gonzalez-Multilayer Iterative Circuit Computer ICC[19]	CDC 7600[32]
Intrinsic Multiprocessor IMP[6]	Davies–An Associative Processor[13]	Squire-Multidimensional Machine[20]	RCA LIMAC[33]
Burroughs D825[7]	Distributed Logic Memory DLM[14]	Vector Arithmetic Multiprocessor VAMP[21]	Bull Gamma 60[34]
Multics[8]	Hughes Associative Storing Processor[15]	Unger's Machine[22]	
Hughes 4118[9]	Goodyear's Associative Processor[16]	Solomon Machine[23]	
		Knapp-Iterative Array Computer[24]	
		Associative Logic for Highly Parallel Systems ALPS[25]	
		Conway's Machine[26]	
		ILLIAC IV[27]	
		Autonetic's Distributed Processor[28]	
		Litton's Block Oriented Computer, BOC[29]	

Tabla 1.2.1. Categorizacion de tipos de procesadores paralelos.[2]

Multiplicidad de flujos de instrucciones y datos.

Probablemente la clasificacion de arquitecturas más conocida es la concebida por Michael J. Flynn en 1966. En general, las computadoras digitales pueden ser clasificadas en cuatro categorias de acuerdo a la multiplicidad de flujos de instrucciones y de datos. El termino flujo se utilira aqui para denotar a una secuencia de entidades (datos o instrucciones) sobre las que opera un solo procesador. Un **flujo de instrucciones** es una secuencia de instrucciones ejecutadas por un procesador. Un **flujo de datos** es una secuencia de datos, incluyendo entradas, resultados y resultados parciales, utilizados por el flujo de instrucciones.

Se caracterizan a los organizaciones de computadoras de acuerdo a la capacidad del hardware para manejar los flujos de instrucciones y de datos. A continuacion se tratan las cuatro categorias propuestas por Flynn:

1) Flujo de instrucciones unico, flujo de datos unico (SISD).
2) Flujo de instrucciones unico, flujo de datos multiple (SIMD).
3) Flujo de instrucciones multiple, flujo de datos unico (MISD).
4) Flujo de instrucciones multiple, flujo de datos multiple (MIMD).

Estas organizaciones se ilustran en la figura 1.2.1. Conceptualmente solamente son necesarios tres tipos de componentes. El módulo de memoria (MM) reciben tanto las instrucciones como los datos. La unidad de control (CU) toma las instrucciones de la memoria, las decodifica y las manda a las unidades de procesamiento (PU) para ser ejecutadas. Los flujos de datos fluyen en ambas direcciones entre memoria y procesadores. Cada unidad de control genera un flujo de instrucciones. En estos diagramas de bloques simplificados no se muestran unidades de E/S.

Máquinas SISD. Esta organizacion es la típica de la mayoria de las computadoras actuales. Las instrucciones se ejecutan secuencialmente aunque pueden estar traslapadas en sus fases de ejecucion (pipelining). Una computadora SISD puede contener más de una unidad funcional, pero todas deben estar bajo la supervision de una sola unidad de control.

Máquinas SIMD. Estas computadoras son las llamadas procesadores de arreglos. Consisten de multiples unidades de procesamiento supervisadas por una sola unidad de control. Todos los PE's reciben la misma instruccion transmitida por la unidad de control pero cada una opera sobre otro conjunto de datos, correspondiente a un flujo de datos distinto. Las maquinas SIMD se dividen a su vez en segmentadas por palabra (word-slice) y en segmentadas por bit (bit-slice).

Máquinas MISD. Esta estructura, que se le conoce como pipeline de procesador (ver capitulo de pipeline), es la que ha recibido menor atencion de las cuatro. Existen n PE's cada uno recibiendo diferntes instrucciones, pero todos operando sobre el mismo flujo de datos y sus derivados. La salida de un procesador se convierte en la entrada del siguiente. No existe ninguna implementacion actualmente de este tipo.

Máquinas MIMD. La mayoria de los multiprocesadores y multicomputadoras pertenecen a esta categoria. Se le llama a una computadora **MIMD intrínseca** si existe interaccion entre los n procesadores de tal forma que los flujos de memoria de los distintos PE's comparten el mismo espacio de datos. Si los n flujos de datos provienen de espacios distintos, entonces tenemos lo que seria una **SISD multiple** (MISD), que no es otra cosa que un conjunto de n sistemas uniprocesadores SISD. Una computadora MIMD intrínseca puede ser **fuertemente acoplada** (tightly coupled) o **débilmente acoplada** (loosely coupled) dependiendo del grado de interaccion que haya entre los procesadores.

En la tabla 1.2.2 se listan algunas computadoras de cada una de estas categorias.

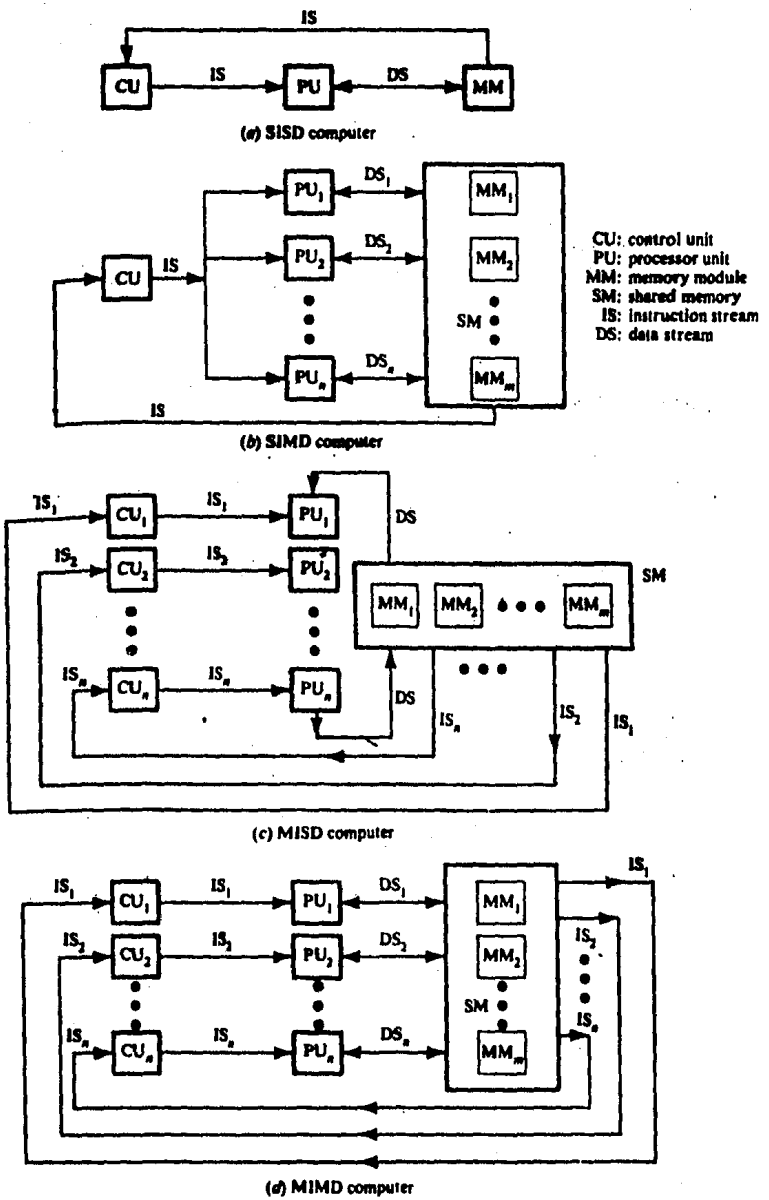


Figura 1.2.1. Clasificación de Flynn.[1]

Computer class	Computer system models (chapters where the system is quoted or described)
SISD (uses one functional unit)	IBM 701 (1); IBM 1620 (1); IBM 7090 (1); PDP VAX11/780 (1).
SISD (with multiple functional units)	IBM 360/91 (3); IBM 370/168UP (1); CDC 6600 (1); CDC Star-100 (4); TI-ASC (4); FPS AP-120B (4); FPS-164 (4); IBM 3838 (4); Cray-1 (4); CDC Cyber-205 (4); Fujitsu VP-200 (4); CDC-NASF (4); Fujitsu FACOM-230/75 (4).
SIMD (word-slice processing)	Illiac-IV (6); PEPE (1); BSP (6)
SIMD (bit-slice processing)	STARAN (1); MPP (6); DAP (1).
MIMD (loosely coupled)	IBM 370/168 MP (9); Univac 1100/80 (9); Tandem/16 (9); IBM 3081/3084 (9); C.m* (9)
MIMD (tightly coupled)	Burroughs D-825 (9); C.mmp (9); Cray-2 (9). S-1 (9); Cray-X MP (9); Denelcor HEP (9)

Tabla 1.2.2. Algunos sistemas bajo la clasificacion de Flynn.[1]

Procesamiento Paralelo vs. Serial.

Tsen-yun Feng ha sugerido utilizar al grado de paralelismo para clasificar diferentes arquitecturas de computadoras. El **máximo grado de paralelismo** P se define como el máximo número de digitos binarios que pueden ser procesados en una unidad de tiempo. Sea P_i el número de bits que pueden ser procesados en el i-ésimo ciclo de procesador. Si consideramos T ciclos, el **grado de paralelismo promedio** P_a seria

$$P_a = \frac{\text{Sum}(i=1,T)(P_i)}{T}$$

Como en general, $P_i <= P$, entonces definimos a la **razón de utilización** u de un sistema en T ciclos, como:

$$u = \frac{P_a}{P}$$

La figura 1.2.2 muestra la clasificacion de las computadoras de acuerdo a sus grados de paralelismo. En el eje horizontal se indica la longitud de palabra n mientras que en el vertical se indica la longitud de segmento por bit (bit-slice) m. Por ejemplo, la TI-ASC tiene una longitud de palabra de 64 bits y cuatro pipelines aritmeticos. Cada pipeline es de ocho etapas. Por lo tanto, cada segmento por bit es de 8x4=32 bits en los cuatro pipelines. La TI-ASC se representa entonces como (64,32). El maximo grado de paralelismo de una computadora C es:

$$P(C) = n \times m$$

En la figura 1.2.2, P(C) es igual al área del rectangulo definido por los enteros m y n.

Se pueden deducir cuatro formas de procesamiento a partir de este diagrama:

1) Palabra-serial y bit-serial (WSBS).
2) Palabra-paralelo y bit-serial (WPBS).
3) Palabra-serial y bit-paralelo (WSBP).
4) Palabra-paralelo y bit-paralelo (WPBP).

Al tipo WSBS (n=m=1) se le ha llamado procesamiento serial por bit (bit-serial) ya que se procesa un solo bit a la vez. Esto se hizo únicamente en las computadoras de la primera generación.

Al tipo WPBS (n=1, m>1) se le llama procesamiento bit (bit-slice) porque se procesan segmentos de m bits a la vez.

El tipo WSBP (n>1,m=1) es el que utiliza la mayoria de las computadoras, se le conoce como procesamiento por segmentos de palabras (word-slice) debido a que se procesa una palabra de n bits a la vez.

Finalmente, el WPBP (n>1,m>1) se denomina procesamiento paralelo total. En este caso se procesa al mismo tiempo un arreglo de n x m bits. En la tabla 1.2.3 se pueden observar algunas computadoras bajo esta clasificacion.

Paralelismo vs. Pipeplining.

La clasificacion propuesta por Wolfgang Händler se basa en la identificacion de tres niveles de subsistemas:

1) Unidad de control (PCU).
2) Unidad aritmetica-logica (ALU).
3) Circuitos nivel-bit (BLC).

Las funciones del PCU y del ALU son claras. El BLC se refiere al circuito lógico combinacional necesario para realizar operaciones de un bit en el ALU.

Un sistema de computo C puede ser caracterizado por un triple que contiene seis elementos diferentes:

$$T(C) = \langle K \times K', D \times D', W \times W' \rangle$$

donde K = número de PCU's en el sistema.
K = número de PCU's que pueden operar en cascada (macro-pipelining).
D = número de ALU's bajo el control de un PCU.
D = número de ALU's que pueden operar en cascada. (pipeline chaining).
W = longitud de la palabra de un ALU o PE.
W = numero de etapas de un pipeline dentro del ALU o PE.

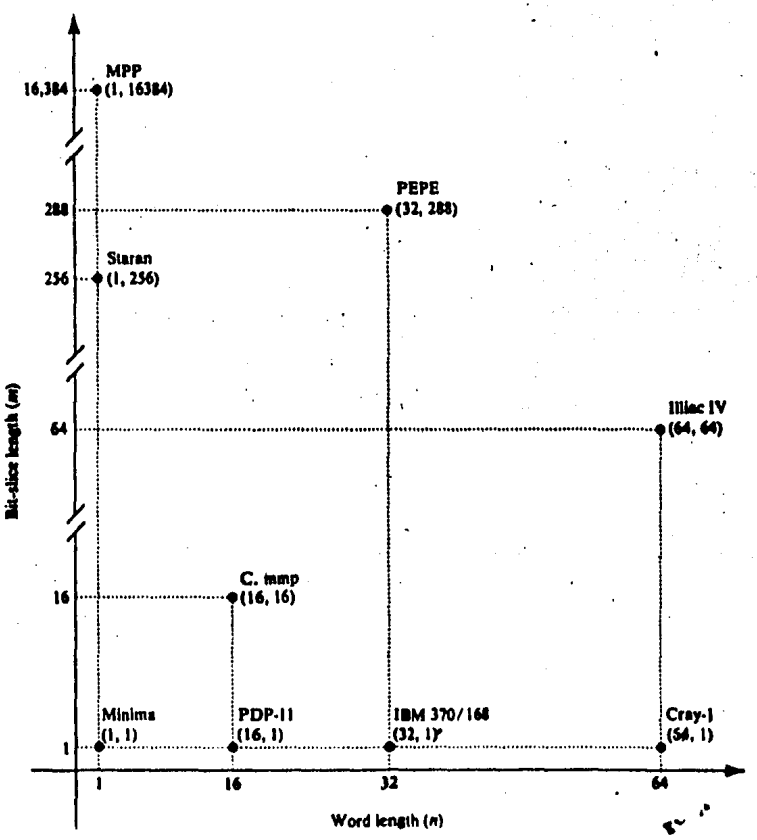


Figura 1.2.2. Clasificacion de Feng en terminos del paralelismo exhibido por la longitud de palabra y la longitud del segmento por bit.[1]

Mode	Computer model (manufacturer)	Degree of parallelism (n, m)
WSPS	The "MINIMA	(1, 1)
$n = 1$	(unknown)	
$m = 1$		
WPBS	STARAN	(1, 256)
$n = 1$	(Goodyear Aerospace)	
$m > 1$	MPP	(1, 16384)
(bit-slice processing)	(Goodyear Aerospace)	
	DAP	(1, 4096)
	(ICL, England)	
WSBP	IBM 370/168 UP	(64, 1)
$n > 1$	CDC6600	(60, 1)
$m = 1$	Burrough 7700	(48, 1)
(word-slice processing)	VAX 11/780	(16/32, 1)
	(DEC)	
WPBP	Illiac-IV	(64, 64)
$n > 1$	(Burroughs)	
$m > 1$	TI-ASC	(64, 32)
(fully parallel processing)	C.mmp	(16, 16)
	(CMU)	
	S-1	(36, 16)
	(LLNL)	

Tabla 1.2.3. Clasificacion de algunos sistemas segun Feng.[1]

Mediante ejemplos de computadoras reales se puede ver más claramente estas descripciones. La TI-ASC tiene un controlador para cuatro pipelines aritméticos, cada uno de los cuales tiene ocho etapas y 64 bits por etapa. Tenemos entonces

$$T(ASC) = \langle 1 \times 1,\ 4 \times 1,64 \times 8 \rangle = \langle 1,4,64 \times 8 \rangle$$

Cuando K', D' o W' es igual a 1 no se acostumbra poner, ya que el pipelining de una etapa no tiene sentido.

Otro ejemplo es la Control Data 6600 que tiene un CPU con un ALU que tiene 10 funciones especializadas en hardware, cada una utilizando palabras de 60 bits. Hasta 10 de estas funciones pueden ser encadenadas para formar un pipeline mas largo.

$$T(CDC\ 6600) = T(procesador\ central) \times T(procesador\ de\ E/S)$$
$$= \langle 1,1 \times 10,60 \rangle \times \langle 10,1,12 \rangle$$

En la tabla 1.2.4 se pueden observar 10 sistemas bajo esta clasificacion.[1]

Clasificación de Gajski y Peir.

Finalmente concluimos esta sección describiendo un esquema de clasificación propuesto por Gajski y Peir [14] basado en cinco problemas cruciales en el procesamiento paralelo. Se describen en detalle en otra sección.

Computer model $T(C)$	System specification† $\langle K \times K', D \times D', W \times W' \rangle$
T(TI-ASC)	$\langle 1, 4, 64 \times 8 \rangle$
T(CDC-6600)	$\langle 1, 1 \times 10, 60 \rangle \times \langle 10, 1, 12 \rangle$ central processor / I/O processors
T(Illiac IV)	$\langle 1, 64, 64 \rangle$
T(MPP)	$\langle 1, 16384, 1 \rangle$
T(C.mmp)	$\langle 16, 1, 16 \rangle + \langle 1 \times 16, 1, 16 \rangle + \langle 1, 16, 16 \rangle$
T(PEPC)	$\langle 1 \times 3, 288, 32 \rangle$
T(IBM 360/91)	$\langle 1, 3, 64 \times (3 \sim 5) \rangle$
T(Prime)	$\langle 5, 1, 16 \rangle$
T(Cray-1)	$\langle 1, 12 \times 8‡, 64 \times (1 \sim 14) \rangle$
T(AP-120B)	$\langle 1, 2, 38 \times (2 \sim 3) \rangle$

† K', D', and W' are omitted when equal to 1.

‡ For Cray-1, the pipeline chaining degree is a variable with a maximum value equal to 8.

Tabla 1.2.4. Diferentes sistemas bajo la clasificación de Händler.[1]

1) Control jerárquico.
2) Particionamiento.
3) Distribución (schedualing).
4) Sincronizacion.
5) Acceso a memoria.

En la tabla 1.2.5 podemos observar cinco sistemas clasificados bajo este sistema. Podemos observar, por ejemplo, que la Arvind y la HEP difieren básicamente en dos puntos: granularidad y particionamiento. La HEP explota paralelismo a nivel de procesos, mientras que la Arvind utiliza granulación a nivel de instrucciones. En la HEP los usuarios deben realizar la particion; la Arvind cuenta con un compilador y un lenguaje de flujo de datos para este efecto.

Por otro lado podemos notar en la tabla que la Cedar y la Ultracomputer no son tan parecidas como lo sugieren sus topologias. La Cedar tiene un control a nivel de procesos que no existe en la Ultra. La Cedar realiza particionamiento mediante un compilador; en la Ultra el usuario es el que lo realiza. La distribución esta centralizada en la Cedar mientras que en la ultra es distribuida. Así mismo, los primitivos de sincronización son diferentes en las dos maquinas.[14]

MACHINES	MODEL OF COMPUTATION	PROGRAM PARTITIONING	SCHEDULING	SYNCHRONIZATION	MEMORY ACCESS
Cray-1	2-level: Task: serial, PC Inst.: Parallel, DD, CF	User, Fortran ext. Vectorizing compiler	Task: static	Synchronous clock	Vector registers Vector parallelism
Arvind's Data flow	2-level: Task: parallel, DD, CF Inst.: parallel, DD, DF & CF	User, Data flow lang. Compiler	Task: dynamic cen. Inst.: static	F/E bit in I-structure	Message-passing (scalars) Inst. parallelism
HEP	3-level: Task: parallel, DD, CF Process: parallel, DD, CF Inst.: serial, PC	User, Fortran ext.	Task: dynamic cen. Process: dynamic dis.; self-scheduling	F/E bit	Register cache Process parallelism
NYU	2-level: Task: parallel, DD, CF Inst.: serial, PC	User, Fortran ext.	Task: dynamic dis.	Fetch&add	Private cache
Cedar	3-level: Task: parallel, DD, CF Process: parallel, DD, CF Inst.: serial, PC	Parafrase compiler	Task: dynamic cen. Process: static	Counter & bit-map	Private cache Shared cache

PC: program counter; DD: data-driven; DF: data flow; CF: control flow; cen.: centralized; dis.: distributed.

Tabla 1.2.5. Comparacion de cinco maquinas según Gajski y Peir.[14]

3 - Pasado, Presente y Futuro del Procesamiento en Paralelo

En esta sección se encuentra una reseña del procesamiento en paralelo desde sus orígenes hasta su estado actual; inclusive se citan algunos de los sistemas que estan siendo construidos actualmente o que estan por serlo.

Se van siguiendo los distintos esfuerzos que se fueron haciendo a lo largo de la historia para lograr paralelismo. Se describe como mientras que por un lado se lograba conectar varios sistemas para cooperar a la resolucion de una sola tarea, por otro lado se incrementaba la longitud de la palabra y de los buses asi como de los relojes con los que operaban los sistemas. Como se lograba paralelismo mediante la utilización de unidades funcionales especializadas, y como se lograba concurrencia utilizando software para pasar de procesamineto batch a multiprogramacion, luego a tiempo compartido y finalmente al multiprocesamiento.

El paralelismo es algo conceptual. Es cierto que el termino "procesamiento en paralelo" ha sido utilizado de muchas formas desde el comienzo de la historia de la computación. Hace veinticinco años se refería a operaciones aritmeticas sobre palabras enteras en lugar de sobre bits. Actualmente se refiere a multiprocesadores maquinas tipo arreglo (paralelas o pipeline), y hasta ocasionalmente a maquinas multiprogramadas.[9]

Es posible lograr actividad concurrente duplicando idénticos recursos, mediante la especialización funcional de unidades semi-independientes (unidades de E/S, por ejemplo), o utilizando espacio de colas locales (sistemas operativos multiusuario, por ejemplo). Estos son algunos de los medios para lograr formas aparentemente diferentes pero en esencia identicas de paralelismo tales como: paralelismo entre instrucciones diferentes, paralelismo entre etapas de instrucciones y paralelismo entre distintas tareas.

Estas formas de paralelismo son realmente lo mismo si definimos las subtareas de nuestro procesador en unidades de tamaño discretas lo suficientemente pequeñas y abandonamos nuestro concepto de ejecución de instrucción como el de operacion fundamental. Por ejemplo, si en un sistema tipo batch tomamos como operación fundamental a las tareas, entonces pensaríamos que no existe paralelismo en el sistema. Sin embargo, si reducimos la unidad de tamaño al de instrucción, vemos que cuando se está ejecutando una instrucción, la siguiente ya se está decodificando; aunque no existe paralelismo a nivel de tareas, si existe a nivel de instrucciones.

Dos extremos de un continuo. A lo largo de la historia se ha podido observar al "uniprocesador" evolucionar a una maquina cuya naturaleza secuencial fue disminuyéndose y que esperaba sólo unas cuantas modificaciones más para emerger como un multiprocesador en el mas amplio sentido de la palabra.

Sin embargo, este no fue el único camino que se siguió hacia el multiprocesamiento. El otro camino partio de las multicomputadoras. La figura 1.3.1 ilustra el desarrollo desde ambos extremos.

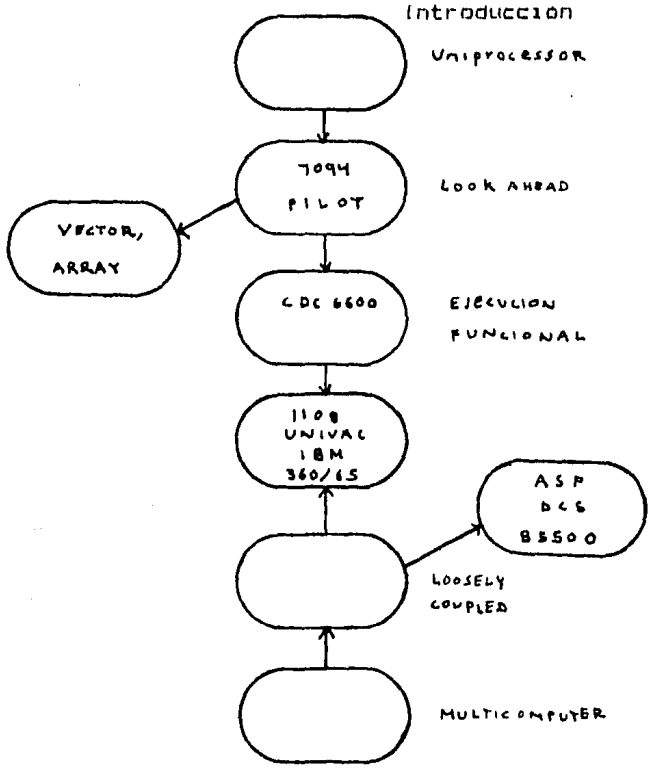


Figura 1.3.1. Dos extremos de un continuo [8].

Observamos el desarrollo del uniprocesador bajando en etapas con cada vez mayor capacidad en el lado de E/S de la parte funcional CPU-E/S. En la otra direccion tenemos cada vez mayores acoplamientos de computadoras independientes para formar tanto configuraciones maestro-esclavo como sistemas generales de multiprocesamiento. Un sistema maestro-esclavo, normalmente incluye el concepto de especializacion funcional, mientras que uno de multiprocesamiento tiene como caracteristica notable la presencia de identicas unidades de procesamiento. En cualquier sistema, por supuesto, existiran unidades especializadas funcionalmente, dentro de los procesadores o para soportar E/S. [8]

Origenes.

Tradicionalmente, las computadoras han ejecutado una sola operacion a la vez. Obviamente, una manera de aumentar la velocidad de ejecucion de un programa es realizando mas de una operacion al mismo tiempo. En 1946 se comenzo la construccion de una computadora que perseguiria este objetivo en su forma más elemental: **aritmética binaria paralela** en lugar de serial. La computadora IASC diseñada por el profesor Von Neumann y sus colegas en Princeton, New Jersey, utilizaba almacenamiento de acceeso aleatorio electrostatico o de un sistema de tubo de rayos catodicos, y aritmetica binaria paralela. [11]

La idea de realizar varias operaciones a la vez parece tener al menos 130 años de antiguedad ya que leemos en la publicacion

de Octubre de 1842 de la descripcion hecha por Menabrea de las conferencias de Babbage; "... cuando una larga serie de cálculos se realiza, tales como los necesarios para la creacion de tablas números, la maquina puede ser puesta en juego de tal forma que de varios resultados a la vez, que reducirían la dimensión total del proceso."[9]

El primer "verdadero" multiprocesador operativo, fue entregado en 1962. De hecho, el sistema operativo para este sistema, el D-825, fue uno de los primeros sistemas operativos de soporte general. El sistema de Burroughs que se utilizaba en aplicaciones militares podia tener hasta cuatro procesadores que compartian toda la memoria.[12]

Sin embargo, mucho antes del D-825 ya existian sistemas con procesamiento en paralelo, aunque estos no eran multiprocesadores en el mas amplio sentido. El sistema de Bell Telephone Laboratories Modelo V que fue construido por Stibitz y Williams a finales de los años 40's tenia dos procesadores. Diseños de maquinas capaces de ejecutar simultáneamente varias operaciones a la vez para resolver ecuaciones diferenciales parciales comenzaron a aparecer en los 50's. En 1952, Leondes y Rubinoff propusieron un procesador de multioperaciones orientado alrededor de una memoria de tambor. En 1958 el pionero alemán de las computadoras Zuse, propuso una máquina paralela orientada a tambor.[9]

Mientras tanto, se iba aumentando la **longitud de la palabra** de las computadoras. El modelo 705 de IBM entregado en 1956 tenia una memoria organizada en grupos de cinco caracteres logrando tiempos de acceso de 17 micro segundos. Las series 1103 utilizaron palabras de 36 bits.

Por otro lado estaba la tendencia de **especificidad funcional.** La 709 de IBM era muy similar a su antecesor la 704 pero tenia un nuevo sistema de E/S que permitia realizar las operaciones de lectura y escritura a periféricos mientras que el procesamiento continuaba.

En 1955 se desarrollaba la Elcom 120 y 125 en la Underwood Corporation bajo la direccion del doctor Lubkin. El sistema 125 incluia un procesador de archivos independiente para oredenamientos y otros procesamientos basicos de datos.

La Electronic Data Corporation comenzó a entregar el sistema Datatron en 1953. Se desarrolló para este sistema uno de cinta magnética que permitia realizar busquedas de bloques de 20 palabras mientras que la computadora seguia realizando otro procesamiento. En 1956 Burroughs absorbió esta compañía.

Estaban tambien los esfuerzos por lograr concurrencia mediante software; un primer esfuerzo más notable fue la Honeywell 800 que tenia un sistema de **multiprogramación** asistido por hardware muy interesante.[11]

Por otro lado venia la corriente de **multicomputación** ya desde 1958 con la conección de tres sistemas que trabajaban en cooperacion hecha por la Nationl Bureau of Standards. Más adelante, en 1963, IBM anuncio su sistema Direct Coupled que combinaba una 7094 con una 7040 o 7044. La computadora más pequeña actuaba como procesador de E/S y supervisor dejándole realmente a la 7094 la ejecucion de las tareas.

Máquinas de arreglos y pipeline. A partir de los primeros años de la década de los 60's han surgido una serie de computadoras de alta velocidad que tienen capacidad de algún tipo de multioperación. La CDC 6600 de los 60's fue sucedida por la 7600 de los primeros 70's. IBM introdujo la 369/91 y sus sucesores. La 7600 y la 360/91 son ambas maquinas pipeline y logran alto rendimiento operando sobre arreglos de datos. Sin embargo, los conjuntos de instrucciones son bastante tradicionales. En contraste, las máquinas pipeline Control Data Star y Texas Instruments ASC tienen conjuntos de instrucciones de vectores, lo cual facilita sustancialmente la compiliación.

Por otro lado se crearon las máquinas de arreglos, como la Illiac IV de Burroughs. Sin embargo, el conjuto de instrucciones de esta máquina todavía era de naturaleza secuencial. Los vectores debian ser distribuidos en particiones de 64 elementos y realizar los ciclos sobre estas particiones. La Goodyear Aerospace Staran IV era un arreglo paralelo de procesadores cada uno de los cuales operaba en forma serial por bits. Este es un ejemplo de procesador asociativo.[9]

Presente y Futuro.

Hemos ido siguiendo el desarrollo del procesamiento en paralelo desde sus inicios. Hemos visto que las primeras computadoras electrónicas eran seriales por bit y completamente secuenciales. Despues, los diseñadores logicos comenzaron a buscar paralelismo a nivel de dispositivos y compuertas, para desarrollar, por ejemplo, algoritmos de multiplicación de 5 y 12 bits, y logica general de bits en paralelo. A nivel de arquitectura de registros cada vez se utilizaron buses y registros de mayor tamaño. La ejecución de instrucciones en forma pipeline estaba reservada a las supercomputadoras y a los sistemas de mayor tamaño. Ahora estas técnicas ya comenzaron a aparecer en microprocesadores de 16 y 32 bits. Mientras tanto, los diseñadores de sistemas y los analistas de sistemas trabajaban en el otro extremo del procesaminento en paralelo, multicomputadoras y multiprocesadores, para en los años 60's sacar los primeros sistemas de estos tipos. La aparición de las comunicaciones simétricas y duplex hicieron posible el desarrollo del multiprocesador de memoria compartida y falla suave (fail-soft). Los sistemas operativos para controlar estos sistemas fueron apareciendo más lentamente, y sus complejidad presionó a los arquitectos de sistemas a diseñar configuraciones homogéneas tipo MIMD.

Las aplicaciones militares y de tiempo real en línea tanto comerciales como industriales provocaron la necesidad de crear sistemas que no fallarán, o que en caso de fallar perdieran a lo sumo una transición recuperable. Hoy, la disponibilidad de microprocesadores con gran capacidad esta tendiendo a la producción de sistemas de alta diponibilidad y tolerantes a fallas (high-availability/fault-tolerant) tipo MIMD tanto en el mercado de los minimultiprocesadores como en el de los multiprocesadores de mayor tamaño.[13]

Cuatro fases de desarrollo. La corriente principal en el uso de las computadoras ha sido en cuatro niveles ascendentes de

computación. Inicialmente se utilizanban las computadoras para procesamiento de datos, en el cual la tarea primordial es el "number crunching". El siguiente nivel es el procesamiento de información en el cual entidades de datos interrelacionadas son procesadas de manera estructurada. La mayoría de las computadoras actuales estan siendo utilizadas para estos dos propositos.

La siguiente fase, el procesamiento de conocimiento, será la de mayor importancia en los 90's. En esta fase, las entidades de datos de la fase anterior adquieren una semántica.

Finalmente, los ingenieros desean utilizar a las computadoras del futuro para el procesamiento de inteligencia mediante el cual se pueden adquirir nuevos conocimientos a través del aprendizaje de la computadora proporcionandole a esta grandes bases de datos de conocimiento. Ver figura 1.3.2.

Actualmente estamos entrando a la era del procesamiento del conocimiento. Las demandas de cómputo se han incrementado grandemente y existen problemas que tomarían de 500 a 100 horas para resolverse en las computadoras actuales. Esta es la razón

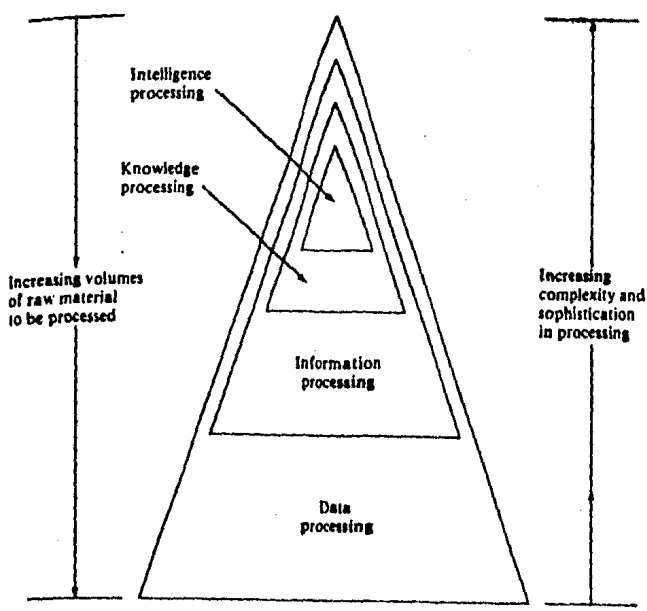


Figura 1.3.2. Cuatro fases de desarrollo de las computadoras [1].

del desarrollo de las **supercomputadoras**, maquinas con algun tipo de procesamiento en paralelo con capacidad de realizar en el orden de cientos de millones de instrucciones de punto flotante por segundo (100 M-flops) con una longitud de palabra de alrededor de 64 bits y capacidad de memoria de millones de palabras. Estas maquinas comenzaron a surgir en los 70's y estan por entrar a su tercera generación.

Una breve historia del procesamiento en paralelo se resume en la tabla 1.3.1 desde 1958 hasta finales de los 70's. De los 70's en adelante se puede observar el desarrollo de las supercomputadoras en la figura 1.3.3.

En la figura 1.3.3. se comparan las velocidades pico de nueve sistemas en terminos de millones de operaciones por segundo. Las velocidades de operación estan medidas en base a operaciones de 64 bits de punto flotante, exceptuando la del MPP que tiene una velocidad pico que varia desde 6.5 G-operaciones enteras de 8 bits hasta 216 M-flops de 32 bits.

Recientemente se anunciaron tres supercomputadoras en Japón. Las velocidades pico de estas maquinas van desde 500 M-flops hasta 1.3 G-flops. La computadora mas rapida jamas construida, por mucho, es la NEC SX-2 (tomando en cuenta velocidad absoluta del hardware). La Galaxy de China, fue anunciada como un procesador de vectores de 120 M-flops.

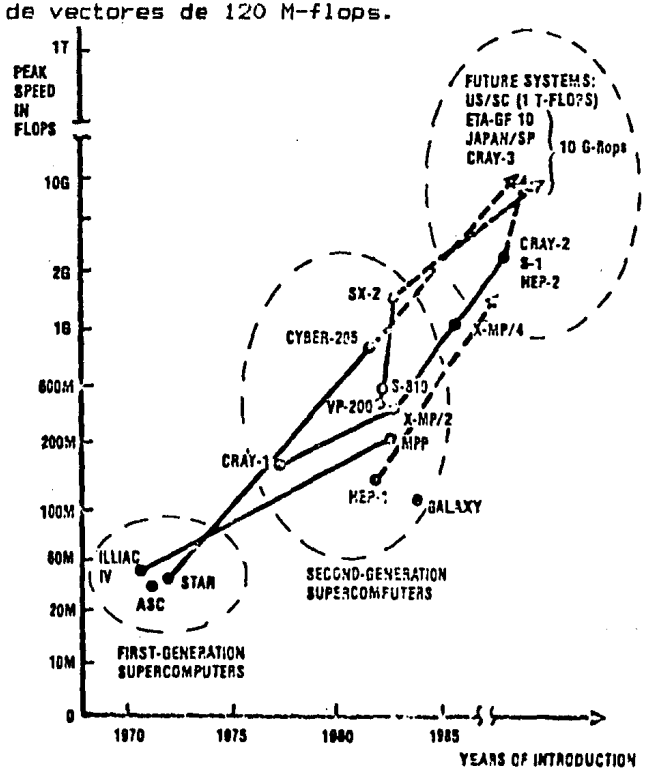


Figura 1.3.3. Desarrollo de las supercomputadoras [4].

Date	System Manufacturer and Model Number	Remarks
1958	National Bureau of Standards PILOT	Three independently operating computers that could work in cooperation.
1958 (circa)	IBM AN/FSQ-31 and 32	Solid-state SAGE computer; not multiprocessors; merely duplexed systems.
1960	Burroughs D-825 (This system carried various military model designations depending on the major system of which it was part.)	First modular system with identical processors. Total memory shared by all processors. Up to four processors, 16 memory modules, 10 I/O controllers, and 64 devices. Important feature was one of the earliest examples of a modern operating system—the Automatic Operating and Scheduling Program (ASOP).
1960	Ramo Wooldridge TRW-400	"Polymorphic system"; for USAF command and control. Some construction done, not completed. Important for early concepts.
May 1960	Univac LARC	One I/O processor and one computational processor capable of operating in parallel. One delivered to Livermore AEC Laboratories. Not a "true" multiprocessor.
May 1961	IBM STRETCH (7030)	Original design called for separate character-oriented processor and binary arithmetic processor. These were dropped from final design; therefore, final product was not a multiprocessor. It did contain look-ahead. Only seven delivered.
Feb. 1963	Burroughs B-5000	One or two processors. Up to eight memory modules. Programs independent of addresses. Supervisor was the Master Control Program (MCP). Utilized virtual memory concepts and hardware. Machine code based on Polish notation. Users programmed only in ALGO or COBOL. Became the B-5500 in Nov. 1964.
1963	IBM 704X/709X (7040 or 44 and 7090 or 94)	"Direct Coupled System"
1963	Bendix G-21 (later CDC)	A multiprocessor version of the G-20 developed for Carnegie Institute of Technology. A crossbar system.
1963	IBM, MSC	A custom multiprocessor system to support Manned Space Center. Originally 7090s sharing large core; later 360/75s.
Sep. 1964	CDC 6600	Contained multiple arithmetic and logic units each of which could execute only a small fraction of the total instruction repertoire. Ten peripheral processors were an integral part of the system. (Number of PPUs increased to 20 in 1969.) The PPUs constitute a multiprocessor system. Overall system an example of an asymmetric multiprocessor.
Nov. 1964	Burroughs B-5500	An upgrade of the B-5000 (see Feb. 1963).
1964	GE 645 (now Honeywell)	Ordered by Project MAC at MIT.
May 1965	GE 645 (now HIS-645)	Delivered to Project MAC at MIT. Hardware not a standard product; however MULTICS operating system is being released.
Nov. 1965	UNIVAC 1108	
1965	SOLOMON I	Design only. First large scale array processor

Tabla 1.3.1 Historia [12].

Date	System Manufacturer and Model Number	Remarks
Mar. 1966	IBM S/360 Model 67	Special dual-processor time-sharing system.
Apr. 1966	CDC 6500	Dual 6400s
Dec. 1966	XDS SIGMA 7	
1966	SOLOMON II	Design only.
Jun. 1967	CDC 6700	Dual CDC 6600s
Aug. 1967	XDS SIGMA 5	
1968	CDC 7600	Very similar to 6600, but higher speed and included hierarchy of main memory as standard feature.
Apr. 1969	IBM S/360 Model 65 MP	Dual-processor version of standard Model 65.
Jun. 1970	XDS SIGMA 6	
Oct. 1970	Burroughs B-5700	Similar to B-5500 with capability for increased memory. Capability for four B-5700 systems to share disk storage.
1970	CII IRIS 80	True multiprocessor; processors considered as anonymous resources; virtual memory.
Feb. 1971	Honeywell 6050, 6060, 6080	
Jun. 1971	Burroughs B-6700	
Sep. 1971	DEC System 10/1055, 10/1077	
Sep. 1971	XDS SIGMA 8, 9	
Nov. 1971	UNIVAC 1110	
1971	SDC PEPE (Parallel Element Processing Ensemble)	Prototype for processing of radar data for ballistic missile defense system.
1971	Fairchild SYMBOL 2R	Seven processors dedicated to separate functions
Jan. 1972	UNIVAC 1106	
Jan. 1972	Honeywell 2088	
Sep. 1972	ILLIAC IV	Array processor. 64 processor elements. Driven by a conventional multiprocessor used as a front-end control processor.
Feb. 1972	Burroughs B-7700	
1972	CDC, CYBER 72, 73, 74, 76	
1972	Goodyear STARAN S	Parallel associative system.
1972	Texas Instruments ASC (Advanced Scientific Computer)	Embodies both multiprocessing and pipelining.
Jun. 1973	Bolt Beranek and Newman, PLURIBUS	Multi-minicomputer stressing reliability.
1973	CDC STAR-100	Pipeline system.
1974	IBM S/370, Models 158 MP & 168 MP	Shared real and virtual storage.
1974	CDC CYBER 175	Similar to CDC 6000 and CYBER 70 series; memory capacity extensions.
1974	Carnegie-Mellon Univ. C.mmp	Multiprocessor with 16 PDP-11s sharing memory through a crossbar.
Mar. 1975	UNIVAC 1100/20, 1100/40	
Oct. 1975	UNIVAC 1100/10	
1975	Tandem, T16	Fault-tolerant multiprocessor
1976	DEC System 10/1088	Dual-processor
1976	CRAY-1	Pipeline system
1976	IBM S/370, Models 158 AP & 168 AP	Asymmetric multiprocessors
Nov. 1976	UNIVAC 1100/80	
1977	Goodyear STARAN E	Parallel associative system.

Tabla 1.3.1 (continuación) Historia [12].

De estos nueve sistemas, el MPP es el único procesador que maneja un solo flujo de instrucción hasta 2^14 flujos de datos. El HEP-1 de Denclors es un sistema multiprocesador que puede tener hasta 16 procesadores. El Cray X-MP/2 es un sistema dual obtenido a partir del Cray-1. El X-MP/4 y el Cray-2 tienen cuatro procesadores cada uno.

En la esquina superior derecha de la figura, se indican varias supercomputadoras del futuro. Estos sistemas podran alcanzar velocidades de por lo menos un G-flop y podran ser finalizadas en la próxima decada.[14]

4 - Razones y Metas del Procesamiento Paralelo

Existen una serie de razones funcionales, y razones economicas y de hardware que justifican la investigacion y el desarrollo de sistemas con procesamiento paralelo. En esta seccion se discuten algunas de estas razones para despues identificar tres grupos distintos de aplicaciones del multiprocesamiento: sistemas orientados a velocidad, sistemas orientados a disponibilidad, y sistemas orientados a respuesta. Las ideas expuestas están basadas en Hobbs y Theis [2] y en Patton [13].

Razones Funcionales.

Algunos de los problemas que han impulsado el desarrollo del procesamiento paralelo, desde el punto de vista funcional, son:

1) Problemas que requieren de gran cantidad de computo.
2) Problemas con paralelismo inherente.
3) Multiprogramacion y multiusuario.
4) Confiabilidad.
5) Respuesta rápida.

Problemas Grandes.

La historia de la computacion ha mostrado que en un punto dado en el tiempo, los usuarios tienen problemas que requieren de gran cantidad de procesamiento y que exceden las posibilidades de los sistemas existentes. A pesar de los esfuerzos y de los logros que se han hecho por aumentar la velocidad de los circuitos lógicos, de las memorias y de los dispositivos de E/S, las necesidades de los usuarios han ido aumentando con la misma velocidad. En los últimos años las demandas de velocidad de computo han crecido marcadamente: existen problemas que para resolverse tomarian de 500 a 1000 horas en las supercomputadoras modernas.

A medida que el mejoramiento en los dipositivos de switcheo y la miniaturización llega a un límite, y que la longitud de

palabra esta limitada por la precisión de la aplicación, es obvio que un mayor incremento significativo en la velocidad de procesamiento puede ser logrado solamente por el procesamiento concurrente de varias palabras.

Algunos ejemplos de estos problemas pertenecen a áreas como: meteorología, física nuclear, hidrodinámica e inteligencia artificial.

Problemas con Paralelismo Inherente.

Existe otro tipo de problemas cuyas necesidades de velocidad de cómputo pueden no justificar la utilización de procesamiento paralelo, pero que sin embargo, el procesamiento de datos envuelto es inherentemente paralelo. Este paralelismo puede ser de dos tipos fundamentalmente. Un caso sería el de los problemas que realizan una misma operación sobre conjuntos de datos diferentes. Por ejemplo, sumar una constante a una matriz implica conceptualmete sumar la constante a una serie de números al mismo tiempo.

El otro tipo de problemas con paralelismo inherente, sería el de los problemas que realizan distintas operaciones sobre distintos conjuntos de datos. En otras palabras el programa puede proceder secuencialemte hasta un punto en el que se "ramifique" de tal forma que distintas rutinas puedan proceder en paralelo hasta un punto en el que se vuelvan a unir. Si estas computaciones en un programa no son muchas, una buena solución es realizarlas secuencialemente, pero si en un programa aparecen en mayor cantidad, utilizando una arquitectura paralela, se puede ejecuetar el programa en forma más eficiente (desde el punto de vista de equipo y económico) y más rápidamente.

Algunos ejemplos de estos problemas se pueden encontrar en áreas como son el procesamiento de señales de radar y sonar, y reconocimiento de patrones.

Multiprogramación y Multiusuario.

Las técnicas de multiprogramación y multiusuario tienen como objetivo hacer un uso más eficiente de los recursos de una computadora. En los sistemas multiusuario y con multiprogramación, el sistema operativo controla la ejecución concurrente de varios programas y asegura que no se interfieran entre si, mientras que al mismo tiempo trata de que los recursos de la maquina se utilicen eficientemente. Los sistemas operativos de este tipo consumen una gran cantidad de recursos en términos tanto de tiempo de ejecución como de memoria. En un sistema de multiprocesamiento se pueden reducir estos requerimientos particularmente si se utilizan distintas unidades de control para cada programa en tanto que las unidades de procesamiento y E/S se comparten por varios programas.

Confiabilidad.

En una computadora convencional, la falla de alguna unidad importante (aritmetica, de control, de memoria, etc.), provoca que el sistema quede fuera de operación hasta que la falla sea corregida. En una organización con procesamiento paralelo, la falla de una unidad puede no interferir con la operación de otras unidades iguales a esta. Por lo tanto un sistema de procesamiento

paralelo incrementa la confiabilidad; particularmente si se incluye algo de redundancia. Aun sin emplear redundancia, un sistema paralelo puede permitir confiabilidad parcial (graceful degradation) al deshechar las funciones menos criticas en el caso de una falla, mientras que las de mayor importancia continúan corriendo en las unidades que aún esten en operación.

En una organización altamente paralela, la falla de una unidad reduciria muy poco el desempeño total del sistema, suponiendo que el sistema tenga la capacidad de detectar la falla y reconfigurar el problema de tal forma que las funciones criticas se realicen en las unidades restantes.

La confiabilidad es deseable en sistemas como son los de tiempo compartido o en procesos de control, pero es esencial en otras aplicaciones como son las militares. Más aún, debido a que la falla puede ser resultado de ataques del enemigo, puede ser conveniente no solo la duplicación de unidades si no también su distribucion física.

Respuesta Rápida.

Existen problemas en los que es muy importante garantizar no tanto la velocidad o la confiabilidad del sistema de cómputo, sino su respuesta rápida. Este es el caso de los sistemas multiusuario en los que es importante dar al usuario la sensación de que el es el unico trabajando en el sistema. En estos casos no es critica la velocidad ni la confiabilidad del sistema.

Existen otras aplicaciones en las que una respuesta rápida es critica, como pueden ser algunas aplicaciones militares y de control. En [15] se describe un sistema para la recepción de señales extraterrestres en el que la respuesta rápida es muy importante.

Razones Económicas y de Hardware.

Aún en el caso de que las razones funcionales no justificasen el desarrollo de los sistemas de procesamiento paralelo, existen razones tecnológicas que favorecen a las organizaciones paralelas. Algunos de los nuevos desarrollos tecnologicos tales como arreglos de circuitos de muy alta integracion, memorias de semiconductores de alta integracion, técnicas ópticas, etc. pueden ser utilizadas más eficientemente en algunos tipos de arquitecturas paralelas.

Un ejemplo es el desarrollo de procesadores de arreglos VLSI [16]. Otro ejemplo es el problema del uso repetitivo de circuitos integrados de alta capacidad. Con las técnicas de diseño actuales, la tendencia es que dentro de unos años se pueda construir una computadora relativamente sofisticada con unos 20 chips. Sin embargo las organizaciones de computadoras convencionales requerirían que casi todos los chips utilizados en la fabricación de una computadora sean de tipos distintos. En cambio, una organización paralela permite el uso repetitivo de circuitos integrados iguales, es decir una computadora paralela tendria relativamente pocos circuitos integrados diferentes pero muchos del mismo tipo, trayendo como consecuencia grandes ventajas económicas y de diseño.

Los avances en el diseño de memorias es otro ejemplo de

factores tecnológicos que se adaptan muy bien al procesamiento paralelo; se pueden usar modulos de memoria que se accesan en paralelo, o memorias asociativas para el diseño de procesadores asociativos.

Tres Enfoques

En la tabla 1.4.1 se comparan tres estilos de multiprocesamiento junto con sus metas respectivas.

Los primeros avances en el desarrollo de hardware y software para multiprocesadores estaban orientados a incrementar la velocidad de los sistemas con respecto a los uniprocesadores; sin embargo, la motivación original fue la necesidad de tener sistemas en línea y en tiempo real con alta confiabilidad y disponibilidad (avalilability).

La propiedad de falla suave (un sistema con falla suave, fail soft, preserva su último estado operacional mientras que uno falla segura, fail safe, preserva la integridad completamente) se trata de lograr mediante la redundancia en hardware, sin embargo se necesitó que pasara una década para poder tener sistemas comerciales falla segura y orientados a disponibilidad.

THROUGHPUT-ORIENTED MULTIPROCESSING	AVAILABILITY-ORIENTED MULTIPROCESSING	RESPONSE-ORIENTED MULTIPROCESSING
• General purpose • Fail-soft requirements • Multiple applications • CPU and I/O balanced workload • Maximize number of independent jobs done in parallel	• Real-time/on-line • Fail-safe or never failing • Database-centered applications • I/O intensive • Maximize number of interdependent tasks done in parallel	• Dedicated/embedded • Performance is failure-sensitive • Application specialized • CPU intensive • Maximize number of cooperating processes done in parallel

Tabla 1.4.1. Aplicaciones del multiprocesamiento.[13]

La meta de los multiprocesadores **orientados a velocidad** es obtener altas velocidades al más bajo costo de cómputo posible en un ambiente de propósito general teniendo el mayor número posible de tareas corriendo en paralelo. Las técnicas que utilizan los sistemas operativos para lograr esta meta es tomar ventaja del balance que debe haber en las cargas de trabajo de la UCP y de las unidades de E/S. El ancho de banda de la memoria compartida es uno de los limitantes del desempeño de un multiprocesador.

Los sistemas de **alta disponibilidad** son generalemente

sistemas interactivos y muchas veces con requerimientos en línea, en tiempo real y sin fallas (never-fail). Estas aplicaciones usualmente están centradas alrededor de una base de datos común y están limitadas por E/S más que por memoria. Los sistemas de alta velocidad fueron una consecuencia natural de los orientados a velocidad. El requisito básico para un sistema de alta disponibilidad en la mayoría de las aplicaciones, es que cada componente importante tanto de hardware como de software esté por lo menos duplicado. El sistema requiere de un mínimo de dos procesadores y posiblemente de dos vías conectando a los dos procesadores; es también deseable tener al menos dos vías de los procesadores a la base de datos. Los controladores de E/S deben ser multipuerto para poder ser conectados con los múltiples procesadores.

La meta de los multiprocesadores **orientados a respuesta** es minimizar el tiempo de respuesta para demandas de cómputo. Las aplicaciones para estos sistemas son por naturaleza de intenso cómputo y la mayoría se pueden particionar en múltiples tareas que pueden ser ejecutadas concurrentemente en varios procesadores. En el pasado, los procesadores SIMD y MIMD eran muchas veces máquinas de propósito específico dedicadas a una sola clase de aplicación científica o de procesamiento de señales en tiempo real. Las aplicaciones de la "quinta generación" junto con la disponibilidad de microprocesadores VLSI han hecho resurgir el interés por este tipo de computadoras que pueden manejar tareas numéricas, simbólicas o de procesamiento de señales concurrentemente.

5 - Aplicaciones del Procesamiento Paralelo

Se tiene una gran necesidad de computadoras rápidas, eficientes y confiables en muchos problemas de ingeniería, científicos, de fuentes de energía, médicos, militares, de inteligencia artificial y de investigación básica. Se necesitan sistemas con procesamiento paralelo para satisfacer estas demandas, que son básicamente para las cuales se han desarrollado: rapidez, confiabilidad, flexibilidad y disponibilidad.

A continuación, en primer lugar, vamos a describir las aplicaciones del procesamiento paralelo en cuatro categorías de acuerdo a sus objetivos. Dentro de cada categoría vamos a identificar varias áreas representativas de aplicaciones que han ocupado a científicos, ingenieros y programadores en todo el mundo [1]. En segundo lugar vamos a describir y a dar referencias de algunas aplicaciones más concretas que se hallan desarrollado o estén por serlo.

I- Aplicaciones Importantes del Procesamiento Paralelo.

1. Modelado y simulación.

El modelado multidimensional de la atmósfera, la Tierra, el espacio exterior, y la economía mundial se han convertido en areas de gran interés científico. En estos problemas se diseña un modelo que trata de predecir situaciones y que al implementarlo en una computadora imita una parte del mundo real. Hasta ahora las computadoras construidas no han logrado satisfacer las demandas de las simulaciones científicas. Se requiere en ocasiones de velocidades de cómputo que se acercan a los mil millones de mega flops o aún más.

A. Predicción climatológica numérica. Se requiere del modelado climatológico para predicciones a corto y largo plazo de peligros como ciclones, sequías y contaminación atmosférica. Las necesidades de cómputo en esta área son enormes.

Los cálculos se realizan sobre un arreglo tridimensional que hace una partición de la atmósfera en k niveles verticales, m intervalos de longitud y n de latitud. Además se considera una cuarta dimensión p que es el número de intervalos de tiempo de la simulación. Utilizando un cubo de 400 kilómetros por lado una predicción de 24 horas tomaría unas cien mil millones de operaciones. En una computadora de cien mega flops el cálculo duraría 100 minutos.

Si doblamos la resolución en las cuatro dimensiones, el cómputo tomaría por lo menos 16 veces más. Una máquina de 100 Mflops como la Cray-1 tardaría 24 horas en predecir el clima de un día. Predicciones confiables de largo plazo requieren de computadoras mucho mas poderosas que 1.6 Gflops.

B. Oceanografía y Astrofísica. Los estudios oceanográficos requieren mayor resolución de volumen pero menor resolución en tiempo que las simulaciones atmosféricas. Realizar una simulación completa del Océano Pacífico con resolución adecuada para 50 años tardaría en procesarse unas 1000 horas en una computadora Cyber-205.

Se puede simular la formación de la Tierra en una computadora de alta velocidad. El rango dinámico de los estudios astrofísicos puede ir desde billones de años hasta milisegundos. En la Illiac-IV se hizo un estudio tridimensional de integraciones de cuerpos comprendiendo 10^5 partículas moviéndose consistentemente bajo fuerzas Newtonianas.

C. Socioeconomía y Aplicaciones Gubernamentales. Se tiene una gran demanda de computadoras grandes en las áreas de econometría, ingeniería social, censos, control de crimen y modelado de la economía mundial. El modelado de la economía mundial propuesto por W. W. Leontief (1980) realiza operaciones de matrices de gran tamaño en una computadora CDC. En el gobierno de E.U. se utilizan grandes computadoras para censos, auditorías recolección de impuestos y control de crimen. Se estima que el 57 % de las computadoras grandes fabricadas en E.U. han sido utilizadas por el gobierno.

2. Automatización y diseño de ingeniería.

Se han necesitado supercomputadoras para resolver problemas de ingeniería como son los de diseño estructural y experimentos con túneles de viento para estudios aerodinámicos. En la industria se han necesitado para automatización avanzada, inteligencia artificial y sensores remotos.

A. Análisis de elemento finito. Para la construcción de presas, puentes, barcos, aviones supersónicos, altos edificios y naves espaciales es necesario resolver grandes sistemas de ecuaciones algebraicas y diferenciales parciales. Los programas convencionales secuenciales toman cantidades mucho muy grandes de tiempo en ejecutarse. Se han utilizado computadoras como la CDC Star-100 y la Cyber-205 para resolver problemas de análisis estructural.

B. Aerodinámica. La NASA está tratando de reemplazar su Illiac-IV para poder realizar simulaciones en tres dimensiones de túneles de viento a velocidades de Gflops. Se han propuesto dos supercomputadoras conocidas como NASF (Numerical Aerodynamic Simulation Facilities) por Burroughs y CDC capaces de simular diseños enteros de aviones.

C. Inteligencia artificial y automatización. Las supercomputadoras del futuro deberan tener interfases inteligentes para poder comunicarse directamente con humanos mediante imagenes, voz y lenguajes naturales. En la tabla 1.5.1 se listan algunas funciones inteligentes que demandan procesamiento paralelo.

Recientemente se lanzó en Japón un proyecto para desarrollar las computadoras que se usarán en los años noventas. Se pretende lograr velocidades de hasta un billón de instrucciones por segundo.

+ Procesamiento de imagenes.
+ Reconocimiento de patrones.
+ Vision por computadora.
+ Reconocimiento de voz.
+ Inferencia.
+ CAD/CAM/CAI/OA.
+ Robótica.
+ Sistemas expertos.

Tabla 1.5.1. Areas de IA que requieren de procesamiento paralelo.

D. Sensores remotos. La información sensada en forma remota se hace, por ejemplo, mediante satelites. Se procesan cantidades masivas de datos en esta área; por ejemplo, una sola imagen del LANDSAT contiene 30 Mbytes y se necesitan 1.3 imágenes como esta para cubrir todo el estado **Alabama.**

La NASA ordenó la construcción de un sistema (MPP) que

procesara 6 mil millones de operaciones de 8 bits por segundo. Casi pueden proveer analisis de estas imagenes en tiempo real.

3. Exploración de recursos naturales.

Las computadoras juegan un papel importante en el descubrimiento de petroleo y gas, en su administracion y extraccion, en el desarrollo de energia de fusion utilizable, y en la seguridad de reactores nucleares. La utilizacion de computadoras en el area de energia resulta en menores costos de produccion y mejores medidas de seguridad.

En la tabla 1.5.2 se listan algunos campos en los que se utiliza procesamiento paralelo:

+ Exploración sismica.
+ Modelado de reservas.
+ Fusión de plasma.
+ Seguridad en reactores nucleares.

Tabla 1.5.2. Campos donde se utiliza el procesamiento paralelo para la explotacion de los recursos naturales.

4. Investigación básica, médica y militar.

En el area médica se necesitan computadoras rapidas para tomografia, diseño de corazones artificiales, diagnósticos de higado, estimacion de daño cerebral y estudios de ingenieria genetica. En el ejercito se utilizan supercomputadoras para el diseño de armas y simulacion de efectos. Casi todas las areas de investigacion basica necesitan supercomputadoras para avanzar en sus estudios.

A. Tomografía asistida por computadora (CAT). En una computadora convencional, generar imágenes CAT toma entre 6 y 10 minutos. Utilizando un procesador de arreglos dedicado, se puede reducir el tiempo a entre 5 y 20 segundos. Actualmente la reconstrucción de imagenes de la anatomia humana se hace en forma bidimensional y toma demasiado tiempo como para poder captar organos en movimiento como corazon o pulmones. Se espera que el CAT scanner de la Mayo Clinic en Rochester sea capaz de reproducir imágenes tridimensionalmente del corazón latiendo, en unos cuantos segundos.

B. Ingeniería genética. Se ha desarrollado una maquina con alto grado de pipeline llamada Cytocomputer para procesamiento de imagenes biomedicas. Puede ser utilizada para la busqueda de mutaciones geneticas.

C. Aplicaciones militares. La mayoria de las supercomputadoras han sido utilizadas en la industria militar. En la tabla 1.5.3 se listan varias aplicaciones militares de supercomputadoras.

+ Diseño de armamento nuclear (Crav-1).
+ Simulacion de efectos de armas atomicas (Cyber-205).
+ Procesamiento de señales de radar (PEPE).
+ Generacion automatica de mapas (Staran).
+ Armas antisubmarinos (Multiprocesador S-1).

Tabla 1.5.3. Aplicaciones militares de las supercomputadoras.

II- Algunos ejemplos.

En la actualidad estan siendo o ya han sido desarrollados muchos sistemas paralelos de computo. Algunos de ellos fueron concebidos para resolver distintos tipos de problemas especificos, mientras que otros fueron pensados para operar como computadoras de proposito general. Aunque se ha dedicado mucho esfuerzo para el desarrollo de organizaciones paralelas que puedan resolver problemas de tamaño muy grande, tambien se ha trabajado en el desarrollo de sistemas paralelos mas pequeños para resolver problemas con paralelismo inherente.

El desarrollo de procesadores de arreglos de muy alta integracion [16] es un ejemplo de sistemas paralelos de este ultimo tipo. De hecho, algunas tecnicas de procesamiento paralelo han entrado ya practicamente en todas las computadoras modernas (pipelining, multiples unidades especializadas).

Vamos a citar a continuacion algunos sistemas de aplicación especifica y algunas aplicaciones que operan en sistemas de proposito general. El objetivo no es proporcionar una lista exhaustiva, si no solamente dar un panorama general de los sistemas paralelos existentes.

El **centro de investigaciones de Ames** ha estado realizando estudios en el area de dinamica de fluidos. El sistema inicial utilizado constaba de dos VAX 11/780 conectados a una memoria común multipuerto [18].

Computadora Navier-Stokes. La NSC esta siendo desarrollada por Daniel M. Nosenchvik y Michael G. Littman en la Universidad de Princeton para simular numericamente las ecuaciones completas de Navier-Stokes y energia de campo. [18]

La AHR (Arquitecturas Heterarquicas Reconfigurables) aunque no es una maquina de proposito especifico estrictamente, es una computadora diseñada para procesar lenguajes expresables en notacion lambda como LISP. La maquina, desarrollada en el IIMAS de la UNAM, esta compuesta por un numero grande (unas decenas) de microprocesadores Z80 debilmente acoplados, sin jerarquias entre ellos y que comparten memorias publicas. [19]

El paralelismo inherente de muchos problemas de **procesamiento de señales** ha provocado que esta sea una area donde aparecen sistemas paralelos. En el articulo de Allen [5], se

describen arquitecturas para el procesamiento digital de señales.

Una aplicación poco convencional de procesamiento paralelo de señales es la que está siendo desarrollada en la NASA en el proyecto "Search for Extraterrestrial Intelligence" (SETI). El proyecto, que tiene como objetivo detectar señales inteligentes provenientes del espacio extarior, requiere de la realizacion de una cantidad muy grande de procesamiento altamente concurrente se señales por un sistema de hardware especifico. El sistema que se esta considerando es una computadora que en tiempo real debe realizar transformadas discretas de Fourier de millones de puntos.

En el procesamiento de señales de sonar y de radar, las computadoras paralelas han tenido gran importancia. Knapp et. al.[2] desciben una aplicación de la Illiac-IV al problema de defensa urbana por radar. Bird [2] escribe la realizacion de una memoria asociativa para el procesamiento de señales de sonar. Katz, tambien en [2], decribe la realizacion de operaciones con matrices en un procesador asociativo.

En enero de 1984, la revista Proceedings of IEEE publicó un número especial acerca del impacto de las supercomputadoras en la ciencia y la tecnología. Este número contiene varios artículos interesantes acerca de aplicaciones de supercomputadoras, incluyendo investigación en energía de fusión magnetica, análisis de elemento finito, modelado de reserva petrolera y solucion de problemas de aereodinámica.

En Tokio, el **Instituto para Tecnología de Computadoras de la Quinta Generación** (ICOT), ha estado desarrollando a partir de 1982 el prototipo de un nuevo sistema de cómputo capaz de apoyar diversas actividades inteligentes relacionadas con la producción. Se escogió como lenguajes de interfase entre el hardware y el software del sistema a Prolog y Prolog Concurrente. El desarrollo del hardware y de la arquitectura incluirá la implementación de mecanismos para procesar y controlar una base de conocimientos y la ejecucion eficiente de técnicas de inferencia. El sistema estará basado en tecnicas de procesamiento paralelo que serán desarrolladas durante la fase inicial del proyecto. Como primer paso hacia una máquina de base de conocimiento, ICOT está desarrollando una maquina llamada Delta. En [19] se describe la arquitectura de este sistema multiprocesador, así como otros aspectos del proyecto de la quinta generacion de ICOT.

C A P I T U L O II

Procesamiento Pipeline

Principios Básicos de Procesamineto

Pipeline

2.1. Introducción al Procesamiento Pipeline Lineal.

Una tecnica importante para diseñar computadoras con alto grado de desempeño se conoce como Pipeline. En esencia, las computadoras con procesamiento pipeline pueden manejar operaciones concurrentes, es decir, una operacion puede ser iniciada antes que se complete la anterior. El grado de computacion esta determinado por la tasa de operaciones que son iniciadas en vez de por el tiempo necesario para realizar cada operación. La IBM 360/91 y la Control Data 6600 son ejemplos tipicos de computadoras diseñadas con una arquitectura de Pipeline.[2]

El concepto de procesamiento Pipeline utilizado en las computadoras digitales es similar al concepto de una linea de ensamble en las plantas industriales, es decir, un producto debe pasar por una serie de etapas o niveles para que pueda ser considerado como un producto terminado, en cada nivel se lleva a cabo una modificacion especifica que contribuye a la elaboración de dicho producto.

Al igual que en las líneas de ensamble, el procesamiento Pipeline requiere que la tarea de entrada pueda ser subdibidida en una secuencia de subtareas, cada una de las cuales deberá ser ejecutada por un sistema de hardware especializado que trabajará concurrentemente con los otros sistemas.

En un instante determinado, pueden estarse ejecutando tantas tareas como niveles se tengan, dándose una sensacion de sobreponerse unas con otras.

La subdivision de tareas en las lineas de ensamble, trajo como consecuencia inmediata la masificacion de la producción. Asi mismo, el procesamiento Pipeline ha contribuido al mejoramiento de los sistemas de computo sustancialmente.

Idealmente, en una línea de produccion todos sus niveles deben operar a la misma velocidad, de no ser asi, el nivel mas lento sera el cuello de botella de todo el sistema. Este cuello de botella puede ocasionar que niveles posteriores se encuentren sin trabajo. Por este motivo, la labor de subdividir una tarea en una secuencia de subtareas se convierte en un factor crucial para el diseño de un sistema Pipeline.

De esta forma tenemos que dada una tarea T, podemos obtener un conjunto de subtareas $(T_1, T_2, \ldots, T_k)$, entonces una subtarea T_j no puede iniciarse antes de que T_i termine para $i<j$, ahora bien, si T_j no puede iniciarse antes de que todas las subtareas anteriores terminen (T_i, para todo $i<j$), se dice que se tiene una relacion de precedencia lineal. Un sistema de Pipeline lineal puede procesar una serie de subtareas con relacion de precedencia

lineal.

Un sistema de Pipeline lineal se muestra en la figura 2.1. Este sistema esta constituido por un conjunto de niveles dipuestos en cascada. Los niveles son circuitos combinacionales que realizan operaciones aritmeticas. Cada nivel se encuentra separado del siguiente mediante registros de alta velocidad. Estos registros (latches) permiten mantener la informacion del nivel anterior mientras que se genera un nuevo resultado intermedio. El flujo de la informacion dentro de todo el sistema se controla por medio de un reloj que simultaneamente se aplica a todos los registros.

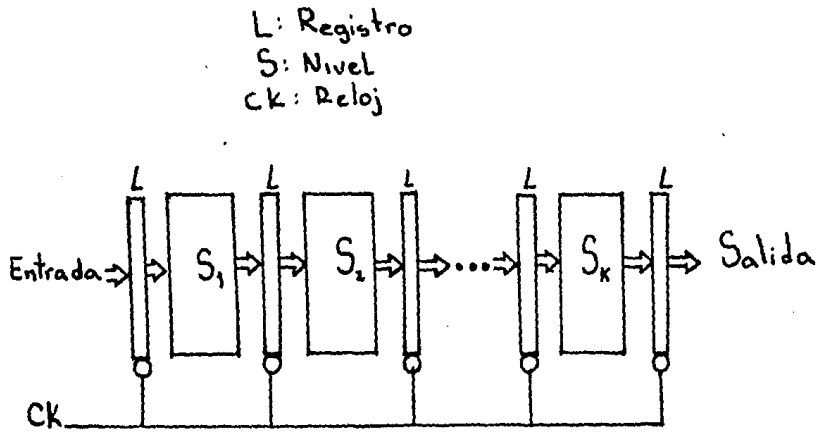


Figura 2.1. Estructura basica de un procesador pipeline lineal. [1]

Cada nivel S_i tiene un retardo t_i debido, principalmente, al tiempo de propagación de los circuitos combinacionales. Cada registro intermedio tiene un retardo de t_l. Se define al **período de reloj** de un sistema pipeline lineal como:

$$t = \max(t_i)_1^k + t_l = tm + t_l$$

donde k es el numero de niveles y el inverso de t será la frecuencia del procesador pipeline.

En la figura 2.2. se muestra un diagrama de tiempo-espacio de un procesador pipeline lineal, se puede ver que las operaciones se encuentran traslapadas. Cuando el procesador pipeline está lleno, en la salida tenemos un resultado en cada período de reloj, independientemente del numero de niveles del procesador pipeline.

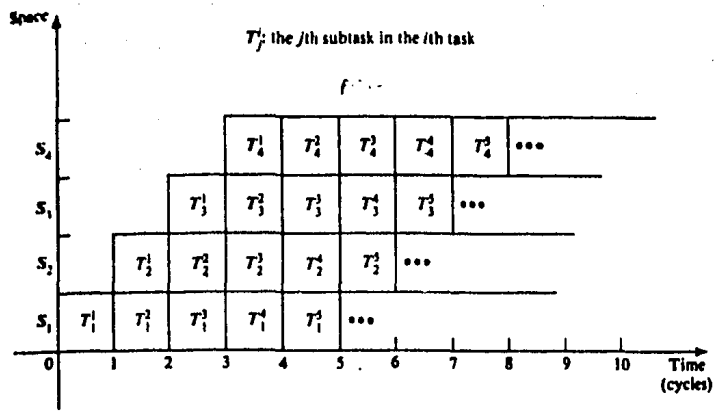


Figura 2.2. Diagrama de tiempo-espacio que muestra el traslape de las operaciones. [1]

En un procesador pipeline de k niveles deben transcurrir k ciclos para obtener el primer resultado, es decir, para que la primera tarea sea completamente ejecutada es necesario que pase por k niveles. Ahora bien, si se tienen n tareas para ejecutarse, idealmente se necesitan n-1 ciclos depues de que la primera tarea ha sido procesada. De esta forma podemos decir que, idelamente, un procesador pipeline de k niveles puede ejecutar n tareas en:

$$T_k = k + (n-1) \qquad \text{periodos de reloj.}$$

Si el procesador no fuera pipeline se necesitarian $T_1 = nk$ periodos de reloj, que resulta ser considerablemete mayor.

Podemos definir al **speedup** de un procesador pipeline de k niveles respecto a un procesador no pipeline, como:

$$S_k = \frac{T_1}{T_k} = \frac{nk}{k + (n-1)}$$

Si hacemos que $n \gg k$, entonces el maximo speedup se tiene cuando $S_k \longrightarrow k$, es decir, el maximo speedup que puede alcanzar un procesaror pipeline lineal es k. Como se analizara mas tarde, este limite de velocidad no se puede obtener en aplicaciones regulares debido a interrupciones, saltos, dependencia entre las instrucciones o cualquier instruccion que rompa la secuencia del programa y genere ciclos de retardo o espera.

Para medir **la eficiencia** de un procesador pipeline lineal

utilizamos el cosiente del speedup del procesador entre el speedup ideal, es decir:

$$Ef = \frac{S_k}{k} = \frac{n}{k + (n-1)}$$

Podemos notar que si el número de tareas es muy grande (n tiende a infinito), entinces se tiene el maximo de eficiencia (Ef tiende a 1). Dicho de otra forma, cuando un procesador pipeline lineal se encuentra en estado estable, la eficiencia será muy cercana a 1. Sin embargo este estado generalmente no se alcanza por las razaones anteriores.

El desempeño de un procesador pipeline se define como el numero total de tareas entre el periodo de observacion kt+(n-1)t, es decir:

$$W = \frac{n}{(kt + n-1)t} = \frac{Ef}{t}$$

Podemos ver que el desempeño se puede expresar como la eficiencia entre el periodo de reloj t. Idealmente, W=1/t = f para cuando Ef-->1. Podemos ver que el maximo desempeño que se puede tener en un procesador pipeline es igual a su frecuencia, o bien, se tendrá un resultado en cada periodo de reloj.

Como un ejemplo de un procesador pipeline lineal, podemos mencionar a la unidad central de procesamiento (CPU) de algunas computadoras modernas [1]. La CPU tiene como tarea, coordinar y ejecutar instrucciones, esta labor generalmente es subdividida en las siguientes etapas: Unidad de instruccion, unidad de cola y unidad de ejecución. La figura 2.3. muestra un diagrama de la CPU dispuesta como un procesador pipeline lineal.

En la memoria principal se almacenan las instrucciones y datos necesarios para el programa. Esta seccion generalmente es un arreglo de memoria. La memoria cache es un dispositivo de almacenamiento muy rápido que contiene segmentos del programa y datos listos para ser ejecutados, de esta forma la CPU puede trabajar eficientemente. La unidad de instrucción contiene los niveles de fetch, decodificación, calculo de la dirección del operando y fetch del operando (si se necesita). La unidad de cola es una memoria FIFO que almacena las intrucciones decodificadas. La unidad de ejecución puede a su vez estar constituida por procesadores pipeline para realizar las operaciones aritmeticas y logicas. En general, si la unidad de instrucción está trabajando con la instrucción I+k+1, la unidad de cola guarda a las instrucciones I+1, I+2, ..., I+k y la unidad de ejecucion ejecuta la instruccion I.

El concepto de procesamiento pipeline se ha convertido en un atributo esencial de la mayoria de las computadoras modernas. Las supercomputadoras tales como la Texas Instruments TI ASC, Burroughs P_{BPE}, IBM 360/91-195, Cray Research C_{RAY}-1, CDC STAR-100, Amdahl 470 V/6, CDC 6600 y CDC 7600 tienen

procesamiento pipeline ya sea en unidades de funciones especiales o internamente en la unidad de instruccion o aritmetica. [3]

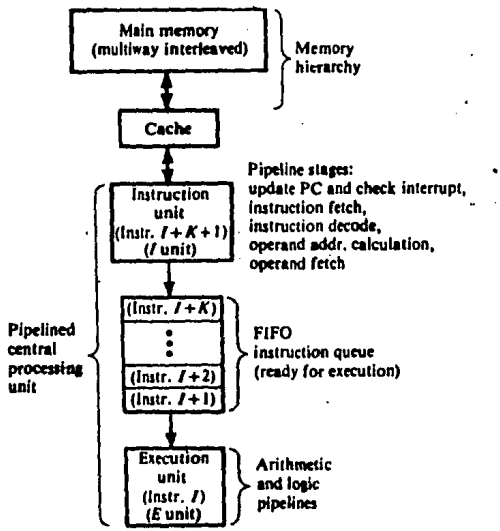


Figura 2.3. Estructura pipeline de una CPU tipica. [1]

2.2. Clasificación del procesamiento pipeline.

Una clasificacion propuesta por Händler, se basa en los niveles de procesamiento; dividiendo en pipeline aritmetico, pipeline de instruccion y pipeline de procesador.

Pipeline aritmético. Las unidades artmeticas y logicas de una computadora se pueden segmentar para ejecutar operaciones pipeline que se encuentran en distintos formatos, ver figura 2.4. Como ejemplos podemos citar a la STAR-100, TI-ASC y la Cyber-205.

Pipeline de instrucción. En la ejecucion de una cadena de instrucciones se sobreponen la ejecucion de la instruccion actual con el fetch, la decodificacion y el fetch del operando de instrucciones posteriores, ver figura 2.5. Esta técnica tambien es conocida como prevision (look-ahead). La mayoria de las supercomputadoras constan de este tipo de procesamiento pipeline.

Pipeline de procesador. En esta técnica se utiliza una serie de procesadores dipuestos en cascada, cada uno procesando una tarea distinta, ver figura 2.6. El primer procesador deposita sus resultados en un bloque de memoria y pueden ser accesados por el segundo procesador. Despues el segundo procesador refina el resultado y lo entrega al tercer procesador, etc. Este tipo de procesamiento pipeline aun no es muy usual.

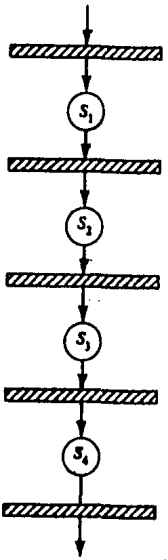


Figura 2.4. Pipeline aritmetico.[1]

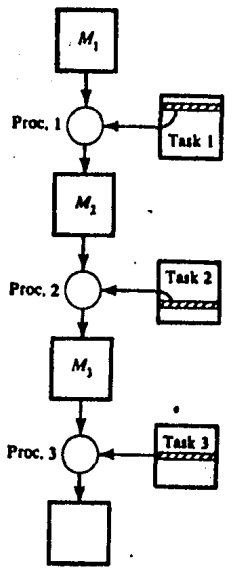


Figura 2.6 Pipeline de procesador.[1]

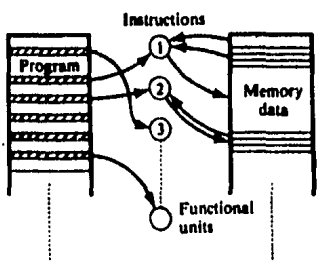


Figura 2.5. Pipeline de instruccion.[1]

Otra clasificacion, basada en las diferentes configuraciones y las estrategias de control, fue propuesta por Ramamoorthy y Li [3], teniendo los siguientes tres esquemas.

Unifuncionales y Multifuncionales. Un procesador pipeline que realiza una tarea especifica tal como una suma de punto flotante, figura 2.3, es conocido como unifuncional. La computadora Cray 1 tiene 12 unidades para varias operaciones. Un procesador multifuncional puede realizar diferentes funciones, al mismo tiempo o en distintos tiempos, en base a interconectar diferentes conjuntos de niveles o etapas del procesador pipeline. La TI-ASC es un ejemplo de este tipo de procesadores, en donde se tienen 4 procesadores multifuncionales cada uno de los cuales se puede reconfigurar para realizar diferentes tipos de operaciones.

Estáticos y Dinámicos. Un procesador pipeline estático únicamente puede tener una configuracion funcional a la vez. Este tipo de procesadores pueden ser unifuncionales o multifuncionales. Es necesario que todas las instrucciones sean del mismo tipo para que se puedan ejecutar continuamente en un procesador estático. Si se trata de un procesador estático multifuncional, entonces no es recomendable que se cambie la función muy frecuentemente porque se reduciría el desempeño del sistema. Un procesador pipeline dinámico permite tener varias configuraciones funcionales trabajando simultaneamente; podemos decir que un procesador dinamico debe ser multifuncional. Para realizar una configuración dinámica, es necesario un control más elaborado. En general las computadoras están diseñadas con procesadores estáticos unifuncionales.

Escalares y vectoriales. Esta clasificacion esta basada en los tipos de instrucciones o datos. Un procesador pipeline escalar ejecuta una secuencia de operandos escalares dentro de un ciclo iterativo DO o FOR, es decir, cuando en un programa se tiene un ciclo DO, las instrucciones se almacenan en un buffer mientras que los operandos escalares requeridos se depositan en una memoria cache para alimentar continuamente el procesador pipeline; como ejemplo tenemos a la IBM 360/91 con la variante de que no utiliza memoria cache. Los procesadores pipeline de vectores estan diseñados para manejar instrucciones y operandos vectoriales, tambien se les conoce como procesadores de vectores (vector processors). Los procesadores pipeline de vectores son una extension de los escalares. El manejo de los vectores de operandos se realiza por medio de hardware y/o firmware (en los escalares el control es mediante software). Como ejemplo tenemos a la TI ASC, la STAR-100 , la CYBER 205, la Cray-1, la VP-200, la AP-120 B, la IBM 3838 y la MATP de Datawest.

2.3. Procesamiento Pipeline General.

Hasta este punto se ha estudiado el procesamiento pipeline lineal, sin retroalimentacion. En algunas aplicaciones es

necesario que la salida sea tomada como parte de la entrada para obtener una nueva salida, este tipo de procesamiento puede no tener un flujo lineal de informacion, es decir, dependiendo de la informacion procesada anteriormente se genera el estado presente. Tambien es posible tener una prealimentación (feed- fordward), por esto, el tiempo de retraso que tengan las entradas de retroalimentación y de prealimentación será crítico en el desempeño del sistema, es decir, una mala utilizacion de las retro y pre alimentaciones podrá destruir las ventajas de un procesamiento pipeline lineal. Sin embargo, cuando las retro y pre alimentaciones son bien utilizadas la eficiencia del procesador se ve sustancialmente mejorada. Generalmente, muchos de los procesadores aritmeticos, están diseñados como pipelines generales para implementar recursividad y multiplicacion.

Podemos decir que un procesamiento pipeline general es un procesamiento pipeline lineal mas pre y retro alimentaciones, en otras palabras, un procesamiento pipeline lineal solo tiene alimentación en cascada. Un ejemplo de un procesamiento general se muestra en la figura 2.7. Podemos ver que se tiene tanto prealimentacion como retroalimentación, los niveles se enumeraran como S_1, S_2,..., S_n, desde la entrada hasta la salida.

Definimos a la **prealimentación** como una conección que va del nivel S_i al S_j tal que $j>i+2$ y a la **retroalimentación** como una conección que va del nivel S_i al S_j tal que $j<=i$.

Tambien se tienen en la figura 2.7. dos tipos difernetes de salidas, A y B, esto nos dice que el procesamiento pipeline general es, ademas, multifun- cional. Para poder evaluar las diferentes funciones se utilizan multiplexores que en la figura se muestran como circulos tachados.

Es necesario definir tanto los niveles que utilizara cada función como el orden de su utilización. Para ello existen las tablas de reservado (reservation tables) que son una copia de las cartas de Grantt empleadas en investigacion de operaciones. En la figura 2.8. se presentan las tablas de reservado para las funciones A y B. Las columnas representan ciclos pipeline, mientras que los renglones corresponden a los niveles dentro del pipeline.

El **tiempo de evaluación de una función** es el número total de ciclos de reloj utilizados en la tabla de reservado, por lo tanto, cada función tendrá un tiempo de evaluacion especifico, aunque dos funciones diferentes pueden tener iguales tiempos de evaluación.

En resumen, una tabla de reservado representa el flujo de información dentro del pipeline desde la entrada de los datos hasta la salida de un resultado.

Podemos ver en la figura 2.8. que si el casillero (i,j) está marcado, quiere decir que el nivel S_i será utilizado j ciclos de reloj despues de que la función fue iniciada. En la figura 2.9. se analiza, detalladamente, el flujo de información dentro del pipeline utili- zando la tabla de reservado de la función A, dejandose al lector el análisis de la función B.

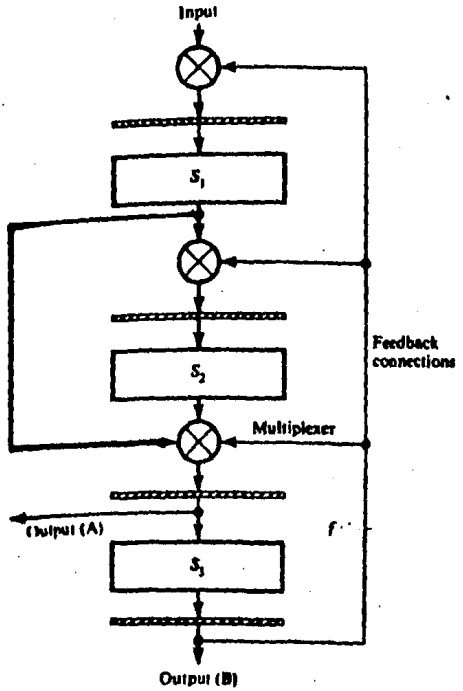
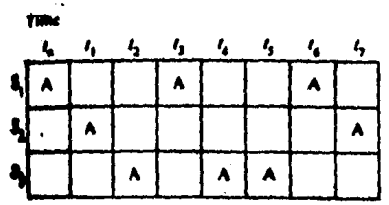
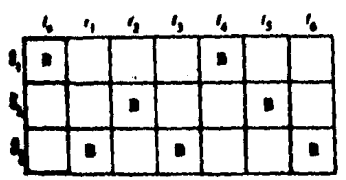


Figura 2.7. Ejemplo de un procesamiento pipeline general.[1]



a. Tabla de resevado para la función A.



b. Tabla de reservado para la funcion B.

Figura 2.8. Tablas de reservado para las funciones A y B.[1]

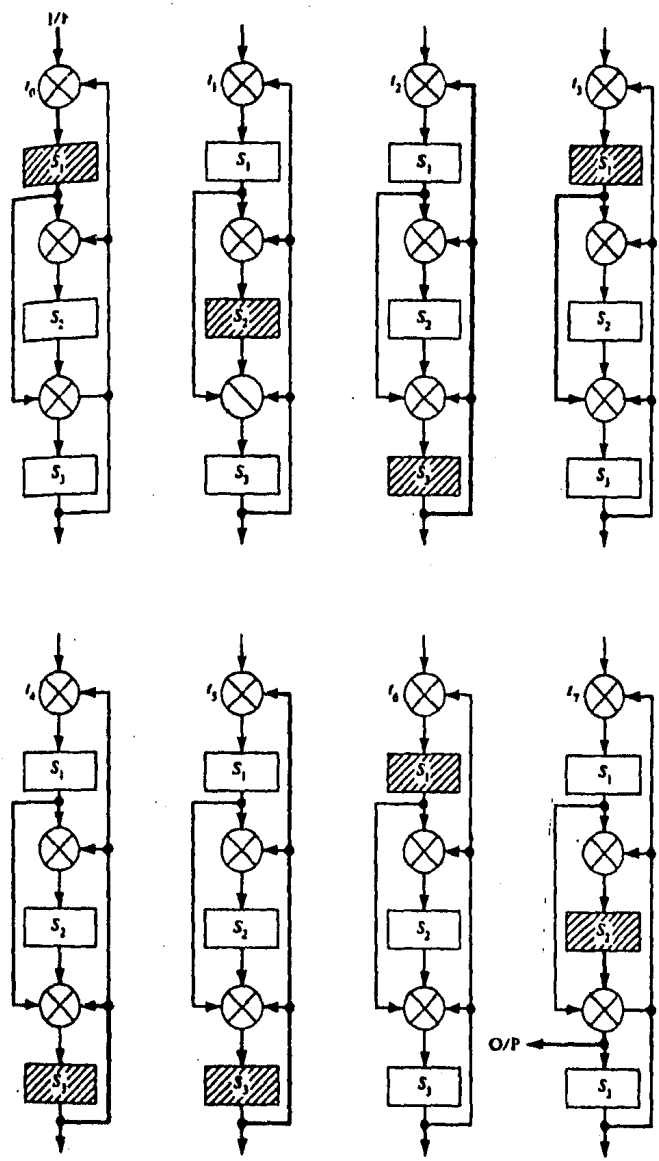


Figura 2.9. Analisis del flujo de informacion dentro del pipeline, utilizando la funcion A. [3]

Cuando se tiene un procesador pipeline unifuncional es posible marcar con una "X" en la tabla de reservado los casilleros utilizados por la función. Cuando el procesamiento pipeline es multifuncional, entonces es preciso usar diferentes simbolos para las distintas funciones como el caso de A y B.

Cuando se tiene un procesamiento pipeline estatico, unifuncional, su función puede quedar completamente definida utilizando una sola tabla de reservado. Para un procesamiento multifuncional es necesario tener varias tablas.

Cuando en una tabla de reservado hay una columna con más de un casillero marcado, quiere decir que al mismo tiempo se estarán utili- zando los niveles señalados. Tambien se pueden tener en un renglón varias marcas consecutivas, diciendo que un nivel tendrá una utilizacion prolongada. En un procesamineto pipeline general es factible tener más de una función, como son; uso paralelo de varios niveles y flujo de información no lineal.

2.4. Arreglos de Memoria Tipicos para Sistemas Pipeline.

Cuando se trabaja con sistemas pipeline es necesario que se tenga un acceso eficiente de las instrucciones almacenadas secuencialmente en memoria. Una forma para poder medir dicha eficiencia es tomar el numero promedio de palabras accesadas por segundo; a esta medida se le conoce como **ancho de banda de la memoria** (memory bandwidth).

Los factores que más afectan el ancho de banda son: la arquitectura del procesador, la configuración de la memoria y las caracteristicas del modulo de memoria. El ancho de banda de la memoria debe estar diseñado para satisfacer la demanda del procesador. En esta seccion se estudiarán las configuraciones de memoria para obtener un ancho de banda optimo al utilizar un sistema pipeline general.

La configuración con acceso S. Una de las configuraciones de memoria más simples utilizadas en procesadores pipeline, consiste en tomar las lineas de direccionamiento más significativas (n-m) y conectarlas a todos los modulos de memoria simultaneamente ($M=2^m$ modulos), con esto se obtienen M palabras consecutivas, una por modulo. Luego se utilizan los m bits menos significativos de las direcciones para accesar la palabra de un modulo en particular.

En la figura 2.10. se presenta esta configuracion, se puede observar que asociado a cada modulo de memoria se tiene un registro (latch), este registro es utilizado para almacenar la palabra que cada modulo proporcione, despues, mediante un multiplexor podran ser accesadas M palabras consecutivamente.

Esta configuracion se ha denominado tipo S debido a que el acceso es simultáneo en los M modulos de memoria, es muy utilizada para accesar vectores o para realizar el prefetch de instrucciones secuenciales en un procesador pipeline. Tambien son muy comunes estas configuraciones cuando es necesario accesar un bloque de información en un procesador pipeline con memoria cache.

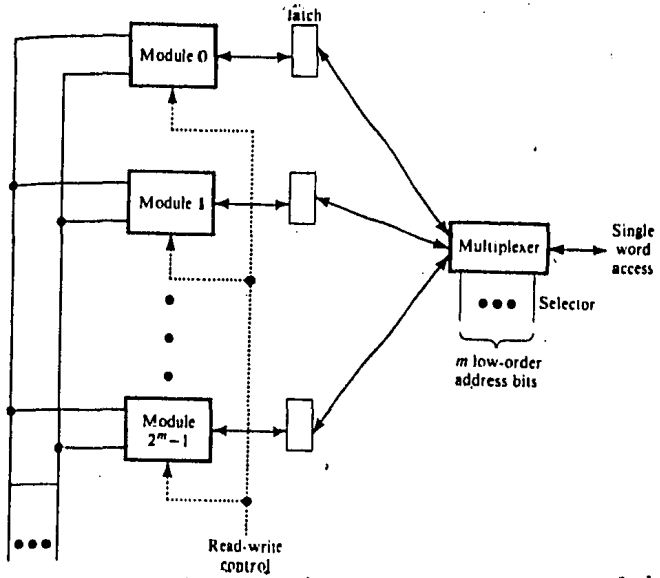


Figura 2.10. Arreglo de memoria con acceso S.[1]

En un arreglo de memoria tipico, el tiempo necesario para accesar k palabras consecutivas esta dado por kT_a, donde T_a es el tiempo de acceso de la memoria. Para un arreglo S, donde el retardo que tiene un registro es t y todos los registros utilizados son iguales, el tiempo necesario para accesar las mismas k palabras esta dado por $T_a + kt$, partiendo de el modulo i tal que $i + k <= M$, o bien sera $2T_a + (i + k - M)t$ para $i + k > M$, suponiendo en ambos casos que k es un entero entre 1 y M.

Lo anterior es debido a que $T_a >> t$ para que no se vea afectado el ancho de banda de la memoria, entonces cuando $i + k > M$ se necesita un primer T_a para cargar en los registros las primeras (M-i) palabras. Después es necesario un segundo T_a para cargar en los registros las restantes (i+k-M), palabras. Durante el segundo T_a es posible barrer con el MUX las (M-i) palabras iniciales puesto que $T_a > t$. Por lo tanto, cuando $i + k > M$ el retardo producido en los registros sera $t(i + k - M)$ y se necesitarán dos tiempos de acceso a memoria. En la figura 2.11. se presenta un diagrama donde se puede observar el traslape entre el segundo tiempo de acceso y el retardo en los registros.

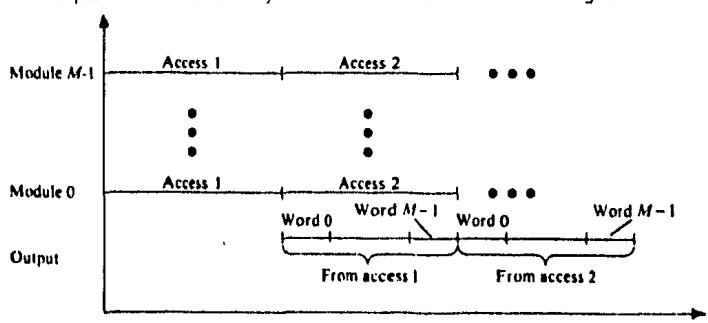


Figura 2.11. Diagrama de tiempos para la configuración con acceso S.[1]

Cuando se realizan direccionamientos no secuenciales, entonces el sistema se deteriora rápidamente.

Otro arreglo de memoria que es muy utilizado en aplicaciones de procesamiento pipeline se conoce como **acceso C.** Cuando se inicia una operacione de memoria dentro de un modulo, provoca que el banco de memoria se active por un periodo de t_a segundos y que el modulo se active por t_c segundos. Si suponemos que t_a es mucho menor que t_c entonces el modulo que se acceso utilizará el banco de memoria por un periodo mucho menor que el ciclo de memoria. Debido a lo anterior, el banco de memoria puede ser utilizado por más de un modulo a la vez, mejorando el aprovechamiento del banco y disminuyendo su costo. A esta configuracion se le conoce como acceso C debido a que los modulos se accesan concurrentemente. En la figura 2.12. se presenta un diagrama de esta arquitectura.

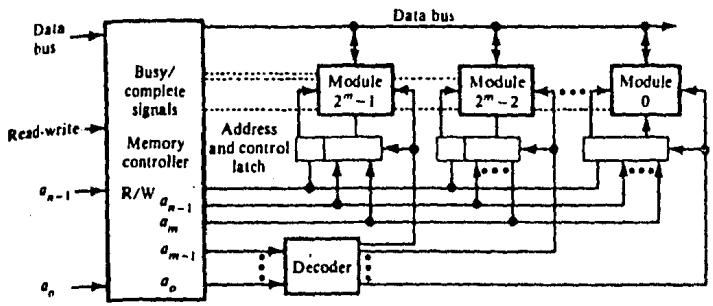


Figura 2.12. Arreglo de memoria con acceso C. [1]

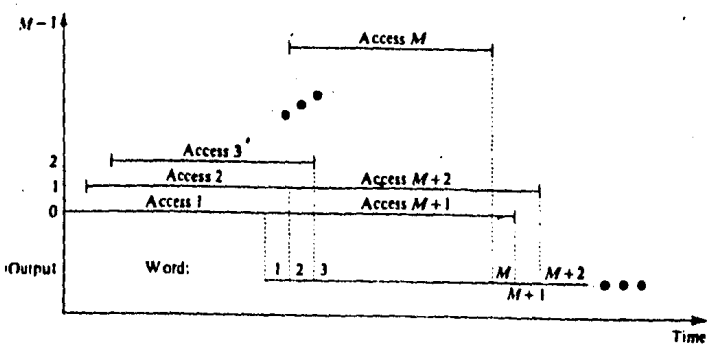


Figura 2.13. Diagrama de tiempos.[1]

Los m bits de direccionamiento menos significativos entran a un decodificador que se encarga de habilitar el registro (latch)

del modulo de memoria deseado. En dicho registro se almacenan los m-n bits restantes, de esta forma el modulo de memoria tiene fijas las direcciones durante todo el T_a, por esto se puede estar accesando concurrentemente un distinto modulo sin necesidad de esperar a que el primero tenga listo el dato. En la figura 2.13 . se presenta un diagrama de tiempos en donde se accesan k palabras consecutivas. El tiempo necesario es igual a $T_a + kt$, donde T_a es el tiempo de acceso de la memoria y t es el retraso en el banco, es decir, la suma de los tiempos utilizados en el decodificador y el registro. Como se dijo anteriormente $t_a<t_a$, para que se cumpla dicha desigualdad se puede proponer a $t_a = t = T_a/M$; donde M es el numero de modulos en el banco. El controlador de memoria sirve para evitar que un modulo sea accesado cuando aún esta ocupado, ademas inicia la rutina de servicio cuando un modulo terminó un ciclo completo.

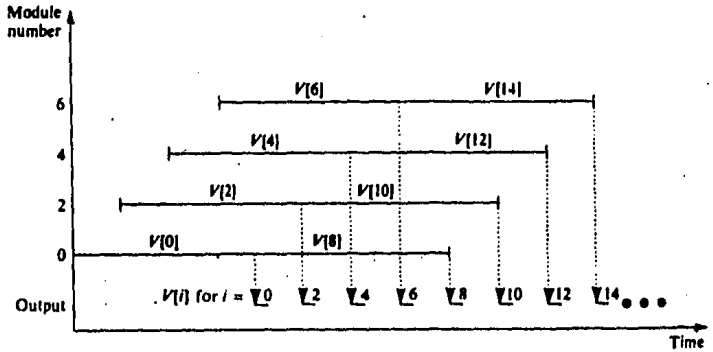
Una de las principales aplicaciones de este tipo de configuraciones es en el acceso de vectores. Si consideramos un vector de S elementos V(0,S-1) y suponemos que queremos accesar un elemento si y otro no, es decir, que tenemos una distancia de 2, y ademas el elemento V[i] esta almacenado en el modulo i mod M para $0<=i<=S-1$, entonces con M=8, en la figura 2.14.a se muestra el diagrama de tiempos.

Despues de haber transcurrido el primer tiempo de acceso T_a, se se tendra un resultado cada 2t segundos, para $t=T_a/M$. Esto es debido a que la distancia es 2 y el numero de modulos es 8, es decir, primero se accesa el modulo 0 y no podera ser utilizado hasta despues de T_a segundos, pasado un tiempo t se accesa el modulo 2, pasados 2t segundos se accesa el modulo 4 y despues de 3t se accesa el modulo 6. En el tiempo igual a 4t se intenta accesar el modulo 1, sin embargo solo han transcurrido $4t=T_a/2$ (para este caso), por lo tanto es necesario esperar a que transcurran 4t mas de tal forma que $8t=T_a$, en este tiempo se accesa de nuevo el modulo 1. En resumen, con una distancia igual a 2 y 8 modulos de memoria, se obtienen, despues del primer T_a, cuatro palabras una cada t y despues hay 4t donde no se obtiene ninguna palabra, en promedio podemos decir que obtenemos un resultado cada 2t segundos.

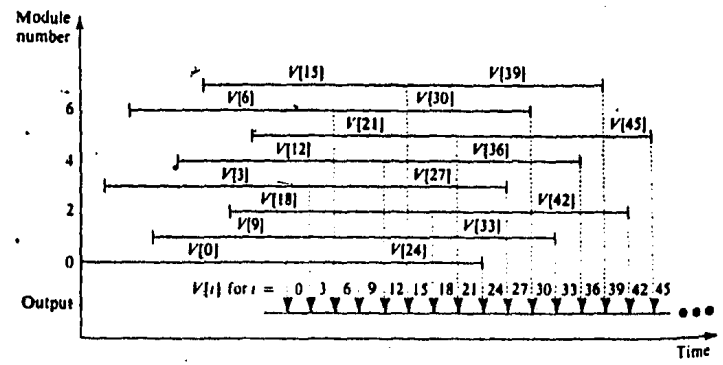
Ahora bien, si se incrementa la distancia a 3, después de hacer un análisis similar llegamos a la conlusion de que se obtiene un resultado cada t segundos despues de realizado el primer tiempo de acceso T_a, como se muestra en la figura 2.14.b.

En general podemos decir que con una distancia d y M modulos dispuestos en configuración con accesso C, de tal forma que M y d sean relativamente primos, entonces los elementos podrán ser accesados a una velocidad máxima de T_a/M segundos por palabra. Resulta claro que un arreglo de memoria con acceso S tendra un desempeño menor para secuencias donde d>1, para el cado en que d=1 no existe diferencia entre las configuraciones.

Una tercera configuracion de memoria es conocida como **acceso C/S**, donde se tiene una combinación de las dos configuraciones anteriores, en la configuracion de acceso C/S, los modulos de memoria se organizan en arreglos bidimensionales. Este tipo de esquema es efectivo para procesadores pipeline multiples.



a. Distancia 2.



b. Distancia 3.

Figura 2.14. Diagramas de tiempos para una configuración con acceso C, con 8 módulos y distancia de 2 y 3 respectivamente. [1]

2.5. Diseños de sistemas pipeline

En esta sección se pretende dar un panorama general de los métodos utilizados para diseñar sistemas pipeline. Esta dividida en tres partes que analizan por separado a los sistemas pipeline según la clasificación de Händler, es decir, diseño de pipeline de instrucción, diseño de pipeline aritmético y diseño de pipeline de procesador.

2.5.1 Diseño de pipeline de instrucción.

En los equipos de cómputo actuales generalmente el procesador central esta basado en la filosofía pipeline. Un

ejemplo tipico de este tipo de sistemas es el IBM 360/91 que contiene un alto grado de pipeline tanto en la unidad de procesamiento de instruccion como en la ejecucion de instrucciones. Esta es una computadora de 32 bits disenada especialmente para aplicaciones cientificas que necesitan realizar operaciones de punto fijo y de punto flotante. La computadora permite la realizacion de operaciones aritmeticas paralelas en cualquiera de los dos formatos debido a que contiene 4 unidades funcionales con procesamiento pipeline.

En la figura 2.15. se presenta un diagrama de bloques del procesador central de la IBM 360/91. Las 4 unidades principales son: unidad de control de memoria principal, unidad de ejecucion de punto fijo, unidad de ejecucion de punto flotante y unidad de instruccion.

La unidad de instruccion tiene procesamiento pipeline con un periodo de reloj de 60 ns. Esta CPU esta diseñada para procesar instrucciones a una tasa promedio de una instruccion por ciclo de reloj, por esto las dos unidades de ejecucion (punto fijo y punto flotante) deberan ser capaces de soportar esta tasa. En la unidad de control de memoria principal se supervisa el intercambio de informacion entre la CPU y las funciones de la unidad de instruccion en la memoria principal. Estas funciones pueden ser fetch, decodificacion salida a una unidad de ejecucion y cálculo de la direccion de un operando. Las unidades de ejecución son responsables del cálculo de operaciones de punto fijo y flotante que sean necesarias durante la fase de ejecucion.

Desde que se realiza un acceso a memoria hasta que se realiza la decodificacion y la ejecución, la CPU esta construida con arquitectura pipeline.

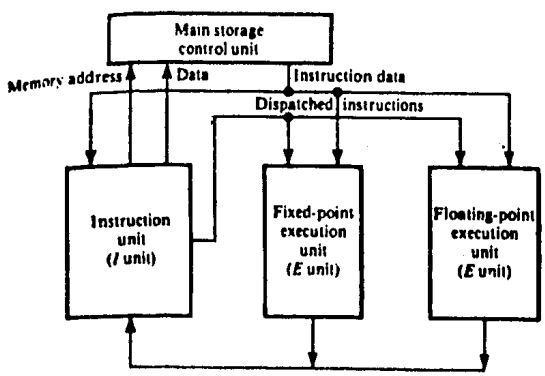


Figura 2.16. Unidad de Procesamiento Central de la IBM 360/91.[1]

La figura 2.16. muestra el traslape en la ejecucion de instrucciones almacenadas en memoria secuencialmente. Se pretende que cada una de las unidades tenga procesamiento pipeline altamente eficiente. Las areas sombreadas corresponden a las funciones de las unidades y las lineas entre ellas representan el retraso debido a los accesos a memoria. Obviamente un acceso a

memoria es mas tardado que el tiempo empleado en realizar la funcion. Despues de que el pipeline se lleno, los resultados se obtendran con una tasa de 1 cada 60 ns.

Se pretende con este ejemplo visualizar la utilidad de diseñar unidades de instruccion con procesamiento pipeline. En la figura 2.17. se muestra un diagrama de tiempos suponiendo que la unidad de instruccion no tiene procesamiento pipeline.

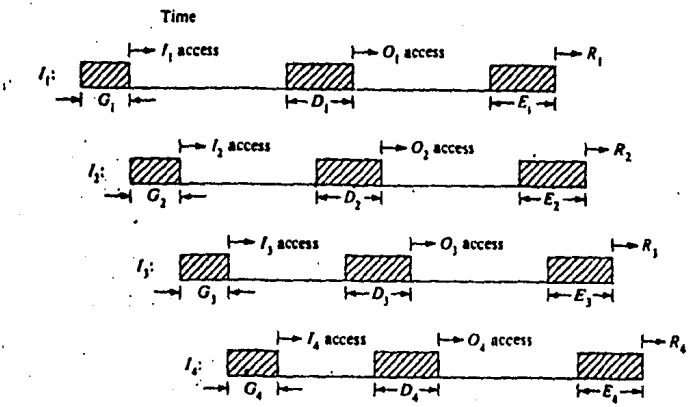


Figura 2.16. Traslape en la ejecución de instrucciones utilizado unidades de intruccion con procesamiento pipeline.[1]

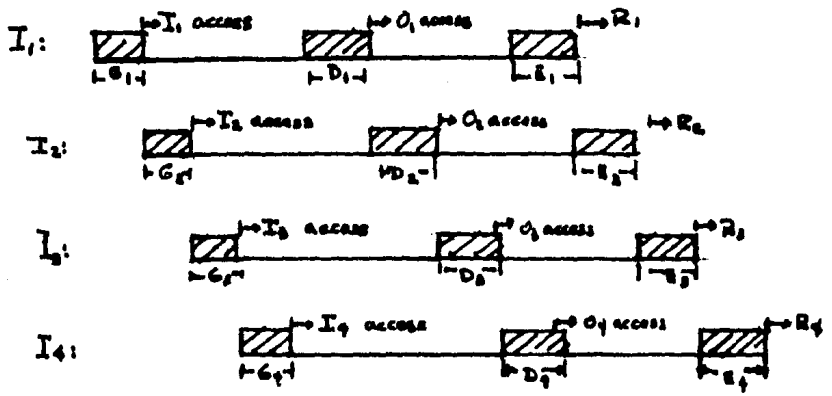


Figura 2.17. Traslape en la ejecucion de instrucciones utilizando unidades de instrucción sin procesamiento pipeline.

2.5.2 Diseño de pipeline aritmético.

En esta parte se tratará unicamente con procesamientos pipeline estáticos y unifuncionales. Se diseñará un pipeline aritmético que podrá realizar multiplicaciones.

Tradicionalmente la multiplicación de dos numeros de punto fijo se realiza mediante la utilización repetida de la operación suma-corrimiento. Esta es la forma más común debido a que en general cualquier ALU tiene incluidas las funciones de ADD y SHIFT. El numero de operaciones de suma y corrimiento necesarias en la multiplicación de dos números resultará ser proporcional al tamaño de los operandos, por lo tanto, este metodo es demasiado lento. Si examinamos el proceso de multiplicacion en la figura 2.18, podemos ver que multiplicar es equivalente a la suma de varias copias de los operandos pero corridos.

$$
\begin{array}{cccccccccccc}
 & & & & & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & = A \\
 & & & & \times) & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & = B \\
\hline
 & & & & & a_5b_0 & a_4b_0 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & = W_1 \\
 & & & & a_5b_1 & a_4b_1 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & = W_2 \\
 & & & a_5b_2 & a_4b_2 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & = W_3 \\
 & & a_5b_3 & a_4b_3 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & & = W_4 \\
 & a_5b_4 & a_4b_4 & a_3b_4 & a_2b_4 & a_1b_4 & a_0b_4 & & & & & = W_5 \\
+) & a_5b_5 & a_4b_5 & a_3b_5 & a_2b_5 & a_1b_5 & a_0b_5 & & & & & = W_6 \\
\hline
P_{11} & P_{10} & P_9 & P_8 & P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0 & = A \times B = P
\end{array}
$$

Figura 2.18. Proceso de multiplicacion de dos numeros. [1]

La suma de varios números se puede realizar mediante un arbol. El sumador convencional donde se propaga el prestamo (carry) conocido como CPA (Carry Propagation Adder), suma dos numeros, A y B, para obtener un resultado, es decir A + B. Un sumador donde el prestamo se guarda, CSA (Carry Save Adder), recibe tres entradas, A, B y C, dando como resultado la suma bit a bit de los tres operandos, es decir $S_i = A_i + B_i + D_i$ donde LSB <= i <= MSB, el prestamo de esta suma se guardará en el elemento C_{i+1} donde $C_0 = 0$. Al terminar podemos obtener el resultado realizando la operacion de OR exclusivo entre C y S, esto es:

R = A + B + D = C EX-OR S .

$$
\begin{array}{lcccccccc}
 & A = & 1 & 1 & 1 & 1 & 0 & 1 & \\
 & B = & 0 & 1 & 0 & 1 & 1 & 0 & \\
+) & D = & 1 & 1 & 0 & 1 & 1 & 1 & \\
\hline
 & C = & 1 & 1 & 0 & 1 & 1 & 1 & \\
\text{EX-OR)} & S = & 0 & 1 & 1 & 1 & 0 & 0 & \\
\hline
A+B+D & = & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
\text{C EX-OR S} & & & & & & & &
\end{array}
$$

Para implementar un CPA podemos utilizar sumadores completos en cascada donde el préstamo de un nivel más bajo se conecta a un nivel superior. Un CSA puede implementerse con un conjunto de sumadores completos donde las líneas de entrada del pestamo se utilizarán para meter el operando y las líneas de salida del préstamo serán el vector C. En otras palabras, las líneas de préstamo de salida en un CSA no se interconectan. En resumen podemos ver que un CDA es un convertidor de 2 números a 1 número y un CSA es un convertidor de 3 a 2 números.

Ya que conocemos el funcionamiento del CSA podemos emplearlo para realizar la suma múltiple. Esto será en si, una multiplicacion con procesamiento pipeline, y el circuito que lo realice será un procesador unifuncional con procesamiento pipeline aritmético.

El diseño mostrado en la figura 2.19, realiza la multiplicacion de dos numeros de 6 bits. Se tienen cinco niveles pipeline, en el primer nivel se generan los 6 X 6 = 36 primeros terminos, es decir; $\{a_i b_j \ / \ 0<i<5 \ , \ 0<j<5 \ \}$ que forman los 6 multiplicandos $\{W_i / 1<i<6\}$ de la figura 2.18. Estos seis números entran en dos CSA en el segundo nivel. En general, es necesario interconectar 4 CSA para obtener dos números de los 6 iniciales, estos dos números serán un vector de suma A y uno de préstamo C. En el último nivel se utiliza un CPA que suma A con C para producir el producto final P = A X B. Si colocamos registros entre cada nivel y los habilitamos todos al mismo tiempo con una señal Ck entonces podremos estar realizando hasta cinco multiplicaciones concurrentemente una vez que todos los niveles estén ocupados, ademas se obtendrá un resultado cada vez que se presente la señal Ck.

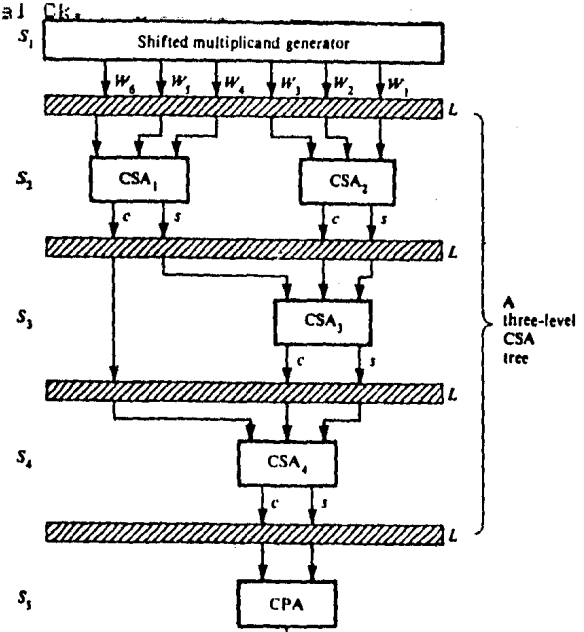


Figura 2.19. Multiplicador pipeline tipo CSA. [1]

2.5.3 Diseño de pipeline de procesador.

En esta sección se estudian los problemas básicos que se presentan en el diseño de pipeline de procesador, debe quedar claro que no se pretende hacer un estudio exhaustivo de todos los problemas que se pueden tener al diseñar un pipeline de procesador, sino solo los más importantes.

2.5.3.1 Interrupciones y saltos.

Desde el punto de vista de ejecutar instucciones concurrentemente, podemos clasificar a los distintos tipos de instrucciones en cuatro grandes grupos, como se muestra en la tabla 2.1. Se puede ver que el tipo de instrucción aritmetica o de carga ocupa el 60% de un programa tipico. El tipo de instrucción llamada únicamente salto corresponde a un salto incondicional. Para los saltos condicionales tenemos dos tipos. El tipo "si", necesita calcular una nueva dirección (a donde debe transferir el control), mientras que el tipo "no" ejecuta la siguiente instrucción del programa. Las instrucciones aritmeticas y de carga no alteran la secuencia del programa. Las instrucciones de salto, que tipicamente representan un 25% del programa, pueden alterar el PC para saltar a una dirección distinta a la siguiente. Enfatizamos en la alteración del PC debido a un salto porque puede tener efectos negativos en el desempeño del procesamiento pipeline. Cuando una instrucción de interrupción se presenta mientras la instruccion I se está ejecutando, entonces la ejecución de la instrucción I+1 debe posponerse hasta que la interrupción se termine.

Segment function / Instruction type and mix rate	Arithmetic/load type, 60%	Store type, 15%	Branch type, 5%	Conditional branch type: Yes, 12%	Conditional branch type: No, 8%
Instruction fetch	6	6	6	6	6
Decode	2	2	2	2	2
Condition test				1	1
Operand address calculation	2	2	2	2	2
Operand fetch(es)	6-12				
Arithmetic logic execution	4-8				
Store result		6			
Update PC and flags	1	1	1	1	1
Total pipeline cycles	21-31	17	11	12	12

Tabla 2.1. Porcentajes de la utilizacion de los distintos tipos de instrucciones empleados en un programa tipico.[1]

El manejo de las interrupciones en un procesador pipeline debe realizarse con mucha precaución de tal forma que aquellas instrucciones que se encuentran dentro del procesador, terminen su ejecucion para evitar que sean destruidas por la rutina de servicio.

En la computadora Cray-1, se tiene un sistema de intercambio para resolver el problema de las interrupciones. En esencia, cuando se presenta una interrupción, esta computadora almacena el estado actual del procesador y carga el nuevo estado, en otras palabras, cuando hay una interrupción se almacenan ocho registros escalares, ocho registros de direcciones, el PC y las banderas. Sin embargo, algo similar debe realizarse en los procesadores normales.

Para explicar el efecto que tiene un salto en un procesador pipeline, se hará uso de un pipeline lineal con n=5 niveles: fetch de instrucción, decodificación, fetch de operando, ejecución y almacenamiento de resultados. Como se muestra en la figura 2.20, este procesador pipeline ejecuta una cadena de instrucciones continuamente. La figura 2.21 muestra el diagrama de tiempo cuando no se presentan instrucciones de salto, este es el modelo que estudiamos en la sección 1, como se dijo después de que el pipeline está lleno se tendrá un resultado en cada ciclo de reloj. Ahora supongamos que en la ejecución del programa, se presenta un salto,esto ocasiona que el PC sea cargado con la nueva dirección provocando que todas las instrucciones que fueron anticipadas resulten inservibles, generándose un retraso de n-1 ciclos como se puede ver en la figura 2.22. Después de que el salto se terminó dee ejecutar, se tendrá nuevamente un resultado por cada ciclo pipelin hasta que se presente otro salto. Es obvio que un procesador pipeline tendrá mayor tiempo de ejecución cuando mayor sea el número de saltos dentro del programa. El peor de los casos será tener saltos uno tras de otro por lo que se hace presente el uso de técnicas de programación estructurada y análisis de la eficiencia de algoritmos para obtener el mayor desempeño de un procesador pipeline.

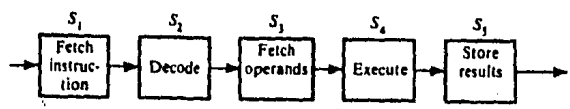


Figura 2.20. Procesador pipeline lineal con 5 niveles.[1]

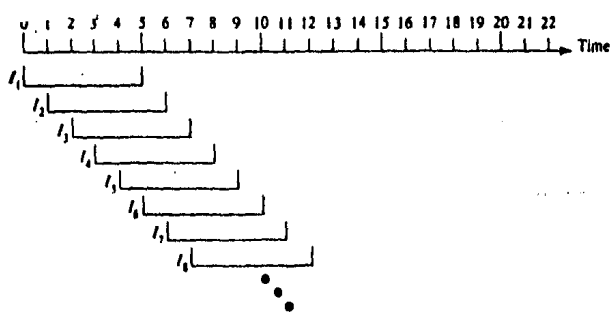


Figura 2.21. Diagrama de tiempos sin instrucciones de saltos.[1]

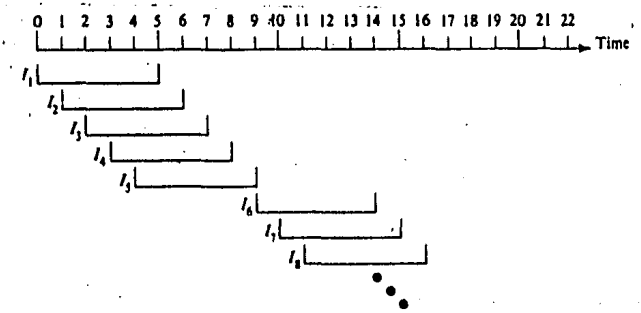


Figura 2.22. Diagrama de tiempos usando instrucciones de salto.[1]

Una estimación del efecto que producen los saltos en procesador pipeline de n niveles se da a continuación: Supongamos que un ciclo de instruccion es igual a 5 ciclos pipeline. Sea p la probabilidad de que exista un salto condicional dentro del programa en ejecución y demosle un peso de 20%. Además, cuando se presente una condición se tiene 12% de probabilidad de que sea verdadera y 8% de que no lo sea (ver tabla 2.1). Entonces la probabilidad de que sea verdadera dado que se presentó un salto condicional será:

$$P(V/C) = q = 12/20 = 60\%$$

Supongamos que tenemos m instrucciones esperando ser ejecutadas por el procesador pipeline. Entonces el numero de instrucciones que ocasionan saltos verdaderos sera igual a mxpxq. Como se dijo antes, cada vez que se presenta un salto se requieren n-1 ciclos extras, ademas el desempeño de un sistema pipeline lineal sin saltos es n+m-1 (esto se demuestra en la sección 1). Entonces el numero total de ciclos pipeline requeridos para procesar m instrucciones será igual a:

$$(n+m-1) + (mxpxq)(n-1)$$

y el total de ciclos de instrucción necesarios para procesar n palabras estara dado por:

$$\frac{(m+n-1) + (mxpxq)(n-1)}{n}$$

Cuando el número de instrucciones ejecutadas es muy elevado, entonces el desempeño del sistema se puede medir como el promedio de instrucciones ejecutadas por ciclo de instrucción, es decir:

$$\lim_{m\to\infty} \frac{m}{(n+m-1)/n+mpq(n-1)n} = \lim_{m\to\infty} \frac{mxn}{(n+m-1)+mpq(n-1)}$$

$$= \frac{n}{1 + pq(n-1)}$$

Cuando no existen saltos, p=0, la ecuación anterior se reduce a un resultado por ciclo de reloj, que es el caso ideal. En general es imposible prescindir de saltos por lo que el desempeño siempre es algo menor. Por ejemplo, para n=5, p=20% y q=60%, tendremos un desempeño de 3.24 instrucciones por ciclo de instrucción (un ciclo de instrucción es igual a 5 ciclos pipeline), este desempeño es 1.76 veces menor que el ideal. En otras palabras, se desperdicia en promedio 35.2% de los ciclos pipeline debido a los saltos producidos en el programa. Podemos decir que si los saltos empleados en la programación no son los estrictamente necesarios estamos deteriorando el desempeño de nuestro procesador pipeline (hardware) desde el programa (software).

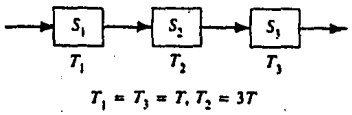
Para resolver los problemas que ocasionan los saltos existen dos tecnicas importantes conocidas como: prefetch de instruccion y manejo de saltos, que pueden ser estudiadas en [1] y [3] con mayor detalle.

2.5.3.2. Cuellos de botella.

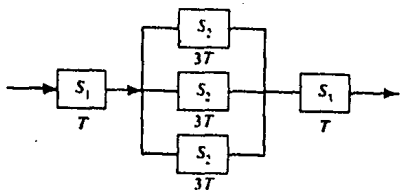
Generalmente las velocidades de procesamiento de los niveles pipeline son desiguales. Consideremos el sistema pipeline de la figura 2.23.a con tres niveles, cada nivel tendra un retraso T_1, T_2 y T_3 respectivamente. Si suponemos que $T_1=T_3=T$ y $T_2=3T$, resulta obvio que el nivel 2 es el cuello de botella del sistema. El desempeño del sistema es inversamente proporcional al cuello de botella, por lo que es deseable eliminarlo dado que ocasiona un congestionamiento.

Un metodo muy utilizado por su simplicidad es el de subdividir el cuello de botella. Lo óptimo es subdividir en niveles que tengan un retraso T cada uno, sin embargo, existen tareas que no son divisibles por lo que es imposible alcanzar el óptimo. Podemos lograr un suboptimo dividiendo en tantos niveles con retraso T como sea posible y dejar un nivel con retrso mayor a T, esto se muestra en la figura 2.23.b. En ambos casos el desempeño se ha mejorado. Sin embargo, si el cuello de botella no es subdivisible, podemos mejorar el desempeño poniendo duplicados del cuello de botella en paralelo como se muestra en la figura 2.23.c. Esta solución resulta ser compleja debido a que necesita tener un control que sincronice las tareas en los niveles paralelos, tambien resulta mas costoso.

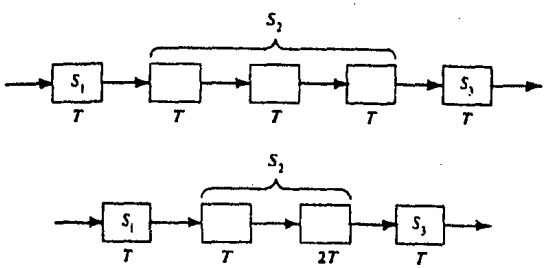
Existen otros metodos para resolver los problemas como son: almacenamiento de instrucciones e información, y estructuras eficientes de alambrado. Estos metodos pueden ser estudiados en [1] y [3].



a. El segmento 2 es el cuello de botella.



c. Replicas del segmento 3.



b. Subdivision del segmento 2.

Figura 2.23. Metodos para resolver los cuellos de botella.[1]

2.6. Procesamiento de vectores.

En esta seccion se explican los conceptos basicos necesarios para procesar arreglos vectoriales asi como los métodos existentes para su implementacion. Inicialmente se describen las caracteristicas del procesamiento de vectores, despues se plantea un modelo de supercomputadora que explota un maximo de concurrencia en el procesamiento de vectores y finalmente se describen tres métodos para procesar arreglos vectoriales utilizando arquitecturas pipeline.

2.6.1 Características generales del procesamiento de vectores.

Un vector es un conjunto ordenado de n elementos, donde n es la longitud del vector. Cada elemento del vector es un escalar que puede ser de punto fijo , punto flotante, entero, caracter o booleano. Existen cuatro tipos de funciones con vectores:

f1: V -> V
f2: V -> S
f3: VxV -> V
f4: VxS -> V

Donde V y S son operandos vectoriales y escalares respectivamente. Por ejemplo, la operacion vectorial raiz cuadrada es del tipo f1, a cada elemento de un vector V_1 se le extrae la raiz cuadrada y se guarda en el vector de resultado V_2, por lo que un vector V_1 de entrada mapea mediante la operación raiz cuadrada a un vector V_2 de salida. De igual forma vemos que la suma de los elementos de un vector es del tipo f2 y la suma de dos vectores pertenece al tipo f3. En la figura 2.24 se presentan estos cuatro tipos de funciones implementadas con procesamiento pipeline.

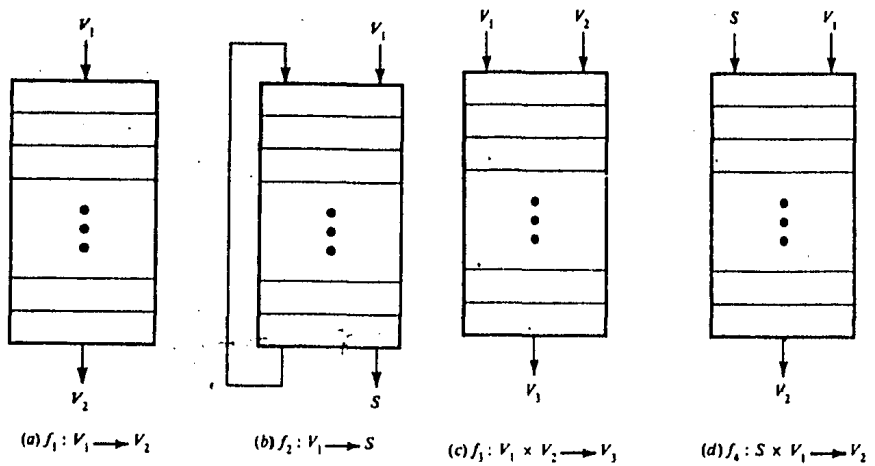


Figura 2.24. Cuatro instrucciones vectoriales implementeadas con procesadores pipeline.[11]

En general, las operaciones que pueden ser implementadas con procesamiento pipeline deben cumplir con los siguientes requisitos:

a. Tienen procesos que se utilizan muchas veces y cada proceso puede ser subdividida.
b. Se alimentan operandos secuencialmente y se requiere la menor cantidad de registros posible.
c. Las operaciones ejecutadas en los distintos procesadores pueden compartir recursos como memoria y alambrados.

Estas caracteristicas nos explican por si mismas porque la mayoria de los procesadores de vectores tienen procesamiento pipeline. Es decir, una instruccion vectorial requiere que se

realice la misma operacion en los diferentes elementos del vector, repetidamente. Los procesadores pipeline de vectores tienen mejor desempeño cuando se procesan vectores con gran número de elementos, debido a que se necesita un tiempo t para llenar todo el procesador y un tiempo t_ para cargar el vector.

Podemos clasificar a los procesadores de vectores en dos grupos de acuerdo a donde depositan sus operandos despues de procesarlos. El primer grupo es el de **memoria a memoria**, en este esquema, tanto los operandos como los resultados intermedios y finales son depositados directamente en la memoria principal. Algunos ejemplos de estos procesadores son la TI-ASC, la CDC-STAR-100 y la Cyber-205 .

El otro grupo es el de **registro a registro**, en esta arquitectura los operandos y los resultados se depositan indirectamente en la memoria principal mediante el uso de registros. Como ejemplo tenemos la Cray-1 y la VP-200.

Mediante el siguiente ejemplo se pretende comparar el procesamiento vectorial y el procesamiento escalar. Supongamos que tenemos el siguiente programa escrito en FORTRAN.

```
      DO 100 I=1,N
         A(I)=B(I)+C(I)
         B(I)=2*A(I+1)
100 CONTINUE
```

Para implementarlo como una secuencia de operaciones escalares debemos tener el siguiente programa.

```
      I=1
10    READ B(I)
      READ C(I)
      ADD B(I) + C(I)
      STORE A(I) <- B(I)+C(I)
      READ A(I+1)
      MULT 2*A(I+1)
      STORE B(I) <- 2*A(I+1)
      INC I<- I+1
      IF I<=N GO TO 10
      STOP
```

Implementando el mismo algoritmo pero con instrucciones vectoriales tendremos:

```
VEQ     TEMP(1:N)= A(2:N+1)
VADD    A(1:N)= B(1:N) + C(1:N)
VSMUL   B(1:N)= 2*TEMP(1:N)
```

Donde VEQ significa igualar un vector, VADD significa suma de vectores y VSMUL significa multiplicacion de un vector por un escalar.

En la actualidad, el paralelismo logrado en los algoritmos se pierde cuando se expresa en un lenguaje de alto nivel. Para lograr tener procesamiento vectorial en la maquina, es necesario que se tenga un compilador inteligente para recuperar el

paralelismo mediante la vectorización. Al proceso de reemplazar un bloque de código secuencial por instrucciones vectoriales se le llama **vectorización** y al programa que realiza este proceso se le conoce como compilador vectorizador (vectorizing compiler).

Por ejemplo, supongamos que tenemos el siguiente programa en FORTRAN.

```
      DO 10 I=4, 100
         .
         .
         .
      C(I)= A(I) + B(I-3)
         .
         .
         .
10    CONTINUE
```

Al compilarlo con un compilador vectorizador el código equivalente ejecutable en un procesador de vectores sería el siguiente.

```
VECT_BEGIN
A,C: VECTOR(4..100);
B: VECTOR(1..97);
C= A+B
VECT_END.
```

Otra forma de lograr procesamiento vectorial es incluir las instrucciones vectoriales dentro del lenguaje de Alto nivel.

El lenguaje FORTRAN ha sido mejorado para que pueda procesar operaciones vectoriales mediante la inclusión de primitivas especiales como suma de vectores y multiplicación de vectores [3]. Como ejemplos de estas extensiones tenemos al LRL-TRAN de Lawrence Livermore Laboratory, diseñado para la computadora STAR-100, también está el ASC-FORTRAN de Texsas Instruments, diseñado para su computadora ASC.

2.6.2 Arquitectura de un procesador pipeline de vectores.

En esta sección se presenta una arquitectura que explota el máximo de concurrencia en el procesamiento de vectores. En la figura 2.25. se presenta el diagrama de bloques de una supercomputadora con varios pipelines capaz de procesar vectores. Esta estructura es una generalización de los procesadores de vectores modernos.

Los operandos pueden ser tanto escalares como vectoriales, por lo que la unidad de procesamiento de instrucción (IPU) deberá realizar el fetch y la decodificación tanto de instrucciones escalares como de vectoriales. Las instrucciones escalares serán ejecutadas en un procesador escalar. Este procesador estaa a su vez constituido por varios procesadores pipeline escalares.

Cuando la IPU reconoce una instrucción vectorial, la manda al controlador de instrucciones vectoriales para que se encargue de su ejecución. Las funciones de este controlador incluyen la decodificación , el cálculo de la dirección efectiva del operando, alerta al controlador de acceso de vectores y monitoreo de la

ejecucion de las instrucciones vectoriales.

El contolador de acceso de vectores es responsable del fetch de los operandos vectoriales. Los registros vectoriales se utilizan para acercar la velocidad de las memorias a la velocidad del procesador. En este ejemplo se considera que el controlador de instrucciones vectoriales es muy capaz y puede realizar una particion de la tarea vectorial para procesarla en distintos procesadores pipeline.

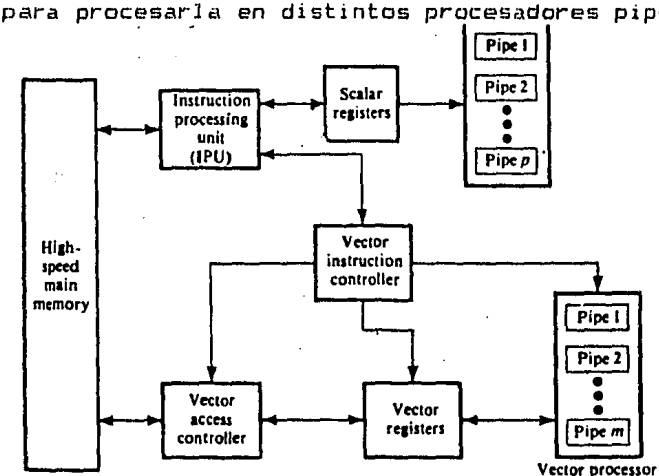


Figura 2.25. Arquitectura de un procesador de vectores que utiliza varios pipelines.[1]

2.6.3 Métodos de procesamiento pipeline de vectores.

Generalmente los computos de arreglos vectoriales están involucrados con el procesamiento de grandes cantidades de informacion. Podemos llegar a establecer una clasificacion del procesamiento de vectores si tomamos en cuenta la forma de como estos cálculos se llevan a cabo. Se han generado tres grandes grupos en el procesamiento vectoriale, en esta parte se da una breve explicación de cada uno de ellos y se reliza una comparacion entre ellos.

1. Procesamiento horizontal.

En este metodo, los computos se relizan horizontalmente y de izquierda a derecha (en forma de renglones). Los componentes de un vector $Y_m = Z_{m1} + Z_{m2} + \ldots + Z_{mn}$ se calculan en orden secuancial Y_1, $Y_2 \ldots Y_m$ y cada renglón (Y_m) debe terminerse de calcular antes de empezar con el siguiente. Este método generalmente es utilizado en procesadores pipeline escalares.

2. Procesamiento vertical.

En este metodo, los cómputos se realizan verticalmente y de arriba hacia abajo (en forma de columna). Se van calculando sumas parciales entre las columnas, por ejemplo, $Z_{m1} + Z_{m2}$. Al terminar con este calculo se toma otra columna y se suma al resultado previo. Este proceso se repite hasta que no quedan columnas, en este momento se obtiene en el vector Y_m el cómputo final. Este método se aplicó en el procesador de vectores de la computadora STAR-100.

3. Procesamiento cíclico.

El procesamiento cíclico (vector looping) es una combinación del procesamiento horizontal y el procesamiento vertical, trabajando por medio de bloques. Esencialmente, el procesamiento ciclico divide los Y_m vectores en grupos, digamos de 5 vectores cada uno, después procesa cada grupo verticalmente. Al terminer con el primer grupo procesará el siguiente bloque y así hasta terminar con los m vectores.

En general, el procesamiento horizontal es adecuado para procesadores escalares pero no es bueno para procesamiento paralelo de vectores. Los procesamientos vertical y ciclico son muyn empleados en el procesamiento de vectores.

El procesamiento vertical no tiene ninguna restriccion en el número de componentes del vector, sin embargo, es necesario almacenar muchos resultados intermedios (sumas parciales) y esto incrementa el ancho de banda de la memoria. Es por esto que el procesamiento vertical se aplica en arquitecturas que tienen un esquema de memoria a memoria. Algunos ejemplos son la STAR-100 y la Cyber 205.

En el procesamiento ciclico tampoco se tiene estriccion en el número de componentes del vector, pero a diferencia del procesamiento vertical, aqui unicamente se deben almacenar algunos resultados intermedios debido a que se procesa por bloques. En este caso es factible utilizar una memoria cache para almacenar los resultados intermedios. Es por esto que el procesamiento ciclico es más aplicable en arquitecturas con esquema registro a registro. Como ejemplo podemos dar a la Cray-1 y a la Fujitsu VP-200.

C A P I T U L O III

Procesadores de Arreglos y Procesadores Asociativos

Introducción a los Procesadores de Arreglos y a los Procesadores Asociativos

En este capítulo se estudian a los procesadores de arreglos y a los procesadores asociativos. Se presentan las organizaciones básicas y las tecnicas de control de los procesadores de arreglos, asi como algunas redes utilizadas en la interconexión de los PE. Despues se estudian a las caracteristicas esenciales de las memorias asociativas para posteriormente entrar en los procesadores asociativos. Se presentan como ejemplos a la Illiac-IV y a la PEPE.

3.1 Computadoras SIMD

Empezamos esta sección definiendo a un **procesador de arreglos** como un arreglo síncrono de procesadores paralelos. Este arreglo consiste de varios elementos procesadores (PE, ver capitulo 1) que se encuentran supervisados por una unidad de control (CU). Los procesadores de arreglos pueden manejar un flujo de instrucciones único y un flujo de datos múltiple, es por esto que a los procesadores de arreglos tambien se les conoce como computadoras SIMD. Las computadoras SIMD están diseñadas para realizar operaciones vectoriales en matrices o en arreglos de información.

Existen dos tipos de arquitecturas en las computadoras SIMD: los procesadores de arreglos que esencialemente utilizan RAM y los procesadores asociativos que utlizan AM (associtive memory). Primero nos enfocaremos en el estudio de los procesadores de arreglos y mas adelante veremos a los procesadores asociativos.

Debe quedar claro que en esta tesis utilizamos el término procesadores de arreglos para las computadoras SIMD que usan memoria RAM convencional y cuando hablemos de un procesador asociativo nos referiremos a las computadoras SIMD que utilizan memoria asociativa.

Podemos clasificar a las computadoras SIMD en cinco grupos, basandonos en como procesan la información (por palabra o por bit) y de acuerdo al número de unidades de control utilizadas:

+ Procesadores de arreglos por palabra.
+ Procesadores de arreglos por bit.
+ Procesadores de asociativos por palabra.
+ Procesadores asociativos por bit.
+ Computadoras SIMD multiples.

En la tabla 3.1 se presenta una lista de computadoras SIMD especificando el tipo de arquitectura con que han sido diseñadas. En esta tesis se hara una breve descripcion de la Illiac-IV como ejemplo de un procesador de arreglos y de la PEPE para ejemplificar un procesador asociativo.

Unger	wos array	Proposed by Unger (1958)
Solomon	wos array	Proposed by Slotnick (1962)
VAMP	wos array	Proposed by Senzig and Smith (1965)
ILLIAC	wos array	Illiac-IV operational 1972 (Section 6.2)
BSP	wos array	Developed by Burroughs and suspended in 1979 (Section 6.2)
CLIP	bis array	Developed at University College, London. See Duff (1976)
DAP	bis array	Developed by ICL, England, section 3.3 in Hockney and Jesshope (1981)
MPP	bis array	Developed by Goodyear Aerospace (Section 6.3)
PEPE	wos ass	Developed by Burroughs Corp. and System Dev. Corp. (Section 5.4.2)
STARAN	bis ass	Developed by Goodyear Aerospace Corp. (Section 5.4.2)
OMEN	bis ass	Developed by Sanders Associates, chapter 7 in Thurber (1976)
RELACS	bis ass	Proposed for database machine in Berra and Oliver (1979)
MAP	wos MSIMD	Proposed by Nutt (1977) (Section 6.4.4)
PM^4	wos MSIMD	Proposed by Briggs and Hwang et al. (1979) (Section 6.4.4)
Phoenix	wos MSIMD	Proposed by Feierbach and Stevenson (1979)
NASF	wos array	Proposed in Stevens (1979)

Tabla 3.1. Sistemas de computadoras SIMD.[1]

Formalmente una computadora SIMD, C, puede ser caracterizada por el siguiente conjunto de parámetros:

$$C = \langle N, F, I, M \rangle$$

donde:
- N = número de PE's en el sistema.
- F = conjunto de funciones que ofrece la red de interconexión o de alineamiento (alignement, ver sección 3.2) para establecer diferentes rutas
- I = conjunto de instrucciones de máquina para el manejo de vectores y escalares, rutas y operaciones con la red.
- M = conjunto de mascaras, donde se pueden tener PE's habilitados o deshabilitados.

Mediante este modelo se pueden evaluar diferentes máquinas SIMD, en las siguinetes secciones se hablará de los distintos tipos de redes empleados así como de las posibles rutas que se pueden seguir.

3.1.1 Historia de los procesadores asociativos y de arreglos.

En 1958, Unger diseño una estructura de computadora para resolver problemas espaciales. La computadora de Unger se presenta en la figura 3.1. Se tiene un arreglo de PE's dirigidos por un control maestro. Esta computadora se propuso para el procesamiento en reconocimiento de patrones.

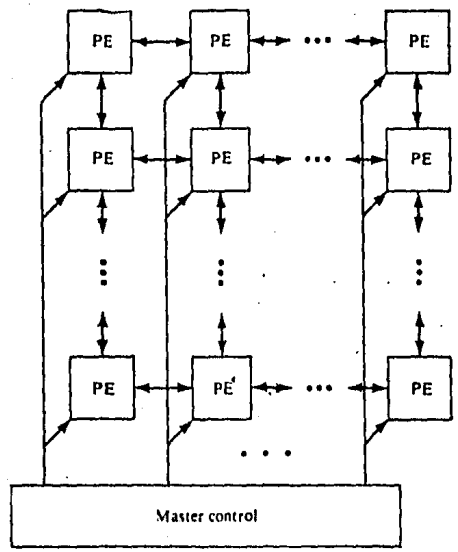


Figura 3.1. Computadora de Unger con procesamiento espacial.[1]

En 1962 el profesor Slotnick diseñó la computadora Solomon que mejoraba el trabajo de Unger. Sin embargo ninguna de las computadoras fue construida. Estos diseños sirvieron como base para que se construyeran varias computadoras SIMD mas tarde. En 1965, Senzig y Smith construyeron un multiprocesador de vectores aritmetico (VAMP) que consistia de un arreglo lineal de PE's con modulos de memoria compartida y pipelines aritmeticos compartidos. La computadora Illiac-IV fue la primera supercomutadora de arreglos y fue construida a finales de 1960. Esta fue la base para desarrollar toda la serie Illiac. Uno de sus principales sucesores es la BSP de Burroughs. Estos dos sistemas ya no se encuentran en operacion, sin embargo resultan muy interesantes en sus principios. En 1979 se propuso extender las arquitecturas de la Illiac-IV y la BSP para alcanzar velocidades de gigaflops. El proyecto Phoenix sugirio un arreglo multiple de computadoras SIMD. Utilizando 16 Illiac-IV para dar un total de 1024 PE's. Por su parte Burroughs desarrolló una arquitectura de 512 PE's compartiendo 521 modulos de memoria, esta propuesta fue hecha por la NASA. Se han desarrollado varios procesadores de arreglos por bit tanto en Europa como en Estados Unidos. La CLIP-4 es uno de estos procesadores construidos con una malla de 96X96 PE's. se diseñó para el procesamiento de imagenes. La DAP (Distributed Array Processor) se desarrolló en Inglaterra y puede ser configurada en arreglos de 32X32, 64X64, 128X128 o 256X256 PE's. La MPP (Massively Parallel Processor) es un procesador de arreglos por bit con 128X128 PE's construida en los 80's.

En la tabla 3.1. se ven cuatro procesadores asociativos. La PEPE es el unico procesador por palabra que se conoce [1]. La STARAN, la OMEN y la RELACS son todas procesadores asociativos por bit. Los procesadores asocitivos tienen grandes aplicaciones en el manejo de informacion y en operaciones con basos de datos. La RELACS es una maquina propuesta para el manejo de grandes

bases de datos, fue diseñada en la Univesidad de Syracuse en 1979. En la actualidad la STARAN es la única computadora comercial con procesador asocitivo que verdaderamente es utilizada. En la figura 3.2. se presenta el árbol genealogico de las computadoras SIMD.

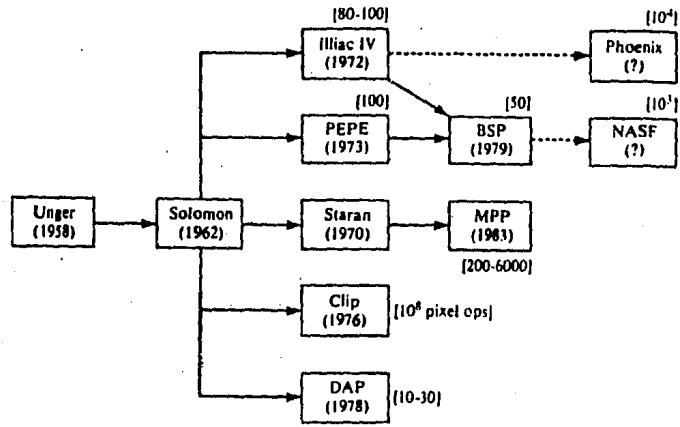


Figura 3.2. Arbol genealogico de las computadoras SIMD (los números entre parentesis representan el desempeño de la computadora en *mflops*).[1]

3.1.2 Perspectivas de las computadoras SIMD.

Se puede ver claramente que este tipo de computadoras son de propósito especifico. Cuando se usan en estas aplicaciones, las computadoras SIMD tienen desempeños verdaderamente impresionantes. Sin embargo, los procesadores de arreglos tienen problemas de programación y de vectorización que no son faciles de resolver. La realidad es que los procesadores de arreglos no son muy aceptados por los fabricantes de computadoras.

Las computadoras MSIMD (multiple-SIMD) son una subclase de las MIMD. Existen flujos de instrucciones múltiples en arreglos de procesadores múltiples. Cada flujo de instrucciones maneja varios conjuntos de información igual que en una SIMD. La Illiac-IV fue inicialmente concebida como una MSIMD. En la siguiente lista se presentan algunas de las aplicaciones sugeridas para los procesadores de arreglos.

+ Algebra de matrices.
+ Calculo de valores y vectores característicos.
+ Programación lineal y entera.
+ Modelado del Tiempo.
+ Filtrado y análisis de Fourier.
+ Procesamineto de imágenes.
+ Reconocimineto de patrones.
+ Generación automatica de mapas.

La lista anterior no es ni con mucho exhaustiva.

3.2 Procesadores de Arreglos

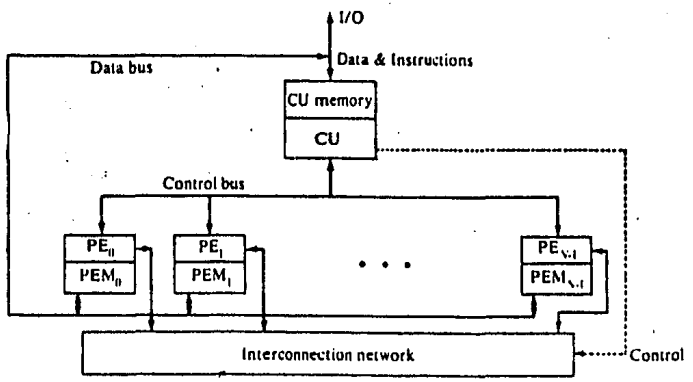
3.2.1 Arquitecturas básicas.

En los procesadores de arreglos existen dos tipos de arquitecturas basicas, como se muestra en la figura 3.3. En el capitulo 1 se presento la configuracion de la figura 3.3.a. y es la que utiliza la computadora Illiac-IV. Esta configuración está estructurada con N PE's sincronizados, todos bajo el control de una CU. Cada PE es esencialmente una ALU con registros internos y memoria local PEM. La CU tiene su propia memoria principal en donde se almacenan los programas. La CU ejecuta los programas del sistema y de usuario. Básicamente la CU decodifica las instrucciones y decide donde deberán ejecutarse. Cuando se trata de instrucciones escalares la ejecución se realiza dentro de la CU y cuando se trata de vectores se distribuyen en los PE's para obtener paralelismo espacial.

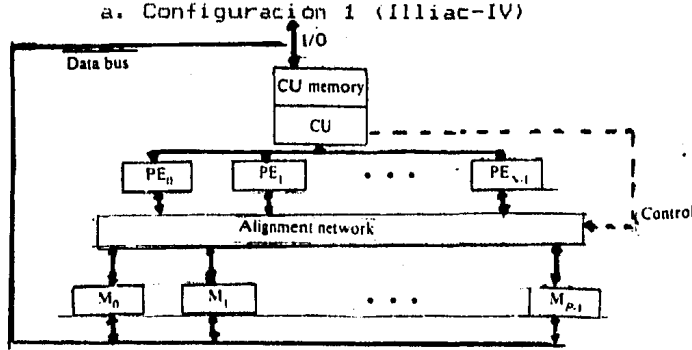
Todos los PE's realizan la misma función sincronamente. Cuando se tienen operandos vectoriales se distribuyen en todos los PE's antes de realizar su ejecución en paralelo mediante el arreglo de PE's. Los operandos se pueden cargar en los PE's via el CU utilizando el bus de control o mediante un dispositivo externo utilizando el bus de datos del sistema. Durante la ejecucion de una instrucción vectorial se utilizan metodos de mascareo para controlar el estado de los PE's. Cada PE puede estar habilitado o deshabilitado durante el ciclo de instrucción. Para controlar el estado de todos los PE's se utiliza un vector. Con lo anterior queremos decir que durante la ejecución de una instrucción vectorial no es necesario que todos los PE's estén habilitados. Para realizar intercambios de información entre los PE's se utiliza una red interna de comunicacion, esta red funciona bajo el control de la CU y puede tener distintas configuraciones. Mas adelante se estudiaran algunas redes comunes de insterconexion entre PE's.

Generalmente un procesador de arreglos se conecta a una computadora anfitrión mediante la unidad de control. Esta computadora es de proposito general y tiene como funciones el manejo y administracion de los recursos asi como la supervisión de los dispositivos de E/S. La CU se encarga de la ejecución de los programas mientras que la computadora anfitrion realiza las funciones de E/S y comunicación con el mundo exterior.

La otra forma de construir un procesador de arreglos es la mostrada en la figura 3.3.b. Esta configuración es diferente a la anterior en dos aspectos basicos. Primero, los arreglos de memoria que antes eran locales a cada PE son reemplazados por un banco paralelo de memoria que es compartida por todos los PE's mediante una red de alineación (alignement). Segundo, la red de interconexion de los PE's es reemplazada por la red de alineacion la cual sigue estando bajo el control de la CU. Se pueden tener N PE's y F modulos de memoria. Generalmente se escogen N y F para que sean relativamente primos. Como ejemplos de computadoras diseñadas con esta arquitectura tenemos a la BSP (Burroughs Scientific Processor).



a. Configuracion 1 (Illiac-IV)



b. Configuracion 2 (BSP)

Figura 3.3 Arquitecutas de computadoras SIMD.[1]

3.2.2 El elemento procesador PE.

En esta parte vamos a realizar un estudio del elemento procesador usado en una arquitectura como la de la figura 3.3.a. En esta arquitectura cada PE_i es un procesador que tiene memoria propia PEM_i, registros de proposito general y banderas A_i, B_i, C_i y S_i, una ALU, un registro para indices I_i, una registro para datos D_i y un registro para rutas R_i. En la figura 3.4 se presenta el diagrama de un elemento procesador.

El registro R_i de cada PE_i está conectado al registro R_j de otro PE mediante la red de interconexión. Cuando se presenta una transferencia de datos entre los procesadortes quiere decir que el contenido de los registros es el que se esta transfiriendo. El registro D_i se utiliza para guardar los bits de direcciones del PE_i. Como veremos mas adelante, esta estructura es la que se empleo en el diseño de la Illiac-IV.

Algunos procesadores de arreglos pueden utilizar dos registros para rutas, uno para entrada y el otro para salida. Aquí solamente se utiliza uno solo y para aislar las entradas de

las salidas utilizamos un flip-flop maestro esclavo. En un ciclo de instruccion cada PE_i puede estar en modo habilitado o deshabilitado. Si un PE_i esta en modo habilitado, ejecutará las instrucciones que le proporcione la CU. En el caso de que se encuentre deshabilitado no ejecutará la instruccion que la CU le dé. El registro S_i se utiliza para determinar el modo de operación o la bandera de estado del PE_i. Por convencion se utiliza $S_i = 1$ para habilitado y $S_i = 0$ para deshabilitado. El conjunto de todos los S_i forman el vector de estado S. La unidad de control tiene dos registros, uno de índices I y otro de mascaras M. El registro M tiene tantos bits como PE's existen en el arreglo. La CU utiliza el registro M para especificar cuales PE's estarán habilitados y cuales no, cuando están definidos todos los elementos del registro M, simplemnte se intercambia con el vector de estado S.

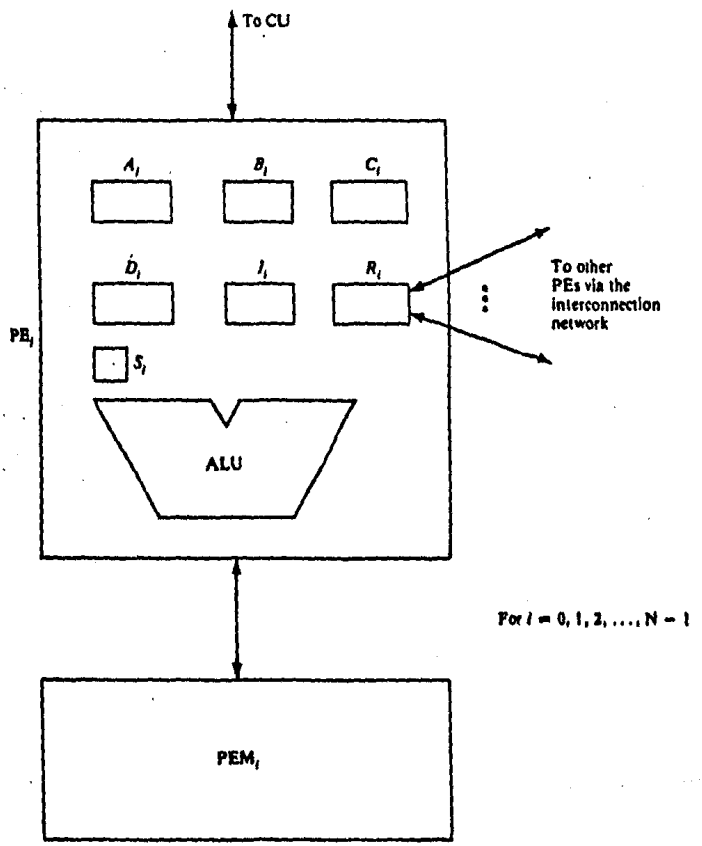


Figura 3.4. Componentes del elemento procesador PE.[1]

Desde el punto de vista de hardware, la longitud máxima que puede tener un vector para ser procesado, está determinada por el número de PE's. Cuando se presenta un vector que supera dicha longitud, entonces la CU se encarga de segmentar el vector en pedazos, de definir la dirección de inicio y el incremento. Un vector (unidimensional) puede ser almacenado en todos los PEM's si su longitud es menor que el número de PE's. En el caso que la longitud del vector sea mayor será necesario almacenarlo cíclicamente en todos los PEM's. Cuando se quieren almacenar matrices se presentan dificultades debido a que tanto las columnas como los renglones son utilizadas por los cálculos intermedios. La matriz deberá ser almacenada de tal forma que permita el fetch de columnas, renglones o diagonales parcialmente en un solo ciclo de memoria. Como resulta evidente esto no es una tarea sencilla.

En un procesador de arreglos, los operandos vectoriales pueden ser especificados mediante los registros que van a utilizar o mediante las direcciones de memoria que referenciarán. Cuando se trata de instrucciones que referencian a memoria, cada PE_i accesa su memoria local PEM_i utilizando su registro de índice I_i para realizar el incremento, es decir, el regitro I_i modifica la dirección inicial propuesta por la CU. Es por esto que se pueden accesar distintas localidades en los diferentes PEM's simultaneamente, utilizando la misma dirección inicial que especifico la CU.

3.2.3 Características de las comunicaciones entre PE's.

En esta sección presentamos algunas características fundamentales que poseen las comunicaciones entre PE's. Estas características deben tomarse mucho en cuenta al diseñar la red de interconexión en el procesador de arreglos de la figura 3.3.a. Existen cuatro características principales: modos de operación, estrategias de control, estrategias de switcheo y topologías de redes. Estas cuatro características se analizan a continuación.

Modo de operación. Se pueden definir dos tipos de operaciones: síncrona y asíncrona. Se emplea la comunicación síncrona cuando se quieren realizar comunicaciones para funciones que manajan datos o para manejar instrucciones. Las comunicaciones asíncronas se emplean en el multiprocesamiento donde las conexiones se deben realizar en forma dinámica. Un sistema puede ser diseñado para facilitar tanto las comunicaciones asíncronas como las síncronas. Es por esto que los modos de operación en las redes de interconexión se clasifican en tres grupos: síncrono, asíncrono y combinado. Todas las computadoras SIMD actuales han optado por el modo síncrono.

Estrategia de control. Una red de interconexión típica consiste de un cierto número de elementos cambiantes (switching elements) y las ligas de interconexión. Las funciones de interconexión se realizan dando adecuadamente el control a los elementos cambiantes. Cuando un controlador centralizado se encarga de manejar la señal de control se dice que la estrategia es de control centralizado. Cuando la señal de control es manejada por

el elemento cambiante, se dice que es una estrategia de control distribuida. La mayoria de las computadoras SIMD han escogido la estrategia de control centralizada utilizando a la CU como controlador.

Metodologia de switcheo. Existen dos metodologias de control básicas: por circuito y por paquete. En switcheo por circuito se establece una union fisica entre el emisor y el receptor. En switcheo por paquete, la información se manda en un paquete a través de la red de interconexión sin establecer una unión fisica. En general, el switcheo por circuito es mejor cuando se quieren transmitir muchos datos, mientras que el switcheo por paquete es mas eficiente cuando se transmiten mensajes cortos. Existe también la posibilidad de realizar una combinación de ambos, por lo que se pueden definir tres metodologias: switcheo por circuito, switcheo por paquete y switcheo integrado. En general las computadoras SIMD utilizan switcheo por circuito.

Topologia de redes. Una red puede ser representada mediante una grafica en la que los nodos sean puntos de cambio (switching points) y los arcos ligas de comunicacion. Las topologias tienden a ser regulares y se agrupan en dos categorias principales: estáticas y dinamicas. En las topologias estáticas las ligas entre dos procesadores son pasivas y los buses no pueden ser reconfigurados para obtener una conexion directa a otro procesador. En las topologias dinámicas si pueden ser reconfiguradas mediante los elementos cambiantes.

Podemos formar un espacio de las caracteristicas de interconexiones de PE's mediante el producto cartesiano de <modos de operación> x <estrategias de control> x < metodologias de switcheo> x < topologias de redes>. Sin embargo no todas las combinaciones son interesantes y para decidir cual es la mas adecuada debemos analizar otros aspectos como demanda, costo, etc. En general la caracteristica más importante ha sido la topologia de la red, por lo que en la siguiente sección se analizan detalladamente algunas topologias importantes.

3.2.4 Redes de interconexión entre PE's.

En esta sección se presentan las redes más comunes utilizadas en las computadoras SIMD. Debemos hacer la aclaración que esta sección es aplicable tanto a los procesadores de arreglos como a los procesadores asociativos. En general, los esquemas aqui presentados también se pueden aplicar a redes de computadoras, sin embargo, no es esta la finalidad. Por otro lado, no se pretenden analizar las redes más complejas tales como la Clos, Benes, Shuffle-Exchange y Omega que pueden ser estudiadas en [1].

Tomando como base la definición de redes dinámicas y estaticas hecha en la sección anterior, empezamos clasificando las redes en estos dos grupos, proponiendo ejemplos y analizando los mas relevantes.

Redes estáticas VS redes dinámicas.

Formalmente, una red de interconexión entre PE's puede ser especificada mediante un conjunto de funciones de ruta. Suponiendo que el conjunto de las direcciones de los PE's es S={0,1,...,N-1}, entonces cada función deberá ser biyectiva del conjunto S a S. Cuando una función de ruta es ejecutada mediante la red de interconexión, el PE_i copia el contenido del registro R_i en el registro $R_{f(i)}$ del $PE_{f(i)}$. Esta operación se presenta simultáneamente en todos los PE's, pero si el PE está deshabilitado entonces puede recibir información mas no transmitirla. Cuando se desea transmitir información entre dos PE's que no estan directamente conectados, es necesario hacer uno o varios puentes con PE's intermedios hasta lograr la comunicación. Las redes de interconexión se clasifican de acuerdo a la topología de redes en: estáticas y dinámicas. A continuación se analizan estos dos grupos.

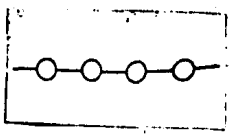
Redes estáticas. Las redes estáticas se dividen en grupos de acuerdo a las dimensiones que poseen, es decir, unidimensional, bidimensional, tridimensional. En la figura 3.5. se presentan algunos ejemplos. El arreglo lineal (3.5.a.) es muy utilizado en arquitecturas pipeline y es una red estática unidimensional. De la figura 3.5.b. a la f. se presentan redes bidimensioneles conocidas como anillo, estrella, árbol, malla y arreglo sistólico. Dentro de las redes tridimensionales se encuentran el anillo completamente interconectado, el anillo seminterconectado, el cubo y el cubo con ciclos; ver figura 3.5.g. a j. respectivamente. Lo más relevante de este tipo de redes es que no se pueden modificar sus interconexiones un vez alambradas (debe entenderse el término "no se pueden" como instantaneamente).

Redes dinámicas. Se pueden clasificar en dos grupos: uninivel y multinivel.

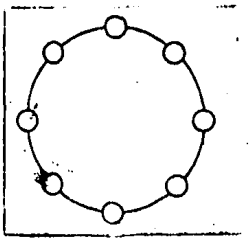
Redes uninivel. Una red uninivel esta constituida por N selectores de entrada (IS) y N selectores de salida (OS), como se puede ver en la figura 3.6. Cada IS es esencialemente un DEMUX de 1 a D y cada OS es un MUX de M a 1 donde 1<= D <=N y 1<= M <=N para seleccionar la ruta deseada es necesario tener diferentes señales de control aplicadas a los selectores de IS y OS.

A las redes uninivel también se les conoce como recirculantes. Esto es debido a que si M<N o D<N, entonces para lograr establecer una comunicación entre dos puntos puede ser necesario que dé varias vueltas antes de lograrlo. El numero de vueltas dependerá de M y D, es decir, entre mayor sea el número de M y D menor será el número de vueltas. Para el caso en que M=D=N , se le conoce como red de cruz uninivel y solamente es necesario pasar una vez. Sin embargo este tipo de redes tienen un costo del orden de N^2 por lo que no son adecuados para N grande.

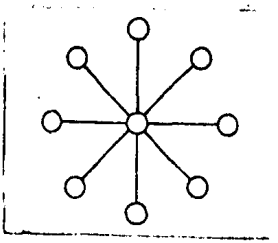
Redes multinivel. Una red multinivel esta formada de varios niveles con switches interconectados. Para describir este tipo de redes se utilizan tres puntos importantes: la caja de cambio (switch box), la topología de la red y la estructura de control. Se utilizan en estas redes muchas cajas de cambios. Esencialmente.



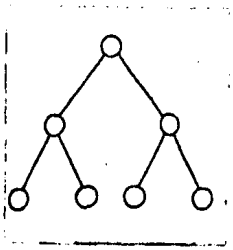
a. Arreglo lineal



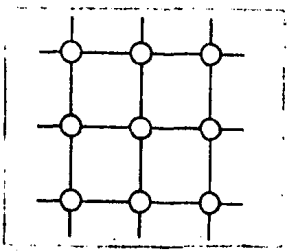
b. Anillo



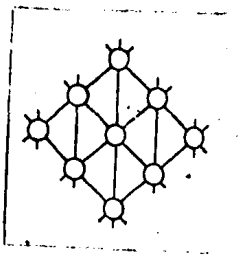
c. Estrella.



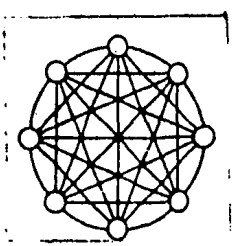
d. Arbol



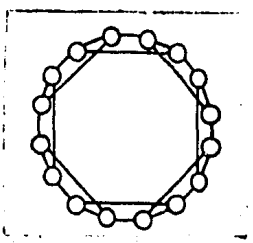
e. Malla



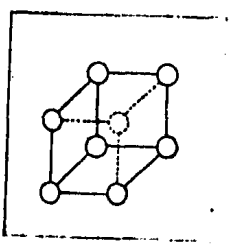
f. Arreglo sistólico



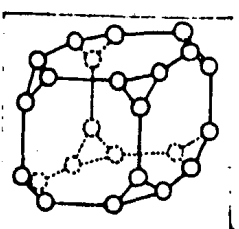
g. Completamente interconectada.



h. Anillo cordal



i. Cubo tridimensional



j. Cubo conectado cíclico.

Figura 3.5. Topologías de algunas redes estáticas.[1]

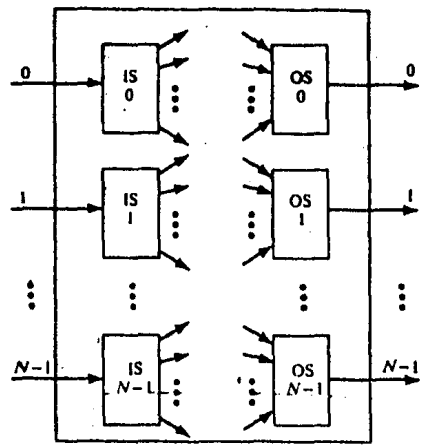


Figura 3.6. Red uninivel.[1]

cada caja es un dispositivo de intercambio con dos entradas y dos salidas como se muestra en la figura 3.7. Se tienen cuatro posibles estados en los que se puede encontrar una caja de cambios: directo, intercambio, inferior y superior. Una caja con dos funciones puede estar en el estado directo o en intercambio. Una caja con cuatro funciones puede estar en cualquiera de las cuatro.

Una red multinivel es capaz de conectar una terminal de entrada X a una terminal de salida Y. Este tipo de redes pueden ser de un lado o de dos lados. Las redes de un solo lado tienen los puertos de entrada y salida en el mismo lado. Las redes de dos lados tienen un lado de entrada y el otro de salida, estas a su vez se dividen en: con bloqueo, sin bloqueo y reorganizables. En las redes con bloqueo, cuando se quiere conectar simultáneamente más de dos terminales pueden resultar conflictos al utilizar las líneas de comunicación. Algunos ejemplos de este tipo de redes son: Omega, flip, Cubo n-ésimo y baseline. En la figura 3.8 se presenta el esquema de la red baseline.

En una red reorganizable se pueden obtener todas las interconexiones entre terminales en base a reorganizar las líneas. Un ejemplo típico es la red Benes que se muestra en la figura 3.9.

Una red sin bloqueo puede lograr cualquier conexión entre dos terminales sin bloquear, como ejemplo está la red Clos que se muestra en la figura 3.10.

En [1] se analizan detalladamente varios tipos de estas redes.

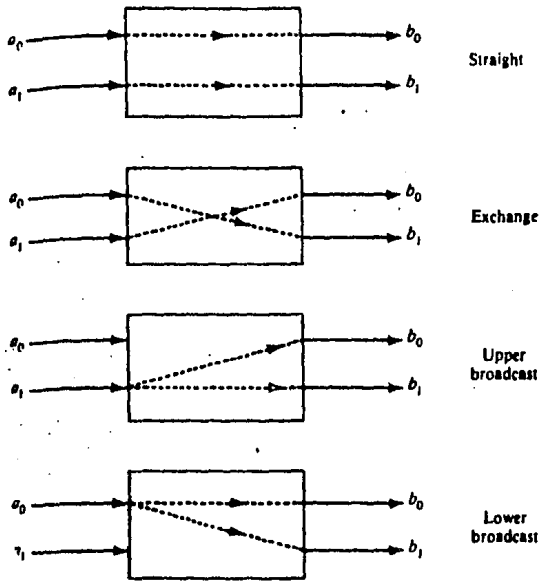


Figura 3.7. Cajas de cambios en sus cuatro estados de interconexión.

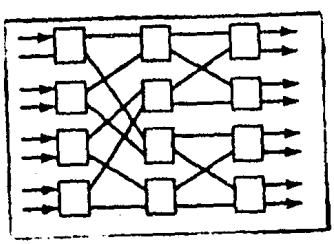


Figura 3.8. Red baseline.

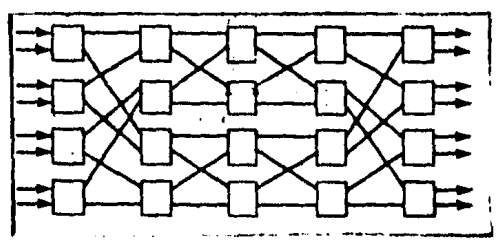


Figura 3.9. Red Benes.

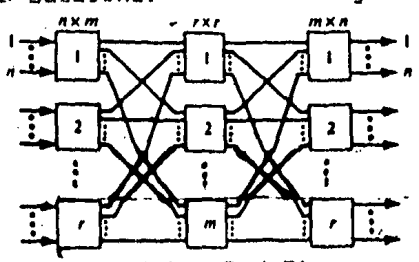


Figura 3.10. Red Clos.

3.2.5 Una aplicación: La Illiac-IV.

Los procesadores de arreglos se dieron a conocer principalmente por el desarrollo del hardware y el software de la Illiac-IV. Es por esto que en esta sección presentamos un analisis de la arquitectura y algunas características importantes de este sistema.

El sistema Illiac-IV fue diseñado en la Universidad de Illinois en 1960. El sistema se implementó en 1972 por Burroughs Co. El objetivo era desarrollar una computadora con alto grado de paralelismo que pudiera realizar operaciones vectoriales y matriciales a una tasa de 10^9 operaciones por segundo. Para lograr esta objetivo, el sistema necesito 256 PE's bajo la supervisión de 4 CU's. Debido al alto costo, el sistema se fabricó con 64 PE's y con una CU. La velocidad se aproxima a 200×10^6 operaciones por segundo.

Los 64 PE's de la Illiac-IV están interconectados en base a una red estática bidimensional como se muestra en la figura 3.11. Los PE's están numerados del 0 al 63. El bus de la unidad de control (control unit bus) se usa para mandar bloques de 8 palabras de las PEM's a la CU. Se pueden usar los siguientes formatos: 64 ó 32 bits en punto flotante, 64 bits lógicos, 48 ó 24 bits en punto fijo y 8 bits en modo caracter. Las instrucciones a ejecutarse se distribuyen en todos los PEM's y el sistema operativo se encarga de que cada instrucción sea ejecutada.

El bus de datos común (comun data bus) se utiliza para llevar la información de la CU a todo el arreglo de PE's. Por ejemplo, cuando se tiene una constante, no es necesario cargarla en los 64 PE s si no solamente se puede guardar en un registro de la CU para que sea utilizada por cualquier PE habilitado.

La red de ruta (routing network) se usa para enviar información de un PE a otro. Para transferir información de un PE a su PEM se usan instrucciones regulares. Cuando más, son necesarias 7 instrucciones para transferir información de un PE a otro via la red de ruta.

La Illiac-IV utiliza un subsistema de E/S para sus comunicaciones con el mundo exterior, tiene un sistema de almacenamiento en disco y como computadora anfitrión utiliza una B6500 que se encarga de supervisar un banco de memoria laser de 10^{12} bits, además está conectada a la red ARPA; un esquema de este subsistema se puede ver en l figura 3.12.

La unidad de disco cuenta con 128 cabezas lectoras, una para cada pista, con una velocidad de rotación de 40 ms y una tasa efectiva de transferencia de 10^9 bits por segundo. La computadora B6500 maneja todas las demandas del programador referentes a los recursos del sistema. El sistema operativo, incluyendo los compiladores, los ensambladores y las rutinas de servicio de E/S también son almacenadas en la B6500. Como un sistema completo, la Illiac-IV se encuentra en segundo plano siendo utilizada como máquina de propósito general. La union con la red ARPA da la facilidad de que la Illiac-IV pueda ser utilizada por cualquier usuario de la red.

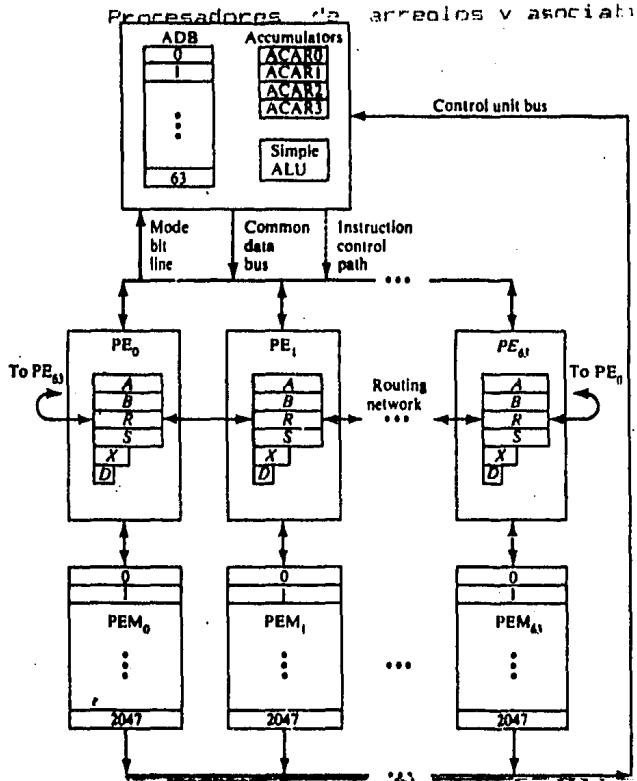


Figura 3.11. El arreglo de 64 PE's de la Illiac-IV.[1]

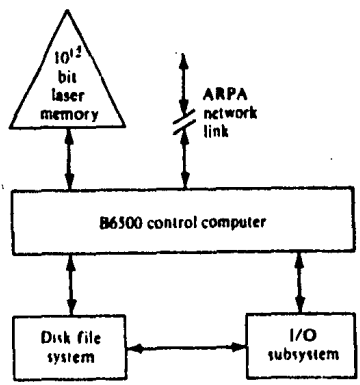


Figura 3.12. El sistema de E/S de la Illiac-IV.[1]

La unidad de control (CU) de la Illiac-IV realiza las siguientes funciones:

1.- Controla y decodifica el flujo de instrucciones.
2.- Transmite las señales de control a los PE's para ejecución de vectores.

3.- Proporciona las direcciones de memoria que son comunes a todos los PE's.
4.- Manipula las palabras que son comunes a todos los PE's en la ejecución.
5.- Recibe y procesa las señales de interrupción.

En la figura 3.13. se presenta un diagrama de bloques de la unidad de control. La CU además de ser un procesador escalar, tiene capacidad para controlar concurrentemente las operaciones del arreglo de PE's. El registro de instrucción PLA (Instruction Buffer) y el registro local de datos LDB (local Data Buffer) pueden almacenar hasta 64 palabras y son de muy rápido acceso. El PLA es direccionado asociativamente (ver siguiente sección) y guarda las instrucciones pendientes y las presentes. El LDB es una memoria cache para datos con 64 bits por palabra. Existen cuatro acumuladores (ACAR). La CU puede ejecutar operaciones escalares como adición, substracción, corrimiento y logicas, en su unidad aritmética. Cuando se necesitan operaciones más complejas o vectoriales entonces se utilizan los PE's. La cola final (final queue) es utilizada para almacenar los datos y direcciones que deben ser transmitidas a los PE's.

Todas las instrucciones son de 32 bits y se clasifican en instrucciones para la CU o instrucciones para PE. Las primeras se utilizan para control (indices, saltos, etc.) y para operaciones escalares. Las últimas son decodificadas por la prestación de instrucción ADVST (advenced instruction station) y luego son transmitidas a todos los PE's mediante señales de control. De hecho, la ADVST decodifica todas las instrucciones y ejecuta las de la CU, además construye las direcciones necesarias y los datos de los operandos despues de decodificar una instrucción de PE. Una caracteristica del PLA es que puede almacenar hasta 128 instrucciones, esto permite tener en memoria cache los ciclos enteros de la gran mayoria de los programas. En la figura 3.14. se presenta un diagrama de bloque de un PE de la Illiac-IV. Los componentes del PE son:

1.- Cuatro registros de 64 bits cada uno: A como acumulador, B como registro de operando, R como registro de ruta y S como registro general.
2.- Una unidad lógica con un sumador/multiplicador, una unidad logica y un switch Barrel para funciones aritmeticas, booleanas y de corrimiento respactivamente.
3.- Un registro de indice de 16 bits y un sumador para modofícacion de direcciones de memoria y control.
4.- Un registro de 8 bits para el modo de control para almacenar los resultados de pruebas y la informacion de las mascaras de los PE's.

Cada PE tiene comunicacion de 64 bits con cuatro vecinos. Para minimizar la distancia entre PE s, en la figura 3.15. se presenta un diagrama de interconexión.

El fetch de las instrucciones y datos necesita bloques de 8 palabras que se accesan paralelamente. Las memorias individuales de cada PE tienen un tiempo de acceso de 120 ns y un ciclo de memoria de 240 ns. Además, tienen capacidad de 2048 palabras.

Cada PEM es accesada independientemente por su PE, la CU o algún dispositivo de E/S. En general la Illiac-IV necesita grandes cantidades de memoria secundaria para almacenar los programas y los archivos de datos, por esto se usa una memoria de respaldo para los programas que exceden la memoria cache.

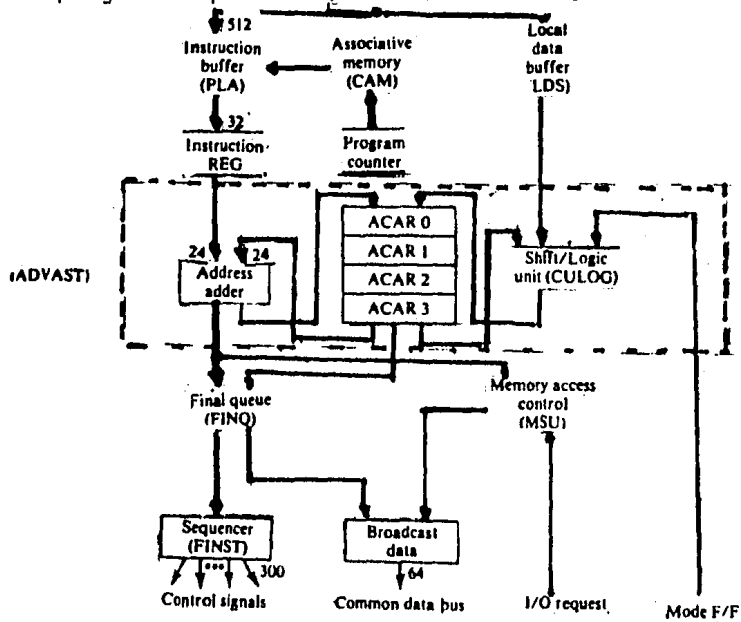


Figura 3.13. Diagrama funcional de la unidad de control de la Illiac-IV.[1]

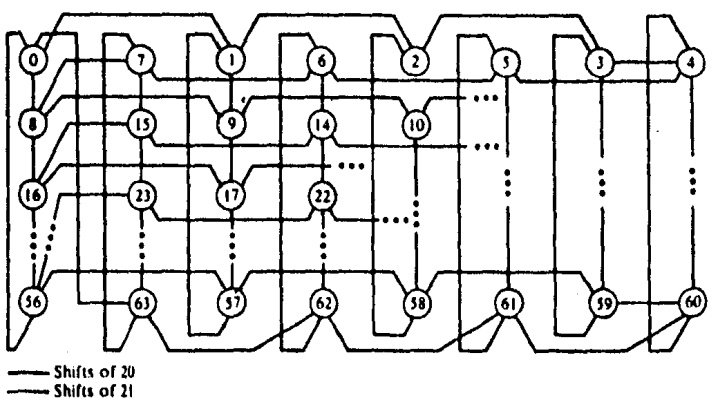


Figura 3.15. Sistema de interconexión de la Illiac-IV.[1]

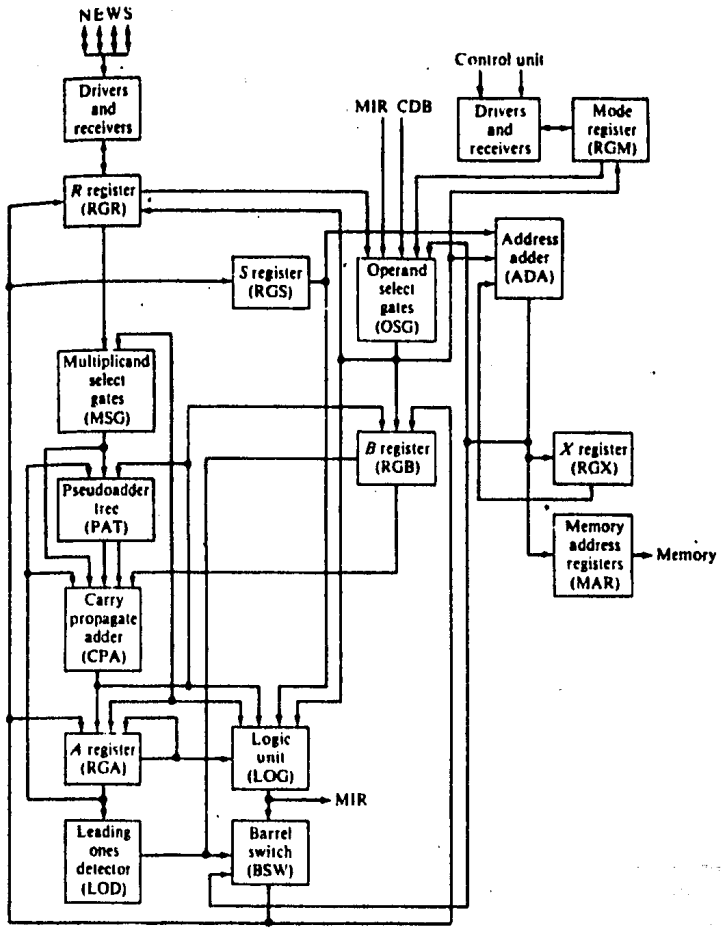


Figura 3.14. Estructura interna del PE de la Illiac-IV.[1]

3.3 Los Procesadores Asociativos

3.3.1. Descripción general.

Un procesador asociativo se puede describir como un procesador que cumple con las dos características siguientes:

1.- Los datos almacenados pueden ser accesados utilizando su contenido o parte de él (en vez de utilizar una dirección).
2.- Con una sola instrucción podemos realizar operaciones aritméticas y lógicas sobre un conjunto de argumentos.

Debido a estas características los procesadores asociativos procesan la información mucho más rápido que los procesadores secuenciales comunes, además pueden ser utilizados en muchas aplicaciones de procesamiento de información tales como búsquedas rápidas en grandes bases de datos, recuperación de información en bases de datos que se actualizan rápidamente, operaciones aritméticas y lógicas sobre grandes conjuntos de datos, procesamiento de señales de radar y predicción del tiempo. Sin embargo, debido al alto costo, los procesadores asociativos generalmente se utilizan en conjunto con computadoras secuenciales. Se piensa que debido al rápido desarrollo de los circuitos LSI, el costo de implementación de los procesadores asociativos se reducirá considerablemente por lo que se podrán construir más sistemas de este tipo [37].

En general la arquitectura de un procesador asociativo es como se muestra en la figura 3.16. Consiste de una memoria asociativa, una unidad aritmética y lógica, un sistema de control, una memoria de instrucción y una interfase de E/S. La principal característica de un procesador asociativo es que utiliza una memoria asociativa. La memoria asociativa es tan importante en esta arquitectura que se clasifican a los procesadores asociativos de acuerdo al tipo de organización que tiene su memoria asociativa. Antes de continuar vamos a describir a las memorias asociativas dado que son la columna vertebral de los procesadores asociativos.

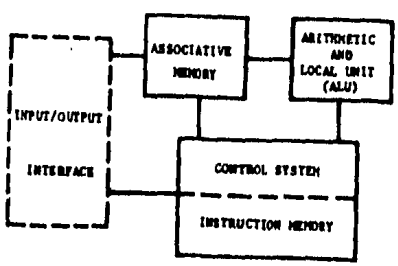


Figura 3.16. Diagrama de bloques de un procesador asociativo.[37].

3.3.2. Memorias asociativas.

Se define a una memoria asociativa como un sistema de memoria con la propiedad de que los elementos almacenados pueden ser recuperados utilizando su **contenido o parte de él** (esto es por la primera propiedad de un procesador asociativo).

Desde el punto de vista de hardware, para poder recuperar la información utilizando el contenido o parte de él, debemos poder accesar las palabras de la memoria en base a una llave de búsqueda que sea igual al contenido de la palabra o parte de la palabra. Hay que notar que no usamos direcciones. El elemento básico de las memorias asociativas se conoce como celda-básica (bit cell) y se muestra en la figura 3.17. Tiene la propiedad de que un bit de información puede ser escrito, leído o comparado con el bit de pregunta (interrogate). Las operaciones de búsqueda mascareo y comparación, son ejecutadas de acuerdo a la organización que se tenga en la memoria. Es posible que de una pasada se obtengan todas las palabras que concuerdan con la palabra de pregunta, sin embargo, es necesario usar algun método para almacenar todas las palabras concordantes. Generalmente se utiliza una función de búsqueda y un bit para indicar las palabras coincidentes, de esta forma con una sola instrucción es posible accesar todas las palabras concordantes. Debemos notar que una memoria puede ser asociativa si realiza las comparaciones en paralelo todos los bits de la palabra (palabra paralela o palabra serial) o en serie por bit (bit-serial).

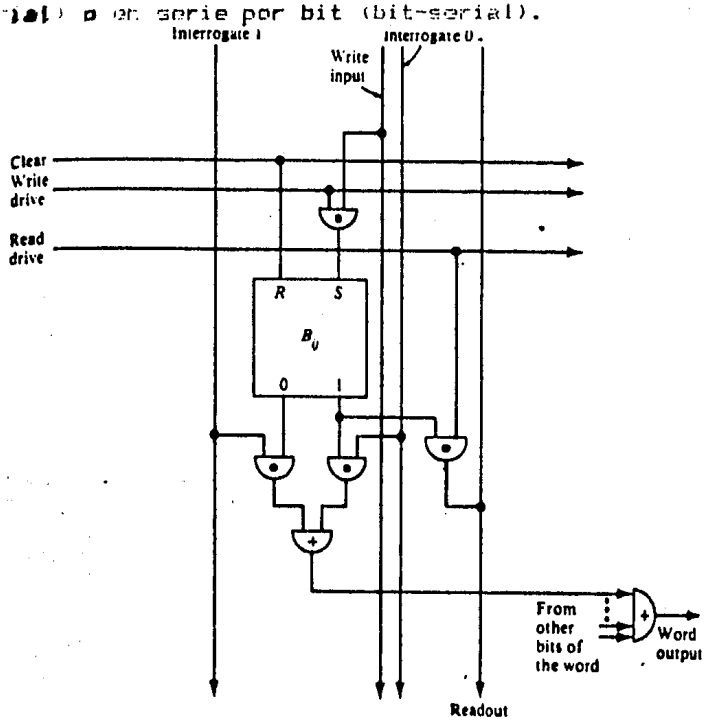


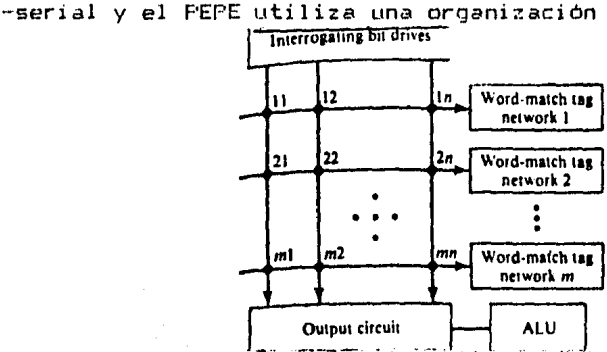
Figura 3.17. Diagrama del funcionamiento de la celda básica.[1]

Podemos clasificar en bit paralelo y bit serial, a continuación hacemos una descripción de ambas organizaciones.

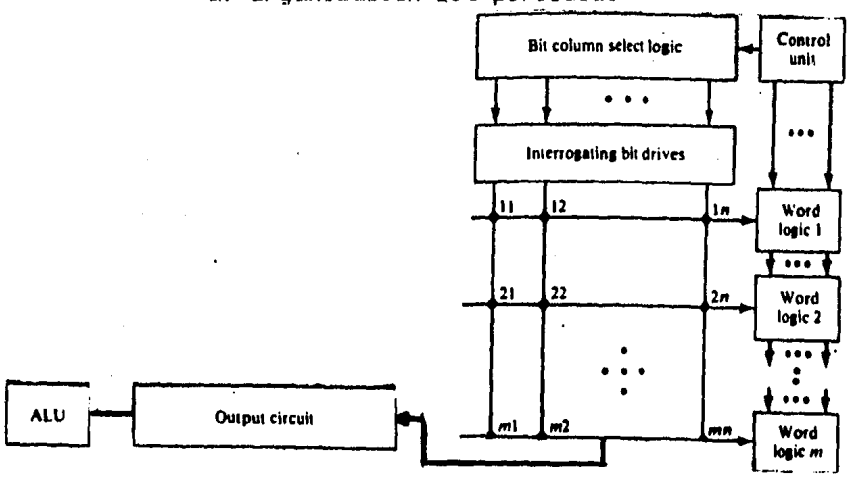
Organización bit paralela: En esta organización las comparaciones se realizan de la forma paralela por palabra o paralela por bit. Se involucran en el proceso de comparación todas las palabras que esten habilitadas. En la figura 3.18.a. se muestra esta organización. Cada cruce representa una celda básica y cada columna representa una palabra. Se tienen n palabras y m bits por palabra.

Organización bit serial: En la figura 3.18.b se presenta esta organización. Aquí no se analizan todos los bits de las palabras al mismo tiempo, si no sólo aquel bit que la unidad de control indique. Nuevamente cada cruce representa una celda básica.

El procesador asociativo STARAN utiliza una memoria bit-serial y el PEPE utiliza una organización paralela.



a. Organización bit-paralelo.



b. Organización bit-serial.

Figura 3.18. Organizaciones de memorias asociativas.[1]

3.3.3. Clasificación.

Desde el punto de vista operacional, un procesador asociativo puede realizar operaciones complejas además de las operaciones de comparación que le permite su memoria asociativa. Esto lo puede realizar si las palabras seleccionadas se transfieren de la memoria asociativa a la ALU vía el circuito de salida (output circuit) ver figura 3.18. En la ALU se realizan las operaciones y se guardarán de nuevo en la memoria asociativa si es necesario.

Desde el punto de vista de arquitectura, los procesadores asociativos pertenecen a la categoría de computadoras SIMD, es decir, un procesador asociativo es una maquina SIMD que cumple con las dos propiedades expuestas anteriormente.

La arquitectura de los procesadores asociativos puede ser clasificada en cuatro categorías de acuerdo a la forma en como realizan las comparaciones sus memorias. Estas cuatro categorías son: completemante paralela, bit serial, palabra serial y orientada a bloques. Los procesadores de palabra serial a su vez se dividen en organizados por palabra y lógica distribuida. Las categorías de bis serial y completamente paralelo son las más importantes y han tenido implementaciones. Las computadoras PEPE (Parallel Element Processing Ensamble) y la STARAN son las más conocidas en estas categorías. En la siguiente sección se realiza un analisis detallado de la arquitectura de la PEPE.

Un procesador palabra serial esencialmente representa una implementación en hardware de un programa de búsqueda. Lo que hace sobresaliente a esta arquitectura es que es mucho más eficiente realizar búsquedas por hardware que cualquier programa para búsquedas en una computadora secuencial.

3.3.4. Historia de los procesadores y las memorias asociativas.

La primera memoria asociativa fue desarrollada por Slade y McMahon en 1956. Desde entonces, se han implementado memorias asociativas con diodos túnel, semiconductores, arreglos de selenoides y circuitos integrados entre otros. Debido a que estas memorias tienen un alto costo de fabricación, las primeras memorias tenían baja capacidad, es decir, de 1Kpalabra de 100 bits cada una.

El primer procesador asociativo fue diseñado por Behnke y Rosenberger en 1963. Desde entonces se han construido una gran cantidad de procesadores utilizando distintos tipos de memorias asociativas. Sin embargo, estos procesadores no fueron muy conocidos hasta que se implemento la PEPE y la STARAN. En estos sistemas se cuenta con mayor capacidad de memoria debido a que la tecnología hizo posible que disminuyera su costo. Recientemente, Anderson y Kain presentaron la ECAM (Extended Content-Addressed Memory) utilizada en la fuerza aérea de E.U. y que tiene una gran cantidad de memoria asociativa en un solo chip. Al parecer el desarrollo de memorias asociativas con gran capacidad será posible en un futuro cercano.

3.3.5. Una aplicación: La PEPE.

La computadora PEPE es uno de los dos sistemas mas grandes construidos en base a un procesador asociativo. El concepto básico se derivo a partir de procesadores asociativos con lógica distribuida propuesto por Lee y fue implementado por los laboratorios Bell para la Agencia de Desfensa Contra Misiles y Balistica Avanzada de la Armada de E.U. Actualmente esta agencia esta construyendo un modelo más avanzado de la computadora PEPE.[37]

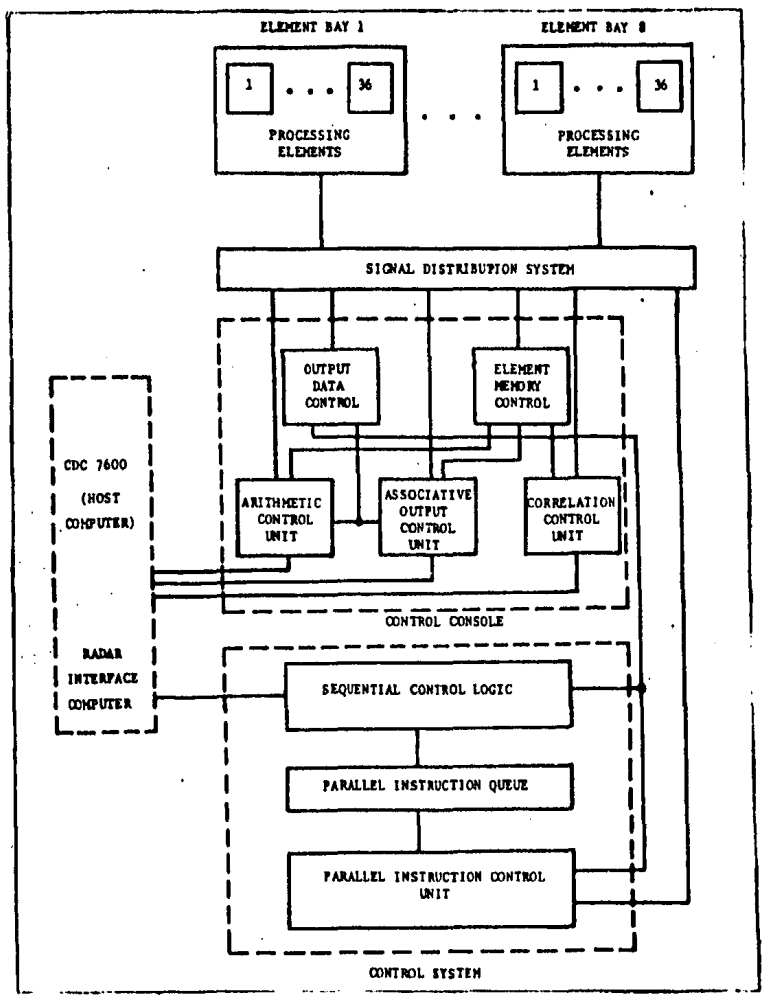


Figura 3.19. Diagrama de bloques de la computadora PEPE.[37]

La PEPE está compuesta de los siguientes subsistemas funcionales: un contol de salida de información, un control de memoria de elemento, una unidad aritmética y lógica, una unidad de correlación, una unidad de control de salida asociativa, el sistema de control y varios PE's. Por su parte, cada PE consiste de una ALU, una unidad de correlación, una unidad de salida asociativa y una memoria de 1024x32 bits. Hay que notar que el número de PE's utilizados en la PEPE es variable y puede ser incrementado o decrementado para satisfacer las necesidades de la aplicación. Esta capacidad no afecta el desempeño de la PEPE. En la figura 3.19 se presenta un diagrama de bloques de la PEPE. La arquitectura interna del PE de la PEPE se presenta en la figura 3.20.

Los PE's son los componentes que se encargan del computo en la PEPE. La computadora huésped es en este caso una CDC 7600 y es esta computadora la que manda los procesos y datos a todos los PE's. El proceso de carga esta definido mediante el paralelismo inherente que tiene la tarea y por la arquitectura de la PEPE que está diseñada para manejar tareas paralelas. En otras palabras a cada PE se le da la tarea de observar un objeto del radar (esta es la aplicación en la que se está usando la PEPE) y cada uno mantiene un archivo de datos actualizando su capacidad aritmética.

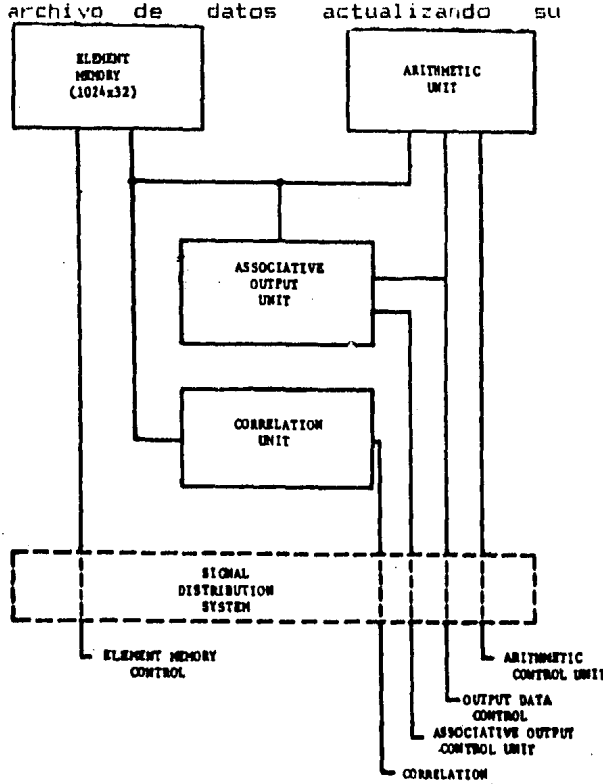


Figura 3.20. Diagrama del PE de la PEPE.[37]

C A P I T U L O IV

Flujo de Datos y Reducción

En este capitulo introducimos dos arquitecturas novedosas para procesamiento paralelo: flujo de datos y reducción. Hemos dividido el capitulo en tres partes que son: Introducción al Flujo de Datos y Reduccion, Flujo de Datos y Arquitecturas para Flujo de Datos y Reduccion. Dedicamos una parte a flujo de datos y ninguna a reducción debido a que se ha hecho más investigación y desarrollo en el primer topico que en el segundo. En la primera parte describimos lo que es el flujo de datos y la reduccion, así como una comparación entre estas y la arquitectura de von Neumann; en la segunda parte profundizamos en algunos aspectos de flujo de datos; finalmente en la tercera parte concretizamos el material de las primeras dos partes con algunos ejemplos concretos de arquitecturas de los dos tipos que tratamos en este capitulo.

PARTE I : Intoducción al Flujo de Datos y Reducción

Los principios de arquitectura de computadoras se han mantenido estáticos por más de 30 años, basados en la arquitectura de von Neumann. Los principios de von Neumann incluyen:

1. Un solo elemento computacional que incorpora procesador, memoria y comunicaciones.
2. Organizacion lineal de celdas de memoria de tamaño fijo.
3. Espacio de direccionamiento de las celdas en un nivel.
4. Lenjuaje de máquina de bajo nivel (las instrucciones realizan operaciones sencillas sobre operandos elementales).
5. Control centralizado y secuencial de la ejecución.

Sin embargo, a lo largo de los últimos años han sido propuestas, y aún implementadas, nuevas arquitecturas basadas en organizaciones paralelas "por naturaleza". El estímulo principal para estas novedosas arquitecturas ha venido del trabajo hecho por Jack Dennis sobre flujo de datos (data flow) y del trabajo de Klaus Berkling y John Backus sobre lenguajes y máquinas de reducción (reduction machines). Las arquitecturas de computadoras resultantes se pueden clasificar en accionadas por datos (data-driven) o accionadas por demanda (demand-driven). En las computadoras accionadas por datos la llegada de los operandos de una instrucción es las que activa su ejecucion, mientras que en las computadoras accionadas por demanda (es decir, reduccion) el requerimiento de un resultado es el que acciona la operacion que lo generará.

A pesar de que la motivacion y el enfasis de los distintos grupos de investigación varian , existen tres fuerzas que interactuan.

En primer lugar esta el deseo de utilizar concurrencia para mejorar el desempeño de la computadora. Como hemos visto

anteriormente existen áreas en las que se requieren computadoras de muy alta velocidad, y existen limitaciones fundamentales sobre los incrementos de velocidad que se pueden lograr mediante tecnología solamente (sección 1.4 y 1.5). Las computadoras convencionales de alta velocidad no satisfacen estas demandas.

En segundo lugar está el deseo de explotar los circuitos VLSI en el diseño de arquitecturas paralelas. Los intentos que se han hecho basados en arquitecturas von Neumann no han resultado ser muy alentadores..

En tercer lugar esta el creciente interes por nuevas clases de lenguajes de programacion de muy alto nivel. El ejemplo mejor conocido es LISP. Debido a que estos lenguajes no se adaptan bien a las arquitecturas von Neumann, las implementaciones han resultado ser ineficientes.

Se cree cada vez más, principalmente en Japon y Gran Bretaña, que las arquitecturas accionadas por datos y accionadas por demanda seran las arquitecturas posibles de las computadoras de la quinta generación. La pregunta es: ¿ Que principios de cada uno de los proyectos de investigación, son los que contribuirán a la nueva generacion de computadoras ?.

El trabajo sobre arquitectura accionada por datos y accionada por demanda cae en dos principales áreas de investigación: flujo de datos y reducción. Estas áreas se distinguen por la manera en las que se realiza el computo, los programas almacenados y los recursos de la computadora. Para resolver dificultades, los investigadores han introducido en cada una de sus areas ideas de las otras áreas, inclusive de la tradicional arquitectura de flujo de control (control-flow). Las ideas expuestas están basadas en [23].

4.1 Conceptos Básicos

En esta parte presentaremos modelos operacionales sencillos para flujo de control, flujo de datos y reducción. Para facilitar la comparación de estos tres modelos utilizaremos representaciones del enunciado a =(b + 1) * (b - c) que solamente incluye operadores, literales y referencias. A pesar de que este enunciado consiste de operadores y operandos sencillos, los conceptos ilustrados igualmente se aplican a operaciones y estructuras de datos mas complejas.

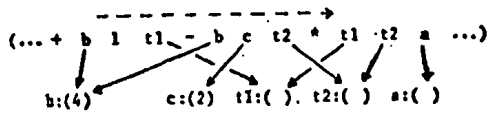
4.1.1 Flujo de control.

Comenzamos examinando el modelo más familiar: flujo de control. En la figura 4.1. el enunciado a=(b+1)*(b-c) se especifica por medio de una serie de instrucciones cada una de las cuales consiste de un operador seguido de uno o más operandos, que pueden ser literales o referencias.

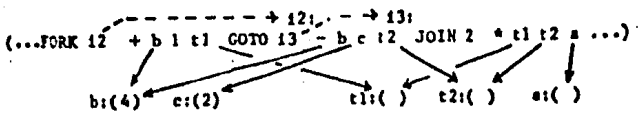
Por ejemplo, una operacion binaria como +, es seguida de tres operandos; los primeros dos, b y 1, son los datos de entrada, y t1 es la referencia a la celda de memoria compartida para el resultado. Es compartida en el sentido de que es el

medio por el cual se comunican las instrucciones; una instruccion deposita ahi su resultado mientras que otra toma de ahi mismo (cuando se encuentra disponible) un dato que le servira de entrada. En la figura, cada referencia se muestra como un arco sólido unidireccional. Los arcos sólidos muestran el acceso a datos mientras que arcos punteados definen el flujo de control. En el flujo de control tradicional existe un solo flujo de control como en la figura 4.1.a., que pasa de una instruccion a otra. Cuando el control llega a una instruccion, el operador es examinado para determinar el numero y el uso de los operandos que le siguen. A continuación se decodifican las direcciones de los operandos, se ejecuta la operación, se almacena el resultado y se pasa el control implicitamente a la siguiente instrucción en la secuencia. Se pueden realizar transferencias de control explicitamente mediante instrucciones del tipo GOTO.

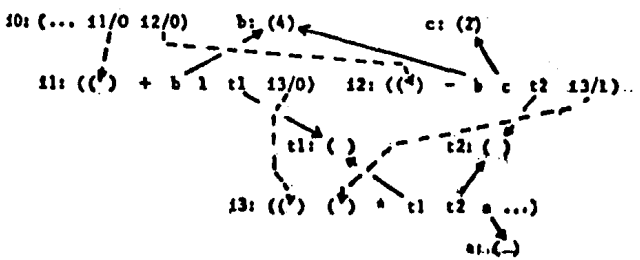
Tambien existen formas paralelas del flujo de control. En la forma mostrada en las figura 4.1.b. el modelo implícito secuencial de flujo de control es extendido al incluir operadores paralelos de control. Estos operadores permiten la existencia de



a. Secuencial.



b. Paralelo "FORK-JOIN".



c. Paralelo "fichas de control".

Figura 4.1. Programas de flujo de control para la expresión a=(b+1)*(b-c).[23]

más de un flujo de control en un mismo instante, y también proveén un medio para sincronizar estos flujos. Por ejemplo, en la figura 4.1.b, el operador FORK activa la instrucción de sustracción de la dirección i2 y pasa en forma implícita el control a la instrucción de adición. Por lo tanto, la adición y la sustracción se pueden ejecutar en paralelo. Cuando se termina de ejecutar la adición, el control pasa a la instrucción JOIN a causa de la instrucción GOTO i3. La tarea del JOIN es sincronizar los dos flujos de control dejando un solo flujo que active a la instrucción de multiplicar.

En la segunda forma de flujo de control, mostrada en la figura 4.1.c., cada instrucción especifica explícitamente su instrucción sucesora. En una referencia de esta forma, i1/0 por ejemplo, se especifica la siguiente instrucción (i1) y la posición del argumento (0) para esta señal de control o ficha (token) de control. Se representan las posiciones para los argumentos mediante paréntesis y una instrucción se ejecuta en el momento que ha recibido las fichas requeridas.

Las dos formas de flujo de control paralelo son semánticamente equivalentes; los FORK's son equivalentes a instrucciones múltiples de direcciones sucesoras y los JOIN's son equivalentes a argumentos vacios (parentesis) múltiples.

Los modelos secuencial y paralelo de flujo de control tienen varias características en común: 1. los datos pasan indirectamente de instrucción a instrucción vía referencias a celdas de memoria compartidas; 2. se pueden almacenar literales en las instrucciones, esto se puede ver como una manera de ahorrarse la referencia que sería necesaria para accesar la literal; 3. el flujo de control es implícitamente secuencial pero se pueden incluir operadores de control explícitos para paralelismo; y 4. debido a que los flujos de datos y de control están separados se pueden igualar o mantenerse diferentes.

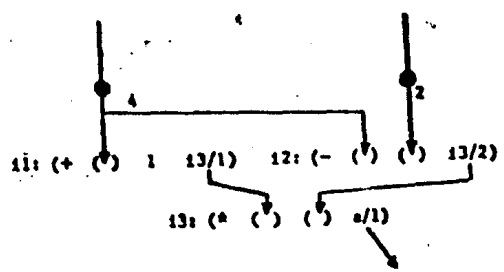
4.1.2 Flujo de datos

La segunda forma de flujo de control paralelo, en la que las instrucciones se activan mediante fichas y la necesidad de una ficha se indica mediante los símbolos (), es muy parecida a lo que es flujo de datos. Normalmente los programas de flujo de datos se describen en términos de gráficas dirigidas que indican el flujo de los datos entre instrucciones. En la representación del programa mostrada en la figura 4.2., cada instucción está formada por un operador, dos entradas que pueden ser literales u operandos desconocidos definidos por los símbolos (), y una referencia i3/1, que define la instrucción y argumento específico a donde va ir a dar el resultado. Las referencias que en la figura se muestran como arcos unidireccionales, son utilizadoas por la instrucción que produce el resultado para almacenar una ficha de datos resultado en la instrucción consumidora. Por lo tanto se pasan los datos directamente entre instrucciones.

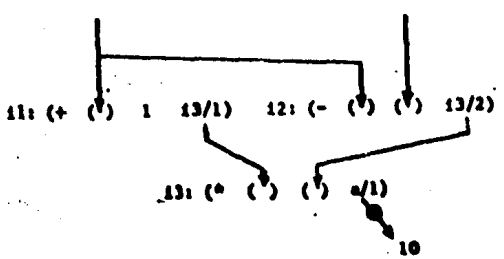
Una instrucción se vuelve factible de ejecutar cuando todos sus argumentos ya están dados, es decir, cuando todas la incognitas han sido remplazadas por resultados parciales que otras intrucciones proporcionaron. El operador entonces es

ejecutado, las entradas borradas de la memoria y la referencia incluida es usada para almacenar el resultado en el operando de otra instrucción. En términos de graficas dirigidas, una instruccion es ejecutada cuando tiene una ficha de datos en cada uno de los arcos de entrada. Durante la ejecucion el operador va removiendo fichas de arcos de entrada para soltar un conjunto de fichas en los arcos de salida.

En la figura 4.2. se mustra la secuencia de ejecucion del mismo fragmento de programa que hemos estado utilizando, indicando con un punto negro la presencia de una ficha de dato. En la parte a de la figura dos puntos negros indican la presencia de las fichas correspondientes a los valores de b y c que fueron generados por instrucciones anteriores. Como b se necesita para otras dos instrucciones, se generan dos copias de la ficha y se almacenan en localidades respectivas de cada instruccion. La disponibilidad de estas entradas completa las instrucciones de suma y resta de tal forma que quedan listas para ser ejecutadas. La ejecución procede en forma completamente independiente de tal manera que cada una consuma sus fichas de entrada y deposita sus resultados en el lugar apropiado de la instruccion i3. Esto activa a la multiplicacion que a su vez se ejecuta y almacena su resultado correspondiente al identificador a, como se muestra en la parte b de la figura.



a. Etapa 1.



b. Etapa 4.

Figura 4.2. Programa de flujo de datos para a=(b+1)*(b-c). [23]

El modelo de flujo de datos tiene varias características interesantes: 1. Los resultados parciales son pasados como fichas directamente de una instrucción a otra; 2. se pueden incluir literales en las instrucciones como una forma de optimizar el mecanismo de fichas de datos; 3. la ejecución de una instrucción consume sus datos de entrada: los valores ya no están disponibles como entradas, ni para esta instrucción ni para ninguna otra; 4. no existe el concepto de almacenamiento de datos compartido como lo es la noción tradicional de variable; y 5. el control del flujo de ejecución esta determinado por el flujo de los datos.

4.1.3. Reducción.

Los programas de flujo de control y de flujo de datos están formados por instrucciones de tamaño fijo cuyos argumentos son operandos y operadores primitivos. Se pueden formar estructuras de programas de más alto nivel a partir de secuencias lineales de estas instrucciones primitivas.

En contraste, los programas de reducción estan formados por expresiones anidadas. La analogía más cercana a una "instrucción" de reducción es una aplicación de función de la forma ⟨función⟩⟨argumento⟩ que regresa un resultado en su lugar. Se define una ⟨función⟩ o un ⟨argumento⟩ en forma recursiva como un átomo (+ ó 1, por ejemplo), o como una expresión. De la misma forma una referencia puede accesar, y una función puede regresar ya sea un átomo o una expresion. En este caso, las estructuras de programas de alto nivel aparecen como aplicaciones de funciones construidas a partir de funciones más primitivas. En reduccion la expresión 3+3 es equivalente al número 6. El requerir el resultado de la **definición** de "a", donde a=(b+1)*(b-c), significa que la referencia de "a" debe ser reescrita en una manera más simple. Debido a esto, ocurre lo que se conoce como **referenciamiento transparente** (referential transparency): solamente una definición de "a" puede ocurrir en un programa y todas las referencias a ella dan el mismo valor. Existen dos formas de reducción que se diferencian por la manera que manipulan los argumentos en un programa, llamadas reducción de cadenas (string reduction) y reducción de gráficas (graph reduction).

La base de **reducción de cadenas** es que cada instrucción que accesa una definición especifica tomará y manipulará una copia separada de la definición. La base de **reducción de gráficas** es que cada instrucción que accesa una definición particular manipulará solamente referencias a la definición. Es decir, la manipulacion de gráficas se basa en la idea de compartir argumentos a través de apuntadores.

Los principales puntos que se deben notar en reduccion son: 1. las estructuras de programas, las instrucciones y los argumentos son todos expresiones; 2. No existe el concepto de almacenamiento que se puede ir actulizando como son las variables; 3. no existen más limitantes sobre el flujo de la ejecucion que los implicados por la demanda de operandos; y 4. las demandas pueden regresar tanto argumentos sencillos como funciones complejas.

Es claro que existen diferencias fundamentales entre flujo de control, flujo de datos y reduccion que se relacionan con sus ventajas y desventajas para representar programas. Sin embargo, tambien tienen elementos interesantes en comun. En las siguientes dos secciones trataremos de resumir las ideas presentadas en [23] que pretenden identificar y clasificar las diferencias y similitudes entre estos tres modelos.

4.2 Organización del cómputo

En esta seccion analizaremos a un nivel abstracto la forma en que procede el cómputo. Este proceso toma la forma de cambios sucesivos en el estado del cómputo de acuerdo a las instrucciones que se van ejecutando. La organización del cómputo describe la manera en que ocurren estos cambios analizando el efecto que producen las instrucciones. Esta forma abstracta de clasificación permite notar claramente las diferencias entre los términos: acción por control, acción por datos y acción por demanda. Estas tres clases se asocian respectivamente con los modelos operativos de flujo de control, flujo de datos y reducción.

4.2.1 Clasificación.

Las organizaciones computacionales se pueden clasificar considerando el cómputo como una repetición continua de tres fases: selección, examen y ejecución.

1. **Selección.** En esta fase se escoje un conjunto de instrucciones para su posible ejecucion. Solamente se pueden ejecutar las instrucciones que hayan sido seleccionadas en esta etapa, pero la selección no garantiza la ejecución. A la regla mediante la cual se elijen las instrucciones se la llama **regla computacional** (computation rule). Las tres reglas que se usan en la clasificacion son: imperativo, "de más adentro" (innermost) y "de más afuera" (outermost). La regla imperativa selecciona las instrucciones indicadas por un registro especial (PC) o por la presencia de fichas de control. Esta selección se hace independientemente de la posición de la instrucción dentro de la estructura del programa. Las reglas de más adentro y de más afuera seleccionan, respectivamente, las instrucciones más profundas y menos profundas dentro del programa. La más profunda tiene como argumentos valores (no instrucciones) mientras que las menos profunda selecciona una instrucción que no sea un argumento de ninguna otra. En la figura 4.3. se puede ver cual sería la instruccion seleccionada por cada una de las reglas:

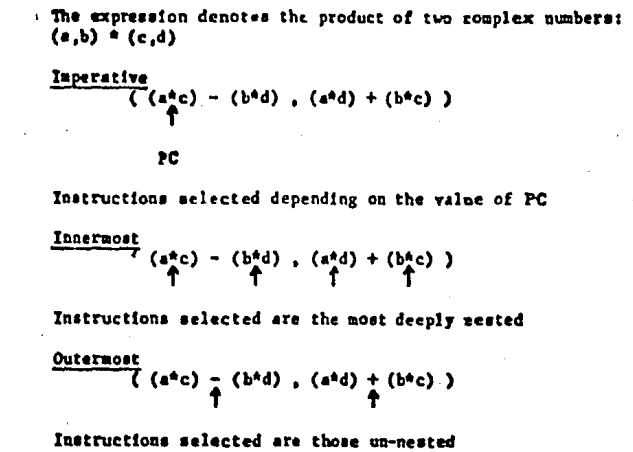


Figura 4.3. Aplicacion de tres reglas computacionales a una expresión.[23]

2. **Examen.** En la fase de examen se analiza cada instruccion previamente escogida en la fase de seleccion para determinar si es ejecutable. La decisión se toma en base al examen de cada uno de los argumentos reales de la instrucción. A la regla para tomar esta decisión se la llama **regla de disparo** (firing rule). Por ejemplo, la regla de disparo puede requerir que todos los operandos sean valores, o puede requerir que solamente uno lo sea como en el caso de una instruccion condicional. Si la instrucción es ejecutable se pasa a la siguiente fase para ejecución; de otra forma la fase de examen puede realizar alguna otra acción como detener temporalmente la ejecución de la instruccion o tratar de obtener los argumentos faltantes para permitir su ejecución.

3. **Ejecución.** Esta fase es muy parecida en todas la computadoras: las instrucciones son ejecutadas propiamente. Los resultados disponibles se mandan a otras partes del programa. La ejecucion puede producir cambios que se perciban globalmente, quizas cambiando el estado de una memoria general compartida, o puede provocar cambios generales, como cuando una expresión se reemplaza por su valor.

4.2.2 Flujo de control.

En la fase de selección, cada PE tiene un contador de programa que es el que indica la proxima instruccion a ejecutar. Una vez seleccionadas las instrucciones a ejecutar no son pasadas por la fase de examen sino directamente a la fase de ejecución. La fase de ejecucion puede cambiar el estado de cualquier parte de la computadora. El estado está representado por la memoria y

por los registros de los procesadores. Los PC's se alteran explícitamente o implícitamente mediante instrucciones GOTO.

Definimos el término **acción por control** para denotar las organizaciones de computadoras en las que las instrucciones son ejecutadas tan pronto como son seleccionadas.

4.2.3 Flujo de datos.

En el esquema "puro" de flujo de datos, se ejecutan las instrucciones tan pronto como todos sus argumentos estan disponibles. Cada instrucción tiene asociado, al menos conceptualmente, un PE listo para ejecutarla en el momento en que sus argumentos lleguen. De esta manera se puede ver a la fase de seleccion como en la que se proporciona un PE logico a cada instrucción. En la fase de examen se implementa la regla de disparo de flujo de datos, que indica que una instrucción se puede ejecutar solamente cuando tiene listos todos sus argumentos. Si la instrucción no está lista para su ejecución es detenida en esta fase hasta que lo esté. La fase de ejecución cambia el estado local de la instrucción y de todas sus sucesoras; consume sus argumentos y coloca el resultado en cada una de las instrucciones que lo esten esperando.

Definimos el término **acción por datos** para denotar las organizaciones de computadoras en las que las instrucciones esperan pasivamente a que les llegue alguna combinación de sus argumentos.

4.2.4 Reducción.

En computadoras de reducción pueden existir diferentes reglas de cómputo, pero la que se usan más comunmente son la de mas adentro y la de más afuera. Esta regla es la que determina a que instrucciones, al principio de cada ciclo, darles un procesador. En la fase de examen se analizan los argumentos de la instrucción para ver si es posible su ejecución. Si lo es, la instruccion se ejecuta. En otro caso, se trata de forzar la obtencion de los argumentos, es decir, demanda la evaluacion de argumentos hasta obtener un numero suficiente de ellos para su ejecución. La fase de ejecución implica el reescribir la instruccion en su mismo lugar (se reemplaza por su resultado). Se cambia solamente el estado local correspondiente a la instruccion y a las instrucciones que utilizan su resultado. Por lo tanto la ejecucion puede habilitar otra instrucción.

Definimos el término **acción por demanda** para denotar a las organizaciones de computadoras cuando se seleccionan instrucciones únicamente en el caso en el que el resultado que generan es requerido por otra instruccion que ya haya sido seleccionada.

4.2.5 Implicaciones.

A continuacion resumimos las implicaciones de la clasificacion de organizaciones de computadoras que acabamos de describir.

La ventaja de acción por control es que se tiene control

total sobre el orden de ejecución de las instrucciones. La desventaja es la disciplina de programación que se necesita para evitar errores a la hora de ejecucion.

La ventaja de acción por datos es que las instrucciones se ejecutan tan pronto como sus argumentos estan disponibles, dando esto un alto grado de paralelismo implícito. La desventaja es que puede haber instrucciones que esten esperando por argumentos que no sean necesarios.

La ventaja de acción por demanda es que solamente se ejecutan las instrucciones cuyos resultados son necesarios. La desventaja es que, en las situaciones en que siempre son necesarios todos los resultados (expresiones aritmeticas por ejemplo) se desperdicia esfuerzo propagando la demanda desde afuera hasta adentro.

4.3 Organización del Programa

El término organizacion del programa se refiere a la manera en que el código del programa se representa y ejecuta en una computadora. Comenzamos la sección con una clasificacion de los mecanismos de fondo comunes a los tres modelos: flujo de control, flujo de datos y reducción.

4.3.1 Clasificación.

Existen dos mecanismos fundamentales para los tres modelos. El **mecanismo de datos** define el modo de utilizar un argumento por un conjunto de instrucciones. Hay tres subclases:

1. Por literal: Una copia del argumento que se conoce a tiempo de compilacion, se coloca directamente en cada una de las instrucciones que lo utilizan.
2. Por valor: Copias del argumento generado a tiempo de ejecución, se colocan en cada una de las instrucciones que lo accesan.
3. Por referencia: La manera de compartir un argumento es teniendo una referencia a él.

El **mecanismo de control** define la manera en que una instrucción causa la ejecucion de otra por lo tanto, definiendo el patron de control de todo el programa. Nuevamente tenemos tres subclases:

1. Secuencial: Un solo flujo de control indica las instrucciones que deben irse ejecutando.
2. Paralelo: El control indica la disponibilidad de argumentos, y una instrucción se ejecuta cuando tiene todos sus argumentos listos.
3. Recursivo: El control indica la necesidad de argumentos y una

instruccion se ejecuta cuando alguna otra necesita su resultado. Una vez ejecutado devuelve el control a la instrucción que lo invoco.

En la figura 4.4 se resume la relacion entre estos mecanismos de datos y de control con los tres modelos operacionales. Utilizando esta clasificacion como base presentamos a continuación ventajas y desventajas de la representación y de la ejecucion de programas de flujo de control, flujo de datos y reduccion.

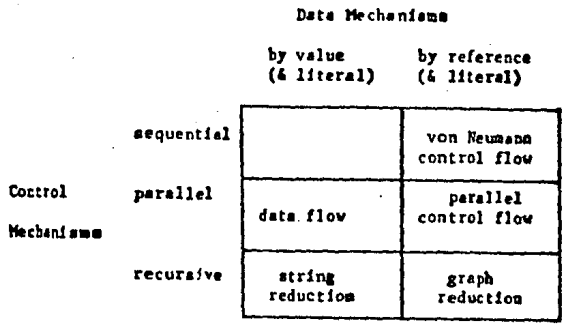


Figura 4.4. Modelos computacionales: Mecanismos de control y de datos.[23]

4.3.2 Flujo de control.

El flujo de control está basado en un mecanismo de control secuencial o paralelo. El flujo de control es implicitamente secuencial aunque puede ser alterado por mecanismos explicitos paralelos como son, respectivamente, GOTO y FORK-JOIN. El mecanismo basico de datos es por referencia.

En computadoras basadas en organizaciones de programas paralelos, como flujo de control paralelo, se necesitan proveer mecanismos de sincronizacion adecuados para evitar indeterminaciones no deseadas. Esto es, básicamente, sincronizar el uso de recursos compartidos. En flujo de control paralelo se utilizan dos mecanismos importantes: iteracion y procedimientos.

Estos dos esquemas se ilustran en las figuras 4.5 y 4.6 respectivamente con los siguientes programas para calcular el término 100 de los numeros de Fibonacci:

Iterativo:

```
(f1,f2) := (1,1);
FOR i:=3 TO 100 DO
  (f1,f2) := (f2,f1+f3)
END-FOR;
answer := f2;
```

Recursivo:

```
fib(f1,f2,i) := IF i>100
                THEN f2
                ELSE fib(f2,f1+f2,i+1)
                END-IF;
answer := fib(1,1,3);
```

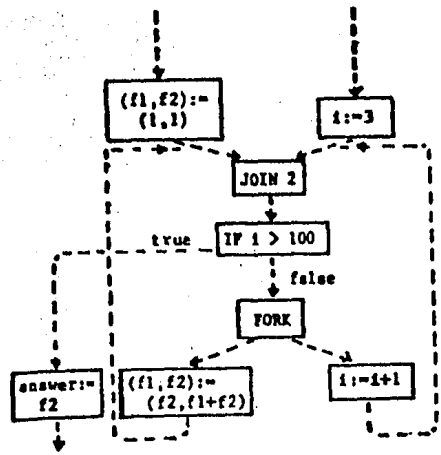


Figura 4.5. Iteracion usando retroalimentacion.[23]

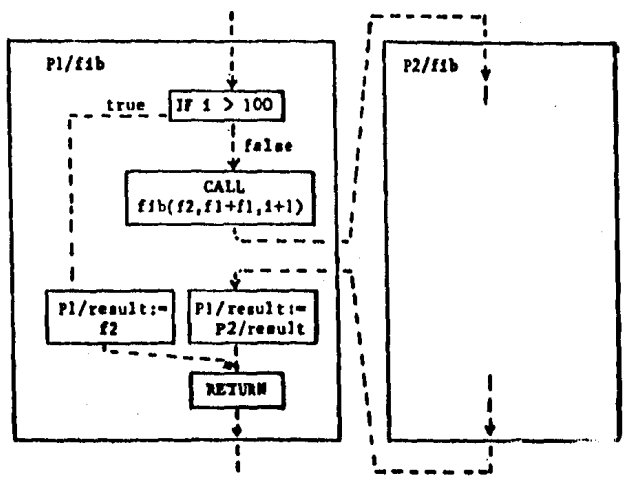


Figura 4.6. Iteracion usando recursion.[23]

La herramienta básica del caso iterativo es la utilización de las instrucciones FORK y JOIN. La instruccion FORK indica que el flujo de control se debe bifurcar mientras que la instrucción JOIN une varios flujos de control esperando hasta que terminen de ejecutarse las instrucciones de cada uno de ellos.

En el caso iterativo se logra sincronización proporcionando identificadores nuevos a cada nueva invocacion del procedimiento.

4.3.3. Flujo de datos.

Flujo de datos se basa en un mecanismo de datos por valor y un mecanismo de control paralelo basado en fichas de datos. Cuando una instruccion se ejecuta, la referencia del resultado debe ser de la siguiente forma: (P / N / A)
donde cada uno de los campos se utiliza de la siguiente manera:

1. El campo de proceso (P) distingue instancias diferentes de la instrucción N que pueden estar siendo ejecutadas en paralelo.
2. El campo de instrucción (N) identifica la instruccion que tomará el dato.
3. El campo de argumento (A) indica la posición dentro de la instruccion en la que se colocará la ficha.

Describiremos a continuación dos esquemas de sincronización utilizados en flujo de datos para iteraciones. El primer esquema se muestra en la figura 4.7. y se basa en la realimentación controlada por la instrucción SYNCH. Esta instruccion deja pasar las fichas de datos solamente hasta que haya una presente en cada una de sus entradas. Se utiliza tambien la instrucción SWITCH que manda el dato por uno de dos caminos dependiendo del valor de su entrada para datos logicos.

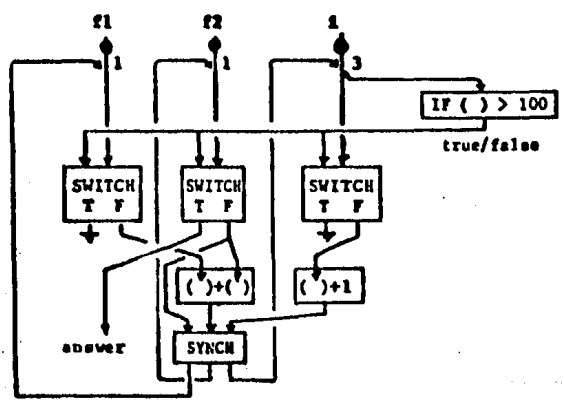


Figura 4.7. Iteración de flujo de datos utilizando retroalimentación de fichas.[23]

El segundo esquema para lograr iteraciones es basado en invocaciones concurrentes de procedimientos atraves del uso del campo de proceso P. En la figura 4.8. se ilustra el sistema para el mismo programa.

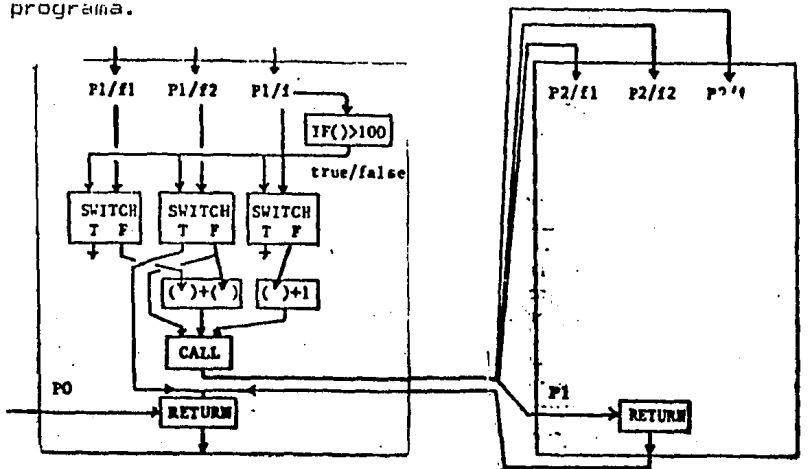


Figura 4.8. Iteración de flujo de datos usando recursion.[23]

4.3.4. Reducción.

El modelo de reducción se basa en un mecanismo de control recursivo y un mecanismo de datos por valor o por referencia. **Reducción de cadenas** utiliza el mecanismo por valor y **reducción de gráficas** el mecanismo por referencia. En el primer caso se reducen copias de la instruccion en tanto que en el segundo caso las subexpresiones se comparten mediante referencias. Como el modelo de reduccion es inherentemente recursiva, solamente mostramos la versión recursiva del programa de los números de Fibonacci.

En la figura 4.9. se mustra el caso de reduccion de cadenas. Las expresiones contienen solamente literales y valores. En la figura se utilizo la regla del de mas adentro.

El modelo de reduccion de cadenas se adapta mejor a reglas del de más adentro ya que como en este esquema se copian realmente los parámetros, no es eficiente utilizar la regla del de más afuera.
de más afuera.

Los programas de reduccion de gráficas son expresiones que contienen literales, valores y referencias. Los parámetros son sustituidos por referencias a funciones definidas. Debido a este mecanismo de referencias, en la figura 4.10. se utiliza una notacion de graficas para mostrar el programa de Fibonacci. Los nodos representan funciones y sus argumentos, mientras que los arcos representan referencias y estructuración. En el ejemplo se usa la regla de computacion del de mas afuera. La perdida de eficiencia que se puede tener usando esta regla de reducion de cadenas no ocurre acá debido a que el mecanismo de referencias permite compartir expresiones. Si se hubiera utilizado la regla del de mas adentro no se hubiera aprovechado el mecanismo de

referencias de reducción de gráficas ya que se hubieran reducido todas las expresiones antes de referenciarlas; las únicas referencias serían a valores. Por esta razón se utiliza la regla computacional del de más afuera para reduccion de graficas.

```
Initial Expression:

 ( answer ) WHERE
                    answer = fib (1, 1, 3);
        fib (f1, f2, i) = IF i > 100 THEN f2
                                     ELSE fib (f2, f1+f2, i+1) FI;
First Reduction:

 ( IF 3 > 100 THEN 1
              ELSE fib (1, 1+1, 3+1) FI )

Next Reductions:

 ( IF FALSE THEN 1
            ELSE fib (1, 1+1, 3+1) FI )

 ( fib (1, 1+1, 3+1) )

 ( fib (1, 2, 4) )

 ( fib (2, 3, 5) ) ... ( fib (3, 5, 6) ) ... ( fib (5, 8, 7) )
```

Figura 4.9 Reducción de cadenas para el programa de Fibonacci.[23]

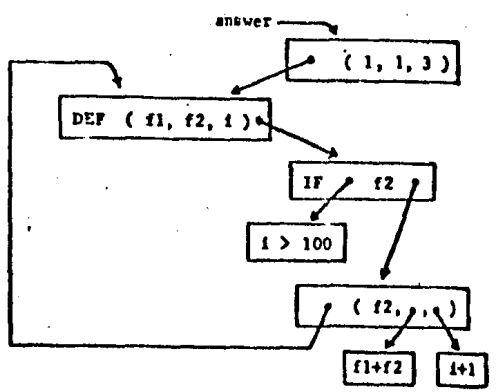


Figura 4.10. Reduccion de gráficas para el programa de Fibonacci.[23]

4.3.5 Implicaciones.

Las organizaciones de programas de flujo de control tienden a ser menos eficientes que las de flujo de datos para evaluar expresiones sencillas; para manipular estructuras de datos complejas, que deben ser manipuladas en su lugar, flujo de control presenta ventajas. Como organización de propósito general, flujo de control es sorprendentemente flexible. Las críticas son debidas basicamente a la falta de propiedades matemáticas para razonar acerca de programas, dificultades para aprovechar paralelismo y a que está basado en conceptos de bajo nivel.

La mayor ventaja de flujo de datos es su simplicidad y la naturaleza altamente paralela de la organización de sus programas. Sin embargo, cuando se requieren patrones de control específicos o manipulación de estructuras de datos complejas, el flujo de datos se encuentra en desventaja.

Las organizaciones de reducción de cadenas y gráficas dan un excelente apoyo a la programación funcional que está tomando cada vez mayor interés. La reduccion de gráficas permite la manipulación de objetos arbitrariamente complejos. Sin embargo, para que las organizaciones de programa de reducción sean aceptadas como candidatos para cómputo de propósito general, es necesario que la programacion funcional se vuelva un estilo de programación usado comunmente.

4.4 Organización de la Máquina.

Utilizamos el termino organizacion de la maquina para describir la manera en que los recursos de la maquina estan configurados para apoyar una organizacion de programa.

4.4.1 Clasificación.

Si examinamos las arquitecturas de las computadoras de accion por datos y de accion por demanda que se están desarrollando, podemos descrubrir tres clases básicas de organizaciones: centralizadas, de comunicacion por paquetes y de manipulacion de expresiones.

1. Centralizada. En la figura 4.11 se muestra una organizacion centralizada. Consiste de un procesador, memoria y comunicaciones.

2. Comunicacion por paquetes. Estas organizaciones consisten de un pipeline circular en el que están los procesadores, comunicaciones y memoria interactuando atraves de "albercas" (pools) de trabajo. Esto se ilustra en la figura 4.12.

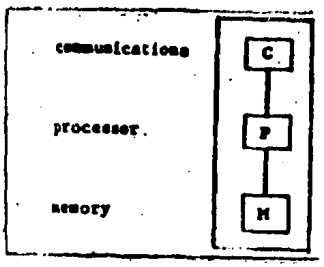


Figura 4.11. Organizacion de maquina centralizada.[23]

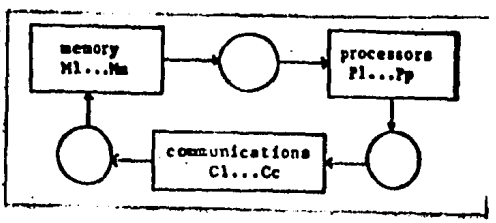
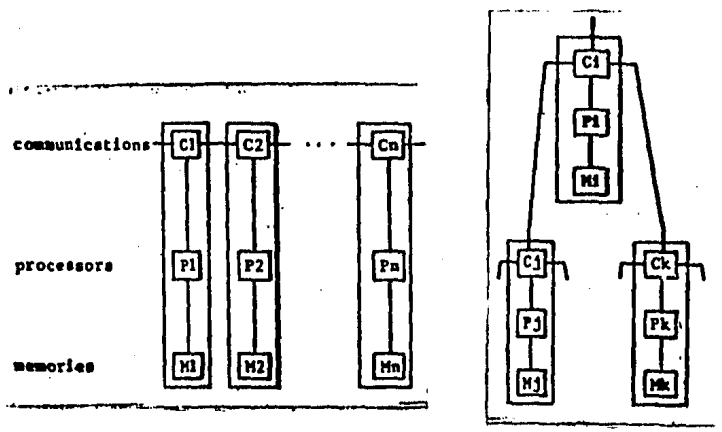


Figura 4.12. Organizacion de máquina de comunicación por paquetes.[23]

Esta organizacion ve al programa en ejecucion como un numero de paquetes de información independientes, todos los cuales están activos. Cada paquete a ser procesado se coloca junto con otros paquetes similares en las albercas de trabajo. Cuando un recurso se queda sin trabajo toma un paquete de su alberca de entrada, lo procesa, coloca el paquete modificado en su alberca de salida y regresa al estado de desocupado. Se logra paralelismo teniendo varios recursos identicos entre albercas o replicando los pipelines circulares interconectandolos mediante las comunicaciones.

3. Manipulacion de expresiones. Las organizaciones de manipulacion de expresiones consisten de identicos recursos organizados en estructuras regulares como arboles o vectores, como se muestra en la figura 4.13. Cada recurso contiene un procesador, comunicaciones y memoria. La organización ve al programa en ejecucion como una gran estructura anidada en la que algunas partes estan activas mientras que otras no lo estan, temporalmente.



a. vector. b. árbol.

Figura 4.13. Organización de máquina de manipulación de expresiones: a. vector, b. árbol.[23]

4.4.2 Flujo de control.

El medio más obvio para implementar el flujo de control es, para las formas secuenciales, una organización centralizada, mientras que para las formas paralelas, ya sea una organización de paquetes o una de manipulación de expresiones.

4.4.3 Flujo de datos.

Para implementar flujo de datos la organización más apropiada es la comunicación de paquetes. La ejecución de instrucciones esta sincronizada, en general por el mecanismo de **almacenamiento de fichas** y por el de **igualación de fichas** (token storage y token matching). En el primer esquema las fichas son realmente almacenadas en la instrucción y se ejecuta la instrucción cuando han recibido todos sus argumentos. Las computadoras de flujos de datos Massachusetts Institute of Technology y Texas Instruments son dos ejemplos de este esquema. En el segundo esquema se van formando conjuntos con fichas de datos y cuando este completo el conjunto de una instrucción se activa esta. Tres ejemplos de este esquema son la Irvine Data Flow, la Manchester Data Flow System y la Newcastle Data-Control Flow Computer.

En las figuras 4.14 y 4.15 se muestran organizaciones de comunicaión de paquetes basadas en estos dos esquemas. Las dos organizaciones se asemejan en que tienen varios elementos procesadores que evaluan asincronamente paquetes de instrucciones ejecutables. Estos paquetes contienen toda la información requerida para procesar la instrucción y distribuir los resultados: el código de operacion, las variables de entrada y las referencias para las fichas de resultado.

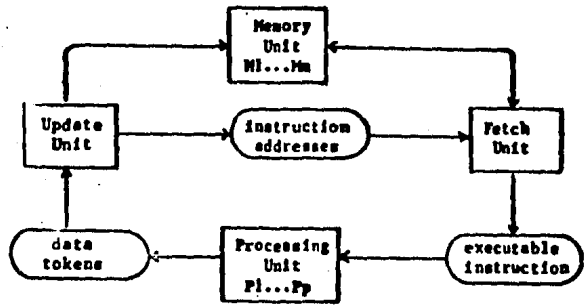


Figura 4.14. Comunicación por paquetes con almacenamiento de fichas.[23]

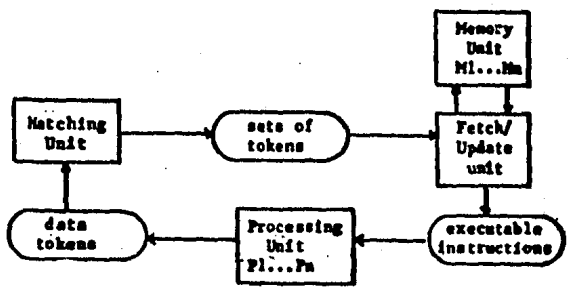


Figura 4.15. Comunicación por paquetes con igualacion de fichas.[23]

4.4.4 Reducción.

En las computadoras de reducción la ejecucion de las instrucciones se basa en el reconocimineto de expresiones reducibles y transformación de estas expresiones. La ejecución se logra mediante un proceso de sustitución que recorre la estructura del programa y reemplaza sucesivamente expresiones reducibles por otras que tienen el mismo significado, hasta que se llega a una expresion constante que representa el resultado del programa.

Hay dos problemas básicos en la implementación de este esquema: primero administrar dinamicamente la memoria del programa que se esta transformando y segundo, mantener información de control acerca del estado de la transformacion. Algunas soluciones al problema de la administración de la memoria son 1. Representar el programa y las instrucciones como cadenas, por ejemplo, "(((a)(b))((c)(d)))", que puede ser

expandida o comprimida sin alterar su significado y 2. Representar el programa como una estructura de gráficas con apuntadores y recoleccion de basura. Soluciones al problema de control son 1. Usar pilas de control que indican, por ejemplo, los ancestros de una instrucción; es decir, las que demandaron su ejecución y 2. Cambiar el sentido de los apuntadores, donde el ancestro esta definido por un apuntador en sentido opuesto almacenado en la instruccion.

En general, las organizaciones de manipulacion de expresiones parecen ser más apropiadas para implementar los programas de reducción, aunque en algunos casos se utilizan organizaciones centralizadas (GMD Reduction Machine, Cambridge SKIM Machine). Tambien se han utilizado organizaciones de comunicación de paquetes para implementar reducción. Un ejemplo de estas organizaciones de reduccion de gráficas mediante acción por demanda es la Utah Applicative Multiprocessing System. Finalmente tenemos maquinas como la Newcastle Reduction Machine y la North Carolina Cellular Tree Machine que se basan en organizaciones de manipulación de expresiones.

4.4.5 Implicaciones.

De la discusión anterior vemos que flujo de control, flujo de datos y reducción gravitan respectivamente hacia organizaciones centralizadas, de comunicacion por paquetes y de manipulacion de expresiones. Sin embargo vimos que hay casos que no corresponden a esta relación.

El flujo de control puede ser implementado por cualquiera de las tres organizaciones, sin embargo, para el tipo secuencial la organización centralizada es la mejor. La organización de comunicación por paquetes favorece a la forma paralela con fichas de control mientras que para el tipo FORK-JOIN la de manipulación de expresiones es la organización más adecuada.

Para flujo de datos, el esquema de comunicación por paquetes provee dos alternativas de comunicacion. La primera (estática) consiste en almacenar fichas de datos en las instrucciones y ejecutarlas cuando estan completas. El segundo esquema (dinámico) se basa en comparar fichas de datos. También mediante manipulación de expresiones se puede implementar flujo de datos, pero es difícil evaluar las ventajas y desventajas de este enfoque.

Finalmente vemos que reducción se puede implementar eficientemente por cualquiera de los tres métodos. Una organización secuencial es más adecuada para la forma secuencial de reducción y puede implementar con eficiencia razonable tanto manipulación de cadenas como de gráficas. Las organizaciones de comunicación por paquetes favorecen una regla computacional paralela.

PARTE II: **Flujo de Datos**

4.5 Lenguajes de Flujo de Datos

La meta principal de los arquitectos de multiprocesadores, de procesadores de arreglos, de computadoras pipeline y de vectores es explotar paralelismo. A lo largo de los años se han diseñado compiladores que tratan de extraer paralelismo de programas escritos en lenguajes convensionales (como compiladores vectorizadores de Fortran, ver capitulo 2.). Se han desarrollado también lenguajes para facilitar el uso de estos sistemas paralelos, como Pascal concurrente. Se ha llegado aun a diseñar lenguajes especializados para aprovechar arquitecturas de computadoras especificas como Glypnir para la Illiac IV. La mayoria de estos intentos han resultado ser poco naturales en el sentido de que reflejan más caractristicas del sistema, que la manera en que los programadores piensan al resolver los problemas.

Las computadoras de flujo de datos también buscan explotar paralelismo y lo hacen no solo a nivel microscopico sino también a nivel mucho más alto. Al igual que otras formas de computadoras paralelas, las computadoras de flujo de datos son mejor aprovechadas si son programadas en lenguajes especiales. Sin embargo, los lenguajes para las computadoras de flujo de datos pueden ser muy elegantes. Las propiedades del lenguaje que requieren estas computadoras son en si mismas benéficas y muy similares a algunas de las propiedades que se saben facilitan el desarrollo de software claro y facil de mantener. Algunas de las propiedades relevantes de los lenguajes de flujo de datos son:

1. Son libres de efectos secundarios.
2. Localidad de efectos.
3. Toda la información requerida para la ejecución del programa esta contenida en su gráfica de flujo de datos.
4. Asignación única.
5. Una notación poco usual para iteraciones, requerida por 1 y 4.
6. Los procedimietos no tienen variables que retengan datos de una invocación a otra; es decir, no tiene memoria.

Esta sección, basada en [24], toca dos lenguajes de flujos de datos: Val (Value Algorithmic Language) y Id (Irvine Dataflow) que fueron desarrollados en MIT y en la Universidad de California en Irvine, respectivamente.

4.5.1 Análisis de flujo de datos.

Si examinamos el siguiente fragmento de programa podemos observar ciertas dependencias entre instrucciones causadas por el flujo de los datos.

```
1   P = X + Y
2   Q = P / Y
3   R = X * P
4   S = R - Q
5   T = R * P
6   RESULT = S / T
```

En la figura 4.16. se ilustra como mediante una gráfica se pueden mostrar estas dependencias. En la gráfica los nodos representan operaciones cuyos argumentos entran por arcos de entrada y cuyo resultado sale por arcos de salida. En la figura 4.16.b. se bautizan los arcos con el nombre de las variables que representan. En el momento en el que un nodo (operador) tiene datos listos en cada una de sus entradas, realiza la operacion indicada sobre los datos, consumiendolos, para mandar el resultado por los arcos de salida (si son mas de uno los arcos de salida se generan varias copias del resultado).

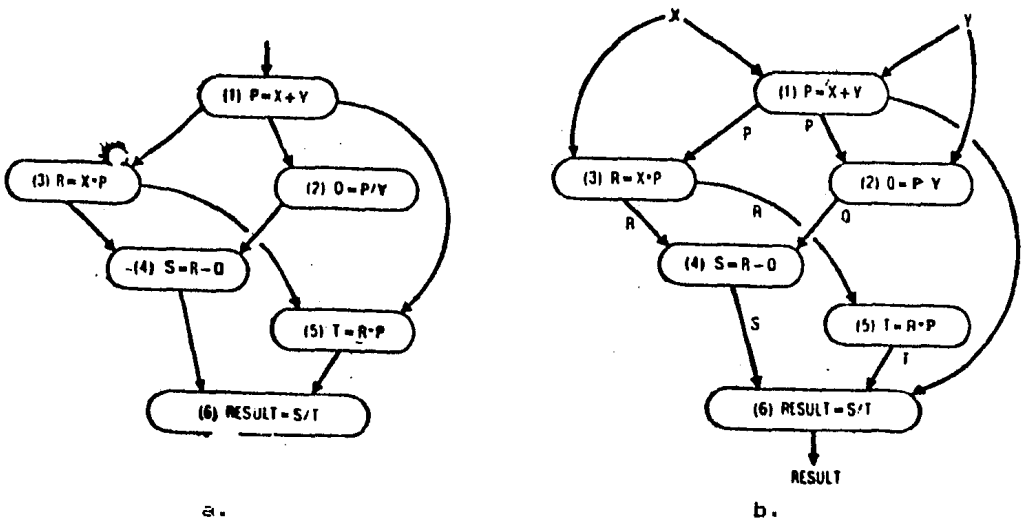


Figura 4.16. Gráfica de flujo de datos para el programa en Fortran.[24]

El lenguaje de programación para una computadora de flujo de datos debe satisfacer dos criterios: debe ser posible deducir las dependencias de los datos a partir de las operaciones del programa; y las restricciones de secuenciación deben ser exactamente las mismas que las dependencias de los datos. Un lenguaje que tiene las propiedades de localidad de efecto y de libertad de efectos secundarios puede satisfacer estos criterios.

4.5.2 Localidad de efectos.

Localidad de efecto quiere decir que las instrucciones no tienen dependecia de datos de largo alcance inecesarias. Por ejemplo, el fragmento de programa anterior parece utilizar las variables P, Q, R, S y T como temporales solamente. Sin embargo, algún otro fragmento en otro lugar del programa podria utilizar las mismas variables para algun otro computo sin relación con el primero y la logica del programa permitiria la ejecución concurrente de los dos fragmentos si no fuera por la duplicación de nombres de variables. Existen compiladores que pueden hacer analisis de este tipo de dependencias aunque se trata de una tarea dificil; en los lenguajes de flujo de datos no se presenta este problema.

4.5.3 Efectos secundarios.

Es necesario que el programa este libre de efectos secundarios para asegurar que las dependencias de datos sean las mismas que las restricciones de secuenciación. Es mucho más dificil de lograr que la localidad de efectos. La forma más conocida de efectos secundarios aparece en procediminetos que modifican variables en el programa que los invoca. En un programa de flujo de datos esto no puede suceder; un procedimiento no puede modificar ni siquiera sus propios argumentos. Una manera de lograr esto es usar siempre llamadas por valor y nunca por referencia, aun en el caso de estructuras de datos.

4.5.4 Lenguajes aplicativos.

Para los lenguajes de flujos de datos se utiliza un esquema que va mucho más alla que llamadas por valor: todos los arreglos son valores en lugar de objetos y son tratados como tales todo el tiempo, no solo cuando son pasados como argumentos en procedimientos. Los arreglos nunca se modifican mediante instrucciones como " A(j) = S", sino mediante instrucciones que generan nuevos valores de arreglos. En el lenguaje Val se utilizan instrucciones de la forma " A[j:S]" cuyo resultado es un nuevo arreglo conteniendo el valor de S en la posicion j del arreglo; **el valor de A no se modifica.** De esta manera, en el programa Val:

```
1  B := A[J:S];
2  C := A[K:T];
```

las instrucciones 1 y 2 no se interfieren entre si o con el arreglo A.

Las restricciones de secuenciacion son exactamente las mismas que las dependencias de datos. Instrucciones como "A:=A[J:S]" no son permitidas.

En lenguajes convencionales, las subrutinas hacen la mayoria de su trabajo atraves de efectos secundarios. Las funciones por otro lado, tipicamente regresan un solo valor que normalmente no es un arreglo. Para hacer a las funciones más poderosas y flexibles, los lenguajes aplicativos frecuentemente

permiten regresar varios valores, arreglos o registros (records).

Los lenguajes que realizan todo el procesamiento por medio de operadores aplicados a valores se llaman lenguajes aplicativos y son los lenguajes naturales para los computadoras de flujo de datos. LISP es el más conocido de los lenguajes aplicativos.

4.5.5 Lenguajes definicionales y la regla de asignación única.

Una vez aceptado el estilo de programación aplicativo y el modelo de ejecucion enfocado a valores más que objetos, analizaremos las aplicaciones de este estilo que se relacionan con el significado de los enunciados de asignación. Excepto en las iteraciones (que serán discutidas más adelante), el efecto de una asignación no tiene otro significado que el de proveer un valor y asociarlo al nombre que aparece del lado izquierdo de la asignacion. El resultado de esta asignacion queda disponible solamente para expresiones posteriores en las que este nombre aparece.

Concretamente, un enunciado como "S:=X + Y" debe ser pensado como una definicion y no como una asignacion. Los lenguajes que utilizan esta interpretacion se llaman **lengujes definicionales** (definitional languages) en contraste con los lenguajes convencionales imperativos. Los lenguajes definicionales son muy utiles para verificación ya que las hipótesis (assertions) hechas para probar correccion (correctness) son exactamente las mismas que las definiciones que aparecen en el programa.

Hay un problema que podria destruir la elegancia de los lenguajes definicionales: definiciones multiples del mismo nombre. En casi todos los casos, los lenguajes definicionales obedecen la regla de asignacion unica que restringe la utilizacion de aberraciones matematicas como "J=J+1". Como la aparicion de la J del lado derecho precede la definicion de J esto implica una inconsistencia que el compilador diagnosticaria.

4.5.6. Iteraciones.

A pesar de la discusion previa, existe una area en la que las instrucciones como " I=I-1" parecen ser necesarias: la de iteraciones.

Una iteración consiste de

1. Definicion de los valores iniciales del ciclo.
2. Una prueba para determinar, para cualesquiera valores de las variables si hay que terminar el ciclo o iniciar una nueva iteracion.
3. Si debe terminar, una expresion que proporciona los valores a regresar y
4. Si debe iterar una vez mas, una expresion que de los nuevos valores a asignar a las variables del ciclo.

Veamos un ejemplo de como resuelve el problema Val en el siguiente ciclo:

```
for J,K := N,1; % inicializa J y K con N y 1
                % respectivamente.
do if J=0       % decide si terminar
   then K       % si, el resultodo final es K
   else iter J,K := J-1, K*J;
                % no, calcula nuevos valores para J y K
   end          % una nueva iteracion
end
```

A pesar de que las variables del ciclo si cambian, lo hacen solamente en el limite entre una iteracion y la siguiente. A lo largo de cada una de las iteraciones nunca se presentan cosas como "J = J-1 ". Val refuerza esto permitiendo redefiniciones solamente despues de la palabra iter que es el comando para comenzar una nueva iteración.

4.5.7 Conclusión.

Existe la tendencia de abandonar las arquitecturas convencionales de von Neumann para aprovechar la nueva tecnologia VLSI. Tambien, existe un reconocimineto de que la mayoria de los lenguajes existentes no son apropiados para aprovechar las nuevas arquitecturas.

Las implicaciones de estas dos tendencias son similares: los lenguajes deben evitar modelos de ejecucion que tomen como base una memoria global cuyo estado es manipulado por la ejecución secuencial de comandos. Esta memoria global tiende a crear un cuello de botella en el sistema ademas de la posiblidad de crear módulos del programa que interactuen en formas difíciles de entender. Los lenguajes basados en conceptos de programación aplicativa tienen grandes posiblidades de resolver estos problemas en los diseños futuros de computadoras.

4.6 Gráficas de Programas Para Flujos de Datos.

Los lenguajes de fluio de datos forman una subclase de los lenguajes aplicativos. Un lenguaje de flujo de datos es un lenguaje aplicativo basado totalmente en la nocion de datos fluyendo de una funcion a otra o un lenguaje que soporta directamente este fluir. Este concepto de flujo les da a los lenguajes de flujo de datos la ventaja de poderlos representar mediante gráficas. Esta seccion, basada en [25], tiene como objeto introducir al lector a las representaciones gráficas y sus aplicaciones.

Los lenguajes aplicativos proveen los beneficios de alta modularidad. Cuando los programas son representados en forma grafica, los subprogramas que parecen independientes, pueden ser ejecutados independientemente y por lo tanto concurrentemente. Existen muchas razones para describir a los lenguajes de flujo de datos mediente representaciones graficas incluyendo las siguientes:

1. La secuencia de las acciones de un programa de flujo de datos esta goberneda por la siguiente regla sencilla de disponibilidad: cuando los argumentos de una instrucción estan disponibles, se dice que la instruccion esta lista para ejecutarse (firable). Cuando se ejecuta, los argumentos son absorbidos y los resultados mandados a otras instrucciones.

La imagen mental de una grafica dirigida es sugerida por este comportamiento en la que un nodo representa cada funcion y cada flecha o arco dirigido representa un medio conceptual por el que fluyen los datos. Los nodos fantasmas, representados por lineas punteadas, indican puntos en los que el programa se comunica con su entorno ya sea recibiendo o mandando datos a el.

2. Los programas de flujos de datos se pueden facilmente juntar para formar programas mas grandes. La interaccion entre componentes se puede apreciar claramente en las graficas de flujos de datos tanto en cuanto a como se pasa de un lado a otro el flujo de control, como en cuanto a la informacion que cruza de un componente a otro (ver figura 4.17).

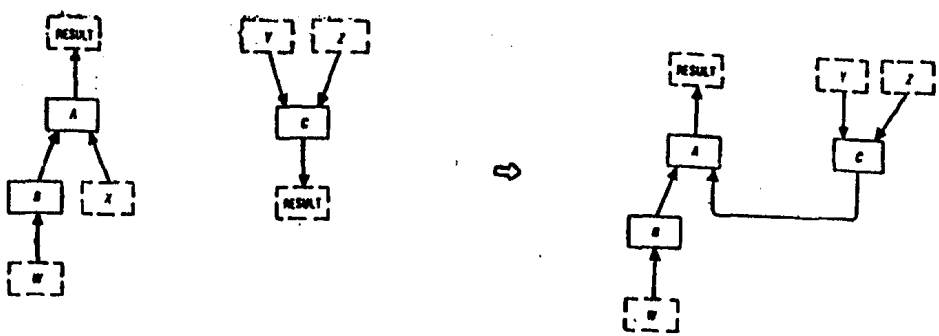


Figura 4.17. Union de dos graficas de flujo de datos.[25]

3. Los programas de flujos de datos evitan prescribir el orden especifico de ejecución inherente en programas tradicionales basados en asignaciones. En las graficas se muestra claramente la concurrencia potencial de la ejecucion de un programa.
4. Se pueden utilizar graficas para atribuirle significado formal a un programa. El significado puede tomar la forma de una definicion operacional o de una funcional. La primera define la secuencia de operaciones permisibles al ejecutar el programa. La segunda describe una sola función representada por el programa y es independiente de cualquier modelo de ejecucion.

El objetivo de esta seccion es explorar la utilidad de las representaciones graficas de programas para flujo de datos incluyendo la posibilidad de prescindir totalmente del texto y de la grafica como el programa. La discusion se enfoca en dos modelos para representacion de flujo de datos: el modelo de fichas (tokens) y de estructuras (structures). Los terminos clasifican a los lenguajes de flujos de datos que se desarrollaron a lo largo de dos distintas lineas.

4.6.1. Modelos de fichas.

Los datos se ven siempre como un flujo de fichas discretas a lo largo de arcos de un nodo a otro. Las fichas se consideran instancias de objetos de datos. Cada objeto se puede representar mediante una codificacion finita.

Cuando un nodo se etiqueta con una función escalar como + o *, la funcion se repite cada vez que llegan fichas a su entrada. Cada repeticion produce una ficha en su salida. La existencia de un arco de salida de un nodo implica la replicacion conceptual de las fichas que salen del nodo. Un nodo marcado con un valor constante regenera ese valor tan seguido como lo requieran los nodos a los que esta conectado. En la figura 4.18. se puede ver la grafica que define el calculo repetido sobre un flujo de valores de x en la funcion $x^2 - 2*x + 3$.

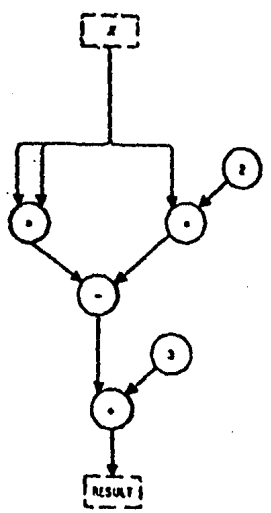


Figura 4.18. Grafica de flujo de datos para $x^2 - 2*x + 3$.[25]

En la figura 4.18. no hay arcos conectando a los dos operadores *. Esto implica que no hay dependencias de datos entre los dos nodos; esto es, la dos funciones se pueden ejecutar concurrentemente. Este tipo de concurrencia se conoce como

concurrrencia horizontal o espacial y está en contraste con la **concurrencia temporal o pipeline**, que existe entre cálculos correspondientes a varias generaciones de fichas de entrada.

En los programas de flujos de datos las construcciones condicionales logran la seleccion de distintos caminos entre nodos. Las decisiones son producidas por fichas de valores booleanos o de indices. En la figura 4.19. se muestra un nodo selector y uno ditribuidor. En el caso del selector, primero se absorve una ficha de la entrada horizontal. El valor de la ficha, verdadero o falso, determina por cual de las dos entradas la siguiente ficha será absorbida; la ficha en la otra entrada se queda ahí hasta que sea seleccionada. En el caso del distribuidor, la ficha de la entrada horizontal indica por cual de los dos caminos saldra la ficha de la entrada vertical.

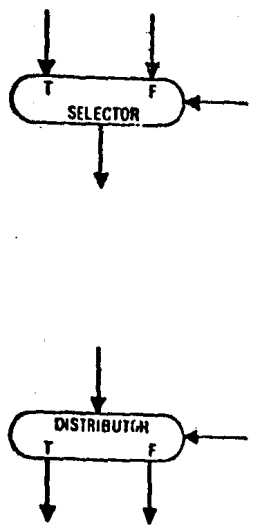


Figura 4.19. Funciones de seleccion y distribucion.[25]

Se puede lograr iteracion mediante graficas ciclicas. Un ejemplo de grafica iterativa se muestra en la figura 4.20. El programa implementa el metodo de Newton para encontrar las raices de una funcion. Los nodos f y f' podrian ser reemplazados por las graficas respectivas.

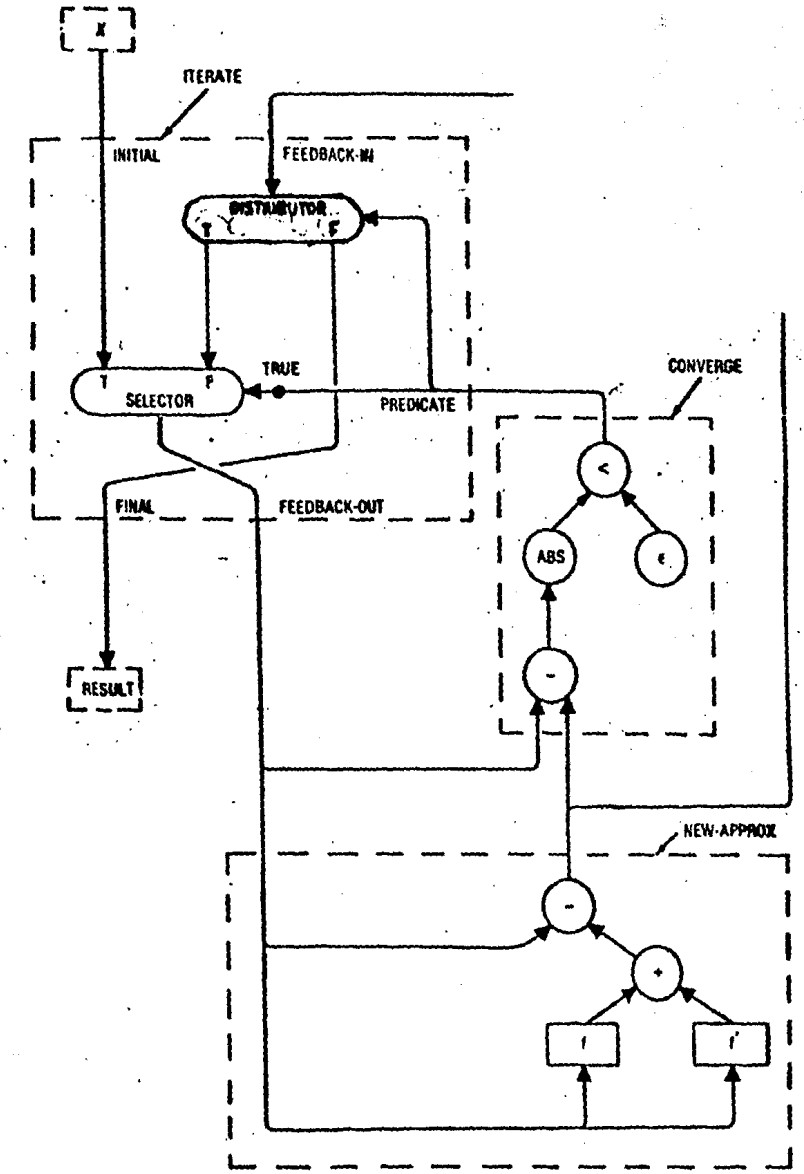


Figura 4.20. Una grafica para el metodo de Newton.[25]

4.6.2. Estructuración del programa.

Es difícil trabajar con programas gráficos que consisten de una sola grafica muy grande. De la misma forma que se utilizan subrutinas para estructurar programas convencionales, se pueden utilizar macrofunciones para estructurar programas gráficos. Se define una macrofunción especificando su nombre y asociandolo con una grafica llamada su **consecuente.** Un nodo en un programa llamado con ese nombre, es efectivamente reemplazado por su consecunete; esta sustitucion se llama **macroexpansión** y es valido pensar que se hace a la hora de ejecucion. Una funcion **atómica** es aquella que no es una macrofuncion. Un ejemplo de como se puede lograr recursion en programas graficos se puede ver en la implementacion de cada programa del mismo programa de la figura 4.21.

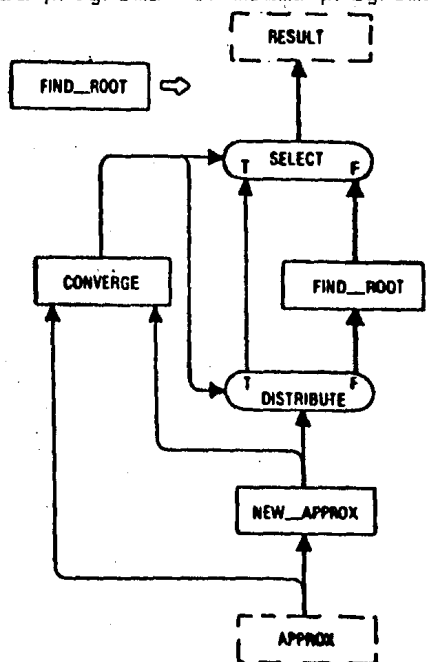


Figura 4.21. Gráfica recursiva para el método de Newton.[25].

4.6.3. Estructuración de los datos.

Hasta el momento solo hemos tratado con fichas de numeros y de valores booleanos. Pero la solucion de problemas mas complicados requiere frecuentemente de la construccion de fichas de objetos de datos mas complejos a partir de fichas de datos primitivos. Ademas, la estructuracion de datos permite una manera de explotar concurrencia; algunas operaciones que tratan con estructuras grandes, como la de suma de vectores pueden realizar subfunciones concurrentemente.

El **tuplo** es un ejemplo importante de una ficha compleja. El tuplo es simplemente la agrupacion de varios objetos en un solo objeto. Un tuplo puede fluir como una sola ficha y los objetos

originales se pueden extraer de el proporcionando el indice del objeto dentro del tuplo. Se pueden extender nuestras gtraficas de flujos de datos para que contengan operadores de tuplos (ver figura 4.22).

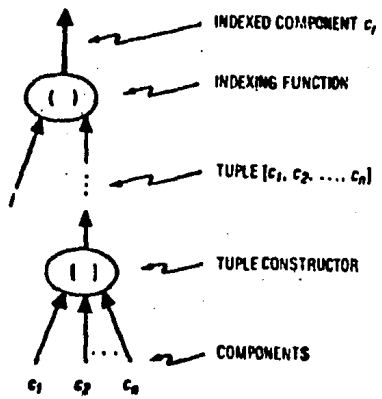


Figura 4.22. Formacion de tuplos y seleccion de componentes.[25]

Se pueden inclusive tener fichas que representen gráficas y un tipo de nodo que se llame "aplica". Cuando a un nodo aplica le llega una ficha funcion, el nodo es reemplazado por la gráfica que represente la ficha.

4.6.4. Representación y ejecución de programas gráficos.

Es posible combinar programas gráficos a lenguajes de maquina convencionales. Sin embargo, si la meta es ejecutar los programas, existen ventajas en codificar directamente la grafica como el lenguaje de maquina de un procesador especial. De esta forma se puede lograr una alta explotacion de concurrencia y ademas la coneccion entre el lenguaje grafico de alto nivel y su representación en lenguaje de maquina es muy clara.

Una discusion de las distintas (y numerosas) formas de codificar los problemas gráficos está más alla del alcance del presente trabajo. A continuación describimos solamente el modelo de fichas. La primera tarea es codificar la grafica que consiste de nodos titulados con nombres de funciones y arcos conectando los nodos. Se representa la gráfica como un conjunto de localidades de memoria contiguas, cada una correspondiendo a un nodo de la grafica. Debe haber un campo en cada localidad para el nombre del nodo ya que este nombre determina la funcion a ejecutar. Existe también una lista ordenada de arcos de entrada y una lista para los arcos de salida.

El modelo de ejecucion puede ser el de accion por datos o el

de accion por demanda. Para el de accion por datos tenemos las siguientes fases:

1. Un nodo recibe datos atraves de sus arcos de entrada.
2. Un nodo manda datos através de sus arcos de salida.

La alternativa de accion por demanda incluye las siguientes fases:

1. Un nodo recibe una peticion atraves de su arco de salida.
2. Un nodo solicita datos atraves de sus arcos de entrada.
3. A un nodo se le mandan datos por sus arcos de entrada si asi lo solicita.
4. Un nodo manda datos por sus arcos de salida.

4.6.5 Perspectivas futuras.

Los investigadores que tratan de convertir a las graficas de programas en lenguajes de programacion practicos se enfrentan con varios problemas. Algunos de estos son de ingenieria humana y otros son tecnologicos, como el desarrollo de terminales graficas de precio razonable.

Hasta la fecha solo dos lenguajes se han diseñado como lenguajes de alto nivel grafico: FGL y GPL. Estos premiten al programador dibujar el programa y ejecutarlo en forma grafica.

Los impedimentos para el uso directo de gráficas como lenguajes de flujo de datos incluyen: los elevados precios de terminales gráficas, el costo del software gráfico y el alto consumo de los recursos debido a la interavtividad y edicion graficas. Sin embargo, se cree que el avance de la tecnologia es tal que pronto permitira resolver estas dificultades.

Por otro lado está en contra la experiencia de 30 años de programadores acostumbrados a programar en lenguajes tipo Fortran, asi como la experiencia de desarrollo de hardware para esta filosofia. Es por esto que se deben hacer evidentes las ventajas de este tipo de programacion.

4.7 Ventajas y Desventajas de Flujo de Datos

En esta seccion exponemos algunas opiniones a favor [1] y alguna en contra [26] de las computadoras y lenguajes de flujos de datos. El desarrollo de los lenguajes y maquinas de flujo de datos esta todavia en una etapa primitiva. Actualmente las opiniones de la comunidad cientifica se encuentran divididas. Aun es muy temprano para obtener conclusiones definitivas ya que el exito del flujo de datos depende fuertemente de aspectos como tecnologia sofisticada y su adaptacion a aplicaciones. Sin embargo, en general se esta de acuerdo en que se debe continuar la investigacion y el desarrollo en esta área.

Los investigadores en esta área proclaman muchas ventajas que han sido solamente en parte apoyadas experimentalmente. Se carece de estadisticas operacioneles de máquinas de flujo de datos existentes. Por lo tanto algunas de las ventajas aclamadas necesitan de una mayor verificacion. Las computadoras de flujo de datos presentan varias ventajas sobre las computadoras convencionales de von Neumann. Algunos de estos aspectos se elaboran a continuacion.

Operaciones altamente concurrentes. El paralelismo puede ser fácilmente expuesto en una gráfica de programa de flujo de datos. El enfoque de flujo de datos ofrece una posible solucion al problema de explotar eficientemente concurrencia a gran escala. Beneficia no solamente en lo que respecta a paralelismo estructurado regularmente sino tambien el paralelismo arbitrario. El uso directo de valores permite la programación funcional sin efectos secundarios. La alta concurrencia obtenible es apoyada por la facil verificacion de programas, mejor modularidad y extensibilidad del hardware.

Acoplo con tecnología VLSI. La homogeneidad y modularidad de las estructuras en las arquitecturas de flujos de datos contibuye a que estas sean implementadas con tecnología VLSI apropiadamente. Las interconecciones entre chips se pueden construir con circuitos integrados de alta densidad.

Productividad de la programación. En un sistema multiprocesador o en un procesador de vectores, el porcentaje de codigo que puede ser vectorizado fluctua entre un 10 y un 90 % segun la aplicacion. El codigo no vectorizable tiende a ser un cuello de botella. La vectorización automatica requiere de análisis sofisticados de flujo de datos. Una computadora de flujo de datos bien diseñada debe ser capaz de resolver estas dificultades. Se ha insistido muchas veces en que los lenguajes funcionales van a aumentar la productividad del software, especialemte en aplicaciones que requieren de mucho procesamiento paralelo.

A continuaqcion presentamos un resumen de las criticas al flujo de datos expuestas en [26].

Desempeño bajo un grado reducido de paralelismo. Aparentemente el modelo de flujo de datos require mas instrucciones para resolver un problema que el de von Neumann y si el problema no es muy rico en paralelismo, la velocidad es mayor en el modelo de von Neumann. Si además el programa requiere de bastante manejo de arreglos, el desempeño de flujo de datos empeora más aún.

Almacenamiento de estructuras. Si se permite que las fichas contengan vectores, arreglos u otras estructuras, se provoca una grán sobrecarga en el sistema por el almacenamiento y transmision. Este es prácticamente el caso cuando de todo el arreglo solo se modifica un elemento. Las soluciones que se han propuesto (I-structures, por ejemplo) resuelven solamente en parte el problema y en algunos casos son un paso atras hacia los

modelos convencionales.

Lenguajes de flujos de datos. Los lenguajes de flujos de datos poseen algunas cualidades muy utiles como las de localidad de efecto y ausencia de efectos secundarios, sin embargo, algunos lenguajes imperativos también poseen estas cualidades. Por otro lado, esta el problema de que estos lenguajes no permiten el control explicito sobre el uso de memoria, provocando que el programador se encomiende totalmete a mecanismos como recoleccion de basura. Otra ventaja que proveen estos lenguajes es la regla de asignacion única, pero esto es fácil de implementar en cualquier otro lenguajes; de todas formas esto es facil de verificar por cualquier compilador.

Paralelismo implícito. El paralelismo implícito que muestran con claridad las graficas de programas de flujo de datos es positiva. El problema esta en que no proveen mecanismos de paralelismo explicito que en muchos casos es importante. Ademas, se conocen técnicas de compilacion que pueden extraer el paralelismo de lenguajes imperativos igual de bien.

PARTE III: Arquitecturas de Flujo de Datos y Reducción

4.8. Arquitecturas de Flujo de Datos y Reducción

Las ideas de esta sección están tomadas de [1], [23], y [27].

4.8.1. Computradoras de flujo de datos estáticas.

Jack Dennis y sus colegas de MIT desarrolllaron un modelo de computadora de flujo de datos estática. La estructura de esta maquina se muestra en la figura 4.23. Consiste de cinco secciones interconectadas por canales a lo largo de los cuales fluyen fichas discretas:

+ La seccion de memoria que consiste de celdas donde se almacenan las instrucciones y sus operandos.

+ La seccion de procesamiento que consiste de elementos procesadores que realizan las operaciones sobre las fichas.

+ La red de arbitraje que manda paquetes de la seccion de memoria a la de procesamiento.

+ La red de control que manda fichas de control de la seccion de

procesamiento a la de memoria.

+ La red de distribucion que manda fichas de datos de la seccion de procesamiento a la de memoria.

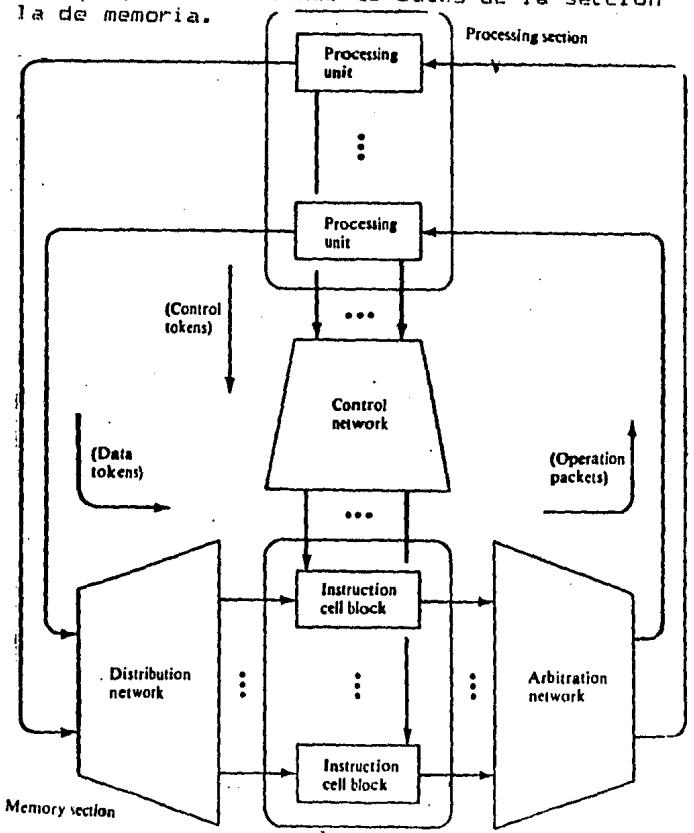


Figura 4.23. Estructura de la computadora de flujo de datos propuesta en MIT.[1]

La organizacion de esta computadora permite que una sola ficha este presente en cada arco. Esto causa que la regla de disparo sea que una instruccion es habilitada si todas sus entradas contienen fichas de datos y no hay ninguna ficha en su salida. Es por esto que se tienen tambien fichas de control además de fichas de datos.

Otra computadora basada en una organizacion similar a la computadora de MIT es la Distributed Data Processor (DDP) diseñada en Texas Instruments.

4.8.2. Computadoras de flujo de datos dinámicas.

Una computadora de flujo de datos dinámica es la Manchester Data Flow Computer descrita en [27]. El proyecto, que comenzó en

1975, incluyo el diseño del lenguaje de alto nivel de asignacion unica LAPSE.

La organizacion, que se muestra en la figura 4.24., es de flujo de datos puro, permitiendo la habilitacion de una instruccion cuando sus arcos de entrada contienen fichas, y viendo a un arco como una cola que permite el almacenamiento de fichas.

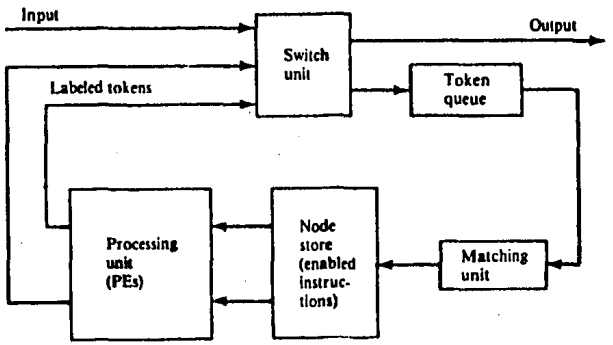


Figura 4.24. Arquitectura propuesta para la computadora Manchester.[1]

Las instrucciones contienen un código de operación, una direccion de instruccion de destino para una ficha de datos y ya sea una segunda direccion de destino o una literal. Cada instruccion consume una o dos fichas y emite una o dos fichas. Una ficha consiste de tres campos: el campo de valor que contiene al operando, un campo de direccion de instruccion que define la instruccion destino y finalmente un campo de etiqueta. Esta ultima se utiliza para agrupar fichas en conjuntos y provee tres tipos de información, identificando el proceso al cual la ficha pertenece, el arco por el que esta viajando y un numero de iteracion especificando de qué ficha en particular se trata. Este mecanismo de agrupacion y colas en arcos es lo que diferencia una computadora de flujo de datos estática de una dinámica.

Las cinco unidades mostradas en la figura son:

1. El switch que provee entrada/salida al sistema.
2. La cola de fichas que es un buffer que provee almacenamiento temporal para fichas.
3. El area de igualacion (matching store) que provee agrupacion de fichas.
4. El almacenamiento de instrucciones que contine los programas de flujo de datos, y,
5. La unidad de procesamiento que consiste de varios elementos procesadores identicos que ejecutan las instrucciones.

4.8.3. Computadora de reducción.

El proyecto de una maquina de reduccion de laboratorio GMD en Bonn, tiene como objetivo demostrar que las máquinas de reduccion son una alternativa practica a arquitecturas convencionales. La idea era construir una computadora que fuera fácil de programar directamente en un lenguaje basado en el cálculo lambda.

La organizacion de programa de la computadora GMD es de reduccion de cadenas. Los programas se representan en la maquinas como una expresion prefija. En la figura 4.25. se puede ver la organizacion de la máquina que se clasifica como centralizada, particularmente por la manera como almacena y ejecuta programas.

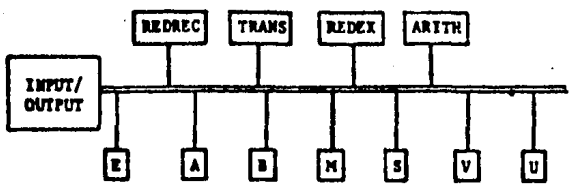


Figura 4.25. La maquina de reduccion GMD.[23]

Consiste de una unidad de reduccion (que comprende cuatro subunidades llamadas TRANS, REDREC, REDEX y ARITH), un conjunto de siete pilas de 4 Kbytes cada una (de los cuales E, A, B, U, V y M son usados para procesar expresiones y S es una pila para control del sistema) y un bus de un byte para comunicaciones. En la unidad de reduccion las cuatro subunidades realizan las siguientes tareas. TRANSport realiza todos los algoritmos de seguimiento (traversal); REDuction- RECongnition busca una expresion reducible durante el seguimiento y cuando la encuentra detiene a la unidad TRANS y pasa el control a la unidad REDEX. REDuction-EXecution esencialmente provee una memoria rapida de control que contiene todos los programas necesarios para realizar las reducciones. En esta tarea es asistida por la unidad ARITHmetic, que realiza todas las operaciones aritmeticas y lógicas.

Al ir recorriendo una expresion, la maquina utiliza principalmente tres pilas. Estas son E, M y A que se conocen como fuente, intermedio y destino (sink) respectivamente. La pila fuente contiene la expresión a reducir con la raiz en el tope. Al ir recorriendo la expresión, una sucesion de operaciones de POP mueve los simbolos de la pila fuente a la pila destino. La pila intermedia se utiliza para movimientos entre las otras dos pilas.

La maquina GMD ya ha sido construida y está conectada a una computadora que apoya con librerias y herramientas de programación. El sistema opera desde 1978.

C A P I T U L O V

Multiprocesamiento

Multiprocesamiento

En este capitulo se realiza una breve decripcion de los multiprocesadores. Primeramente se define el término multiprocesador, despues se hace una clasificacion de los multiprocesadores y se realiza una descripcion desde el punto de vista de hardware. Más adelante se da una lista de requerimientos que deben cumplir los procesadores utilizados en un multiprocesador. Finalmente se realiza una descripcion del software necesario en un multiprocesador incluyendo los sistemas operativos, sistemas de intercomunicacion, el problema de los abrazos mortales (dead lock), algoritmos paralelos para multiprocesadores, etc.

5.1 Definición de un Multiprocesador

Un multiprocesador es un sistema que satisface las siguientes cuatro caracteristicas:

1: Un multiprocesador contiene dos o mas procesadores con aproximadamente la misma capacidad.

2: Todos los procesadores comparten acceso a una memoria comun.

3: Todos los procesadores comparten acceso a canales de E/S unidades de control y otros dispositivos.

4: El sistema está controlado por un solo sistema operativo que permite la interaccion entre los procesadores y sus programas en los niveles de trabajo, tarea, paso y conjunto de datos.

Resulta claro que no todos los sistemas de multicomputadoras son multiprocesadores. Naturalmente, existen muchas similitudes entre multicomputadoras y los multiprocesadores debido a que han sido creados con el mismo objetivo - Realizar operaciones simultaneas en el sistema. Sin embargo, una diferencia importante se puede llegar a establecer si tomamos en cuenta el numero de elementos que se comparten en el sistema. Un sistema de multicomputadoras consiste de varias computadoras separadas (aunque exista comunicacion entre ellas), mientras que un sistema de multiprocesamiento es una sola computadora que contiene varios procesadores. Como ejemplo de un sistema de multicomputadoras tenemos de IBM el "Attached Support Processor System" y como ejemplo de un multiprocesador tenemos al "Denelcor s HEP System". Existen dos tipos de clasificaciones de los multiprocesadores, la primera hecha por Hwang y Briggs [1], clasifica en dos grandes grupos: multiprocesadores debilmente acoplados y

multiprocesadores fuertemente acoplados. Por otro lado Philip H. Enslow Jr. [12] clasifica en base a las diferentes organizaciones de multiprocesadores en tres grandes grupos: sistemas de tiempo compartido o de bus comun, matriz de switches y memorias multipuerto.

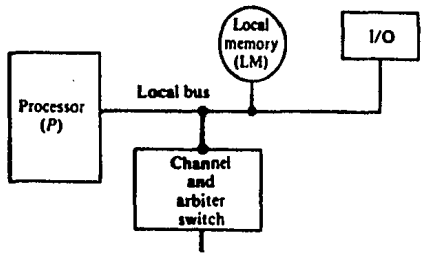
En esta tesis describiremos la clasificacion propuesta por Hwang y Briggs.

5.2 Multiprocesadores Débilmente Acoplados

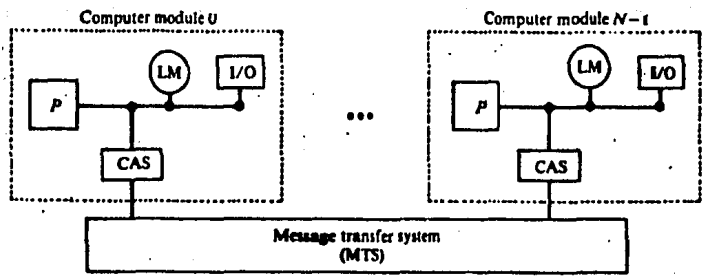
Los sistemas debilcmte acoplados generalmente no presentan tantos problemas en compartir la memoria como sucede en los multiprocesadore fuertemente acoplados. En estos sistemas cada procesador tiene un conjunto de dispositivos para E/S y una memoria local grande de donde accesa la mayoria de sus instrucciones y datos. Todos los procesos se pueden comunicar intercambiando mensajes mediante el sistema de transferancia de mensajes (MTS). El grado de acoplamiento en este tipo de sistemas resulta ser muy debil, es por esto que tambien se les conoce como sistemas distribuidos. El factor determinante para incrementar el grado de acoplamiento es la topologia del MTS. Los sistemas debilmente acoplados (LCS) resultan ser muy eficientes cuando las interacciones entre las tareas son minimas. La arquitectura que presentamos en el siguiente capitulo es debilmente acoplada, por lo que es recomendable en aplicaciones donde cada proceso tiene que realizar una gran cantidad de operaciones para que al terminar transmita un resultado pequeño. Los sistemas fuertemente acoplados (TCS) toleran grados mas altos de interacion entre tareas sin deteriorar el desempeño.

En la figura 5.1.a. se presenta un ejemplo de una computadora no jerarquica debilmente acoplada. Consiste de un procesador, memoria local, dipositivos de E/S e interfases con otros modulos del sistema. Esta interfase puede contener un canal y un switch como arbitro (CAS).

En la figura 5.1.b. se ilustra la conexion entre los modulos de la computadora y el sistema de transferencia de mensajes. Si una peticion de dos o mas modulos consideran al accesar un segmento fisico de la MTS, el arbitro es responsable de escoger una de todas la peticiones simultaneas de acuerdo con una jerarquizacion especifica. Tambien es responsable de retardar a la peticiones que se hicieran posteriormente hasta que se complete el servicio. El canal asociado al CAS debera tener una memoria de comunicaciones de alta velocidad para permitir la transferencia de bloque de mensajes. Esta memoria de comunicaciones puede ser accesada por todos los procesadores. Se espera que con la llegada de la tecnologia VLSI, los modulos puedan ser integrados en un solo circuito y utilizarse como el bloque basico de un sistema de multiprocesamiento.



(a) modulo de una computadora.



(b) union debil entre modulos.

Figura 5.1. Sistema de multiprocesamiento no jerarquico debilmete acoplado.[1]

El sistema de transferencia de mensajes de una LCS no jerarquica puede ser tan simple como un bus compartido, como en la PDP-11, o hasta un sistema de memoria compartida. En el ultimo caso se puede implementar con un conjunto de modulos de memoria y una red de interconexion entre procesadores y memorias. Como se dijo, uno de los factores mas importantes que determinan el desempeño de un sistema multiprocesador es la MTS. En las configuraciones LCS que utilizan un bus compartido, el desempeño esta determinado por la tasa de llegada de mensajes, la longitud de los mensajes y la capacidad del bus (en bits por segundo). En los sistemas LCS con memoria compartida, el factor limitante es el conflicto creado por la red de interconexion entre el procesador y la memoria.

La memoria de comunicaciones tambien puede ser centralizada y estar conectada a un bus de tiempo compartido, o formar parte de un sistema de memoria compartida. Los procesos (tareas) se pueden comunicar con otros procesos dentro del mismo procesador o alojados en otros procesadores. Las comunicaciones entre procesos dentro del mismo procesador se llevan a cabo dentro de la memoria local. Las comunicaciones de tareas alojadas en distintos procesadores se llevan a cabo mediante un puerto de comunicaciones que se encuentra dentro de la memoria de comunicacion. Se asocia un puerto de comunicacion con cada procesador.

En la figura 5.2 se describe la estructura de comunicacion entre procesos. Un proceso ubicado en el procesador i pone un

mensaje en el puerto de entrada dentro de otro proceso tambien dentro de P_1 tal como se muestra con la flecha a. La flecha b muestra una accion de dos pasos para la transferencia de mensajes entre procesadores. La flecha b_1 manda el mensaje al puerto de entrada del procesador 2, la flecha b_2 muestra el translado del mensaje al puerto de entrada del proceso de destino. Como ejemplo de una computadora LCS tenemos a la Cm* que es un proyecto realizado en la Universidad de Carnegie Mellon.

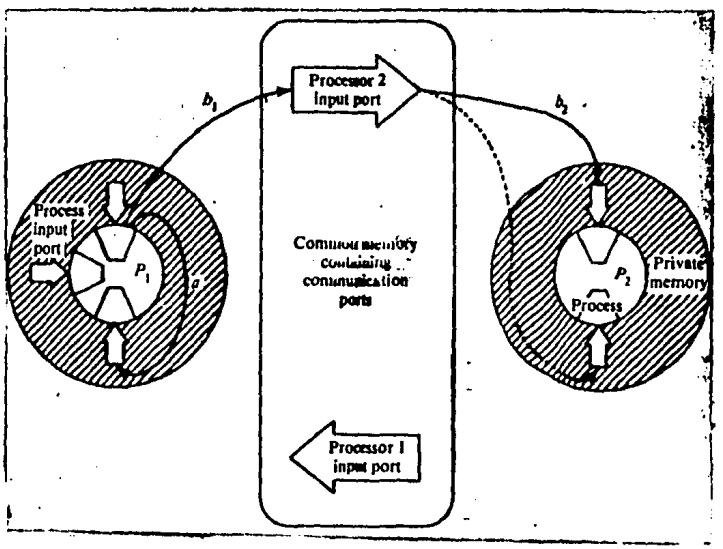


Figura 5.2. Comunicación entre procesos dentro de un sistema de multiprocesamiento.[1]

5.3. Multiprocesaodres Fuertemente Acoplados

Debido a la gran variacion que existe entre los tiempos de interferencia, el desempeño de los sistemas jerarquicos debilmente acoplados puede llegar a ser demasiado bajo para algunas aplicaciones que necesitan tiempos de respuesta rapidos. Si se necesita procesamiento en tiempo real se deben utilizar los sistemas fuertemente acoplados (FCS). Existen dos modelos

tipicos de estos sistemas y se describen a continuacion. El primer modelo se presenta en la figura 5.3. Consiste de P procesadores L modulos de memoria y D canales de E/S. estas unidades estan interconectadas mediante tres redes llamadas, red de interconexion procesador-memoria (PMIN), red de interconexión E/S-procesador (IOPIN) y la red de interconexion interrupcion-señal (ISIN). La PMIN es un switchs que puede conectar a cada procesador con cada modulo de memoria. Para reducir el numero de conflictos se asocia una área reservada de almacenamiento a cada procesador. Esto se presenta como la memoria sin mapeo local (ULM) de la figura 5.3. Un ejemplo de un multiprocesador con estas caracteristicas es la conocida computadora C.mmp diseñada en la Universidad de Carnegie Mellon y que consta de 16 procesadores. En esta organizacion de procesadores, cada procesador puede realizar referencias a memoria que sean accesadas de memoria principal. Estas referencia a memoria contribuiran a los conflictos en los modulos de memoria. Debido a que cada referencia a memoria debe de pasar por la PMIN, entonces habra un retardo en el switch de procesador-memoria y por lo tanto el ciclo de instruccion se ve incrementado. Este incremento de ciclo de instruccion determinara el desempeño del sistema. Para reducir este retardo podemos asociar a cada procesador una memoria cache que capture la mayoria de las referencias a cada procesador. Otra ventaja al utilizar una memoria rápida es que el tráfico en el switch de cruce (Crossbar Switch) que reduce el tiempo de retencion en los puntos de cruce. En la figura 5.4. se presenta una organizacion que utiliza memorias cache privadas. En esta organizacion se presenta el problema de la coherencia de caches. Esto se refiere a que más de una copia inconsistente de informacion puede existir en el sistema y debido a que este tipo de memorias son muy caras, no es conveniente duplicar informacion. Se han propuesto diferentes soluciones para el problema de coherencia y pueden ser estudiados en [1] capitulo 7. Como ejemplo de una computadora con memoria cache privada tenemos a la IBM 3084 y a la S-1.

La ISIN permite a cada procesador dirigir e interrumpir a cualquier otro procesador. Mediante esta red de interconexion se facilita la interconexion entre los procesadores. La ISIN permite que algun procesador descompuesto accione una alarma por medio de un procesador en buen estado. La complejidad de la ISIN puede variar desde un bus de tiempo compartido hasta una compleja matriz de switches de cruce. Por ejemplo en la UNIVAC 1100/80 y en la Honeywell 60/66 se establece una conexión entre cada par de procesadores. En la C.mmp se utiliza un bus de tiempo compartido para la conexion entre procesadores. Resulta evidente que un bus de tiempo compartido es más barato que una matriz de switches de cruce. Sin embargo, en la primera se tienen mayores retardos debidos a la logica de los arbitros. Por otra parte, cuando la demanda del bus es suficientemente baja, esta técnica resulta ser muy atractiva para la comunicacion entre procesadores.

La IOPIN permite a los procesadores comunicarse con un canal de E/S que esta conectado a dispositivos periféricos.

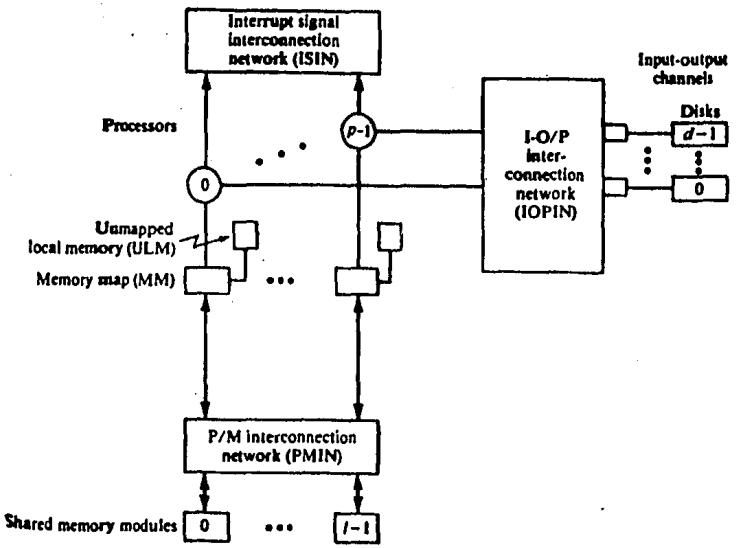


Figura 5.3. Configuración de un multiprocesador fuertemente acoplado.

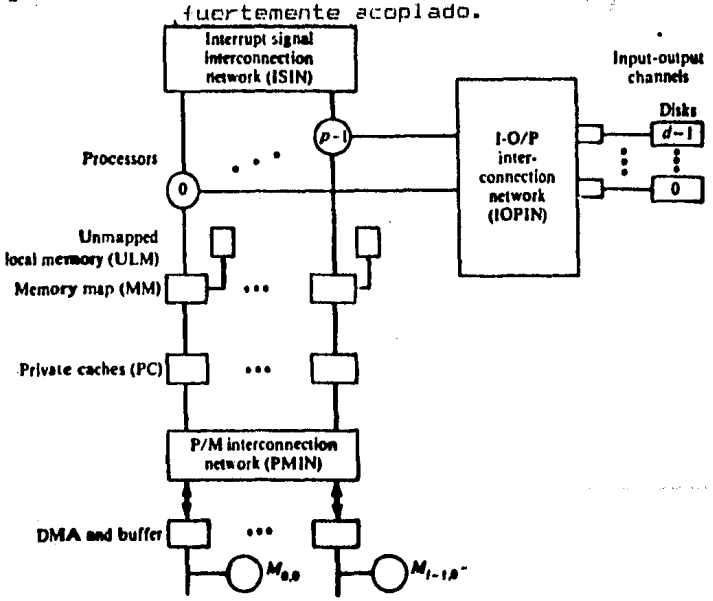


Figura 5.4. Multiprocesador fuertemente acoplado con memoria cache

El conjunto de procesadores utilizados en un sistema de multiprocesamiento puede ser homogeneo o heterogeneo. Se dice que es homogeneo si los procesadores son funcionalmente identicos. Aun cuando los procesadores sean homogeneos pueden ser asimetricos. Esto es dos componentes funcionalmente identicos pueden llegar a diferir en otras dimensiones tales como la accesibilidad de E/S, desempeño, etc. Como ejemplo de las configuraciones simétricas tenemos a la Honeywell 60/66 y la UNIVAC 1100/80. Ejemplos de multiprocesadores asimitricos son la IBM 3084 AP y la C.mmp. En la arquitectura que se presenta en el siguiente capitulo se trata de un multiprocesadort homogeneo asimetrico.

En la mayoria de los casos la simetria o asimetria de los multiprocesadores es transparente al usuario del sistema. Solamente resulta ser interesante para el sistema operativo esencialmente cuando se deben balancear las cargas de trabajo. En general un sistema homogeneo es mas facil de programar y elimina el problema de conector, que se trata de conectar eficientemente dos procesadores diferentes. Los sistemas simetricos ayudan a encontrar fácilmente los errores en caso de que se presente alguna falla.

5.4. Caracteristicas de los Procesadores para el Multiprocesamiento

La mayoria de los multiprocesadores se han construido utilizando procesadores que originalmente no fueran diseñados para el multiprocesamiento como ejemplo tenemos a la C.mmp que utiliza procesadores DEC PDP11, y la Cm* que utiliza microprocesadores LSI-11. Una razon para utilizar componentes ya construidos es para reducir el tiempo de desarrollo. Sin embargo, este tipo de componentes pueden llegar a crear contratiempos no deseados. A continuación se presentan algunas caracteristicas de los procesadores que se deberán satisfacer para que resulten adecuados en un sistema de multiprocesamiento.

Recuperación del proceso. Los procesadores deben poder distinguir que el procesador y el proceso son dos cosas distintas. Si el procesador tiene alguna falla debera ser posible para algun otro procesador recuperar la información del proceso interrumpido de tal forma que su ejecucion puede continuar. En general la mayoria de los procesadores guardan el estado del proceso en registros internos que no se pueden accesar desde afuera y que no son guardados en memoria cuando se presenta

alguna falla.

Cambio eficiente de contexto. Cuando se trabaja con registros que pueden ser accesados por todos los procesadores se puede llegar a tener procesadores multiprogramados. Sin embargo, para tener una utilizacion eficiente es necesario tener dominio de direcciones y por lo tanto poder realizar la operacion de cambio de contexto (context switching) en este dominio. Este tipo de operaciones requieren una gran cantidad de colas y operaciones con pilas. En la operación de cambio de contexto se guarda el estado del proceso actual y se cambia a un proceso seleccionado que este listo para ejecutarse. La computadora IBM 370/168 es un ejemplo de procesadores con varios dominios de procesadores.

Espacios de direcciones grandes, virtuales y fisicas. Un procesador que se pretende utilizar en la construcción de multiprocesadores debera ser capaz de soportar un gran espacio fisico de direccionamiento. Aunque los algoritmos sean descompuestos de tal forma que puedan ser implementados con un minimo de instrucciones los procesos requieren accesar grandes cantidades de datos.

También es deseable que se tenga espacio virtual y de ser posible este espacio debera segmentarse para permitir compartir modularmente y checar las direcciones para protección de la memoria.

Mecanismos para comunicación entre procesadores. El conjunto de procesadores utilizados en un multiprocesador debera tener medios eficientes para su intercomunicacion. Estos medios deberán ser implementados desde hardware, para que se facilite la sincronizacion entre los procesadores. Este mecanismo puede ser utilizado cuando se presente alguna falla en un procesador para mandar una señal (por hardware) de falla a los demas procesadores que al percatarse del error, podrán empezar una rutina de información o un procedimiento de diagnóstico.

Conjunto de instrucciones. El conjunto de instrucciones del procesador debera tener las facilidades para implementar lenguajes de alto nivel que permitan concurrencia en el nivel de procedimiento y que permitan manejar eficientemente estructuras de datos. Las instrucciones deberán ser capaces de poder implementar el ligado de procedimientos, manipulacion de parámetros, computos multidimensionales y chequeo de rangos de direcciones. Mas importante aun, el conjunto de instrucciones debera incluir instrucciones para poder crear y terminar procesos paralelos. En otras palabras, se desea tener un conjunto de instrucciones completo.

5.5 Programas Concurrentes

Esta seccion esta basada en [1],[10],[28] y [29]. Es una introduccion a los procesos y procesos que interactuan; es decir, a los programas concurrentes.

5.5.1 Procesos.

Un **programa secuencial** especifica la ejecucion secuencial de una lista de instrucciones; su ejecucion se llama **proceso.** Un **programa concurrente** especifica dos o mas programas secuenciales que pueden ser ejecutados concurrentemente como procesos paralelos.

Un programa concurrente puede ser ejecutado ya sea permitiendo que procesos compartan uno o mas procesadores, o corriendo cada proceso en un procesador. El primer enfoque se conoce como **multiprogramación.** El segundo enfoque se conoce como **multiprocesamiento** si los procesadores comparten memoria, o como **procesamiento distribuido** si los procesadores estan conectados mediante una red de comunicaciones.

Debido a que nos gustaria poder comprender un programa concurrente en terminos de sus procesos secuenciales y sus interacciones, sin importar como se ejecuten (multiprogramacion, multiprocesamiento o procesamiento distribuido), no hacemos ninguna otra suposicion acerca de las velocidades de ejecucion de los procesos concurrentes que la de que las velocidades son todas positivas. A esta suposicion se le llama **suposición de progreso finito.**

5.5.2 Interacción entre Procesos.

Los procesos concurrentes necesitan comunicarse y sincronizarse para cooperar. La comunicacion permite que la ejecucion de un proceso influencie a la ejecucion de otro proceso. La comunicacion entre procesos se basa en la utilizacion de variables compartidas o en el envio de mensajes.

Cuando los procesos se comunican, en general es necesario que haya sincronizacion. Los procesos se ejecutan con velocidades impredecibles. Se puede ver a la sincronizacion como un conjunto de restricciones en el orden de los eventos. El programador emplea algun **mecanismo de sincronización** para retardar la ejecucion de un proceso de tal forma que se satisfagan estas restricciones.

Existen dos enfoques para analizar la semantica de un programa: el enfoque operacional y el enfoque axiomatico. En el **enfoque operacional**, la ejecucion de un programa concurrente se puede ver como una secuencia de acciones atomicas cada una resultado de una operacion indivisible. Esta secuencia consiste de un entrelazado de acciones atomicas de distintos procesos. Raramente todas las distintas combinaciones resultan en un comportamiento aceptable del programa. Los procesos se deben sincronizar para evitar combinaciones no deseadas.

El **enfoque axiomático** provee un segundo marco en el cual ver el papel de la sincronizacion. En este enfoque, la semantica de los enunciados se define mediante axiomas y reglas de inferencia. Esto resulta en un sistema formal de logica llamado "programacion logica".

Los mecanismos de sincronizacion controlan interferencia de dos maneras. Primero, pueden detener la ejecucion de un proceso hasta que una condicion se cumpla. Segundo, un mecanismo de sincronizacion puede ser usado para asegurar que un bloque de instrucciones sea una operacion indivisible.

Resumiendo, existen tres problemas principales en el diseño y la especificacion de la programacion concurrente:

(i) Como indicar la ejecucion concurrente.
(ii) Que forma de comunicacion entre procesos utilizar.
(iii) Que mecanismo de sincronizacion utilizar.

Ademas, los mecanismos de sincronizacion se pueden ver como una restriccion en el orden de los eventos o como un control de interferencia. Consideraremos todos estos problemas en el resto del capitulo.

5.6. Especificación de Concurrencia

Dos procesos se denominan concurrentes si la primera operacion de un proceso comienza antes que la ultima operacion del otro termine. Se han propuesto diversas notaciones para espacificar ejecucion concurrente. Algunas de las primeras propuestas, como el enunciado FORK, tienen la deficiencia de que no separan la definicion del proceso de la sincronizacion. Posteriormente se separaron estos conceptos y se inventaron enunciados estructurados para la especificacion de concurrencia.

Una de las primeras ideas de la programacion concurrente fue la de especificar el comienzo y la terminacion de un proceso mediante dos operaciones: FORK y JOIN. El significado de estas dos operaciones se puede explicar mejor mediante un ejemplo (ver figura 5.5). En esta figura la instruccion FORK A,J,N indica que el proceso A debe comenzar a ejecutarse en paralelo, y pone el contador J en el valor N. La instruccion JOIN J decrementa en uno la variable J; si el resultado es 0, el proceso en la direccion J+1 se inicia, de otra manera el procesador que esta ejecutando la instruccion JOIN se libera. En el ejemplo, todos los procesos quedan detenidos al ejecutar la instruccion JOIN excepto el ultimo que lo haga.

La instruccion FORK es a la programacion concurrente lo que el GOTO a la programacion secuencial. Una representacion bien estructurada de la concurrencia de procesos fue sugerida por Dijkstra en 1965. La orden

cobegin S1; S2;..., Sn coend

indica que los enunciados S1, S2,..., Sn se pueden ejecutar

concurrentemente; cuando todos han terminado, la instruccion en seguida del enunciado se ejecuta.

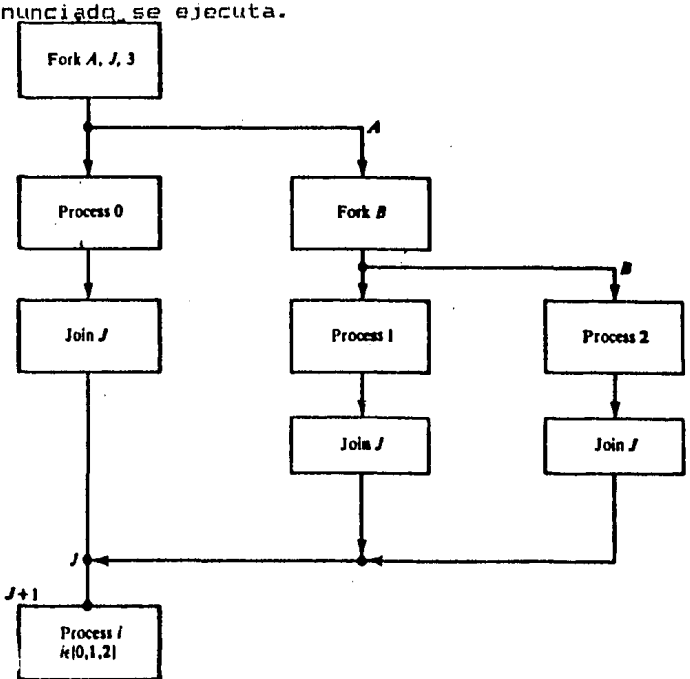


Figura 5.5 Concepto de Conway del enunciado FORK-JOIN [1].

Ya que la instruccion cobegin tiene una entrada y una salida, es adecuada para la programacion estructurada. Los enunciados concurrentes pueden estar anidados arbitrariamente como en el siguiente ejemplo que se ilustra en la figura 5.6.

```
begin
     S0;
   cobegin
     S1
     begin S2; cobegin S3; S4; S5; coend S6; end
     S7;
   coend
   S8;
end
```

Frecuentemente se tienen casos de paralelismo en ciclos (loops). La instruccion **parfor** (parallel for) indica que una serie de procesos se puede ejecutar concurrentemente.

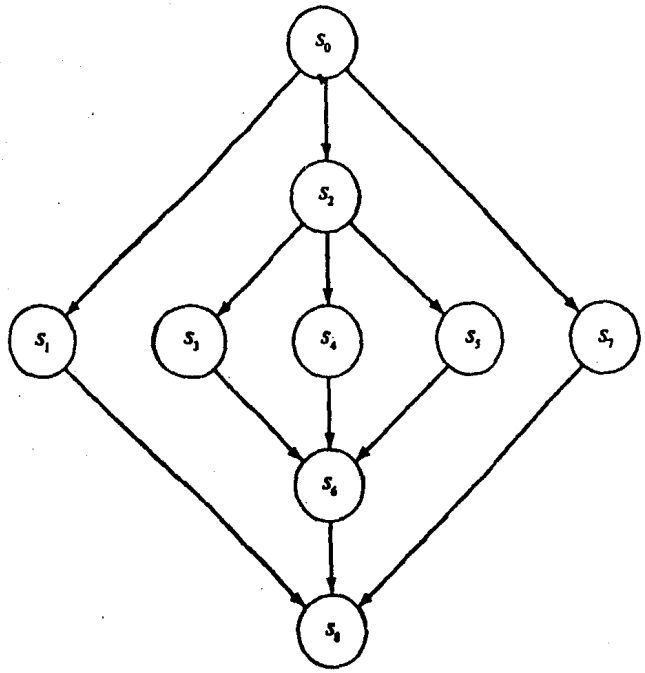


Figura 5.6 Grafica de precedencia de procesos concurrentes anidados [1].

La secuencia

```
parfor i=1 until n do
    S(i);
end
```

indica que n enunciados S(i) en los que se sustituyen las ocurrencias de i por su valor correspondiente se pueden ejecutar en paralelo.

5.7. Detección de Paralelismo en Programas

En la seccion precedente vimos como el programador podia indicar explicitamente el paralelismo. En esta seccion nos dedicaremos al problema de detectar paralelismo en forma automatica a nivel de compilador. Los estudios en esta área caen

en dos grupos: deteccion de paralelismo dentro de una expresion y deteccion de paralelismo entre enunciados consecutivos. El primero tiene mejor aplicacion en la generacion de codigo para multiprocesadores no homogeneos mientras que el segundo esta efocado a multiprocesadores homogeneos.

El problema de la deteccion de paralelismo dentro de una expresion aritmetica se resuelve transformando la expresion en un arbol binario mediante algoritmos que ademas despliegan las operaciones que pueden ser ejecutadas cocurrentemente. El lector debe referirse a [9] o [29] para una discusion de este tema.

La dependencia de los datos es el factor principal para la deteccion de paralelismo entre procesos. El primero en atacar el problema fue Bernstein; sus resultados se resumen en seguida.

Las condiciones de Berstein deben ser satisfechas antes de que procesos secuenciales puedan ser ejecutados en paralelo. Estas estan basadas en dos conjuntos separados de variables para cada proceso T_i.

1. El conjunto de lectura I_i representa todas las localidades de memoria para las cuales la primera operacion que las utiliza en T_i es de lectura.
2. El conjunto de escritura O_i representa al conjunto de todas las localidades en las que T_i manda escribir.

Las condiciones bajo las cuales T_1 y T_2 pueden ser ejecutados como dos procesos concurrentes independientes son:

1. $I_1 \cap O_2 = \emptyset$
2. $I_2 \cap O_1 = \emptyset$
3. $(O_1 \cap O_2) \cap I_3 = \emptyset$

Los sistemas que detectan paralelismo en forma automatica están basados en las condiciones anteriores; refierase a [9], [29] y [32].

5.8 Mecanismos de Comunicación entre Procesos

Se han propuesto varios esquemas de comunicacion entre procesos. Esta seccion enumera algunos primitivos de sincronizacion dividiendolos en los que estan basados en variables compartidas y los que estan basados en envio de mensajes.

5.8.1. Primitivos de sincronización basados en variables compartidas.

Cuando se utilizan variables compartidas para la comunicacion entre procesos, son utiles dos clases de sincronizacion: exclusion mutua y sincronizacion condicional. La primera asegura que una secuencia de expresiones es tratada como

una operacion indivisible; a esta secuencia se le llama **sección critica.**

Otra situacion en la que es necesario coordinar la ejecucion de procesos concurrentes es cuando una variable esta en un estado inapropiado para ejecutar una operacion en particular. A este tipo de sincronizacion la llamamos **sincronización condicional.**

En seguida mencionamos varios mecanismos para implementar estos dos tipos de sincronizacion. Para una explicación mas detallada refierase a [1],[10],[28] o [33].

Espera activa. Una manera de implementar sincronizacion es teniendo varios procesos leyendo y escribiendo variables compartidas. Este enfoque funciona razonablemente bien para implementar sincronizacion condicional, pero no para exclusion mutua.

Semáforos. Dijkstra invento las dos operaciones P y V, que pueden ser compartidas por muchos procesos y que implementan la exclusion mutua eficientemente.

Un semaforo es una variable entera no negativa sobre la cual se definen las operaciones P y V. Dado un semaforo s, P(s) se espera hasta que s > 0 y despues se ejecuta s=s-1; la prueba y el decremento se asumen como operaciones indivisibles. V(s) ejecuta s=s+1 como una operacion indivisible. Si el semaforo solo puede tomar los valores 0 y 1 se le llama **semáforo binario.**

Secciones criticas condicionales. Las secciones criticas condicionales (SCC) fueron propuestas por Hoare y Hansen (1972) para resolver la mayoria de los problemas de las operaciones P y V. Son una manera estructurada y altamente orientada a usuario de especificar comunicacion entre procesos concurrentes. Su uso permite expresar directamente el hecho de que un proceso debe esperar hasta que se cumpla una condicion arbitraria en las variables compartidas. La comunicacion entre procesos se realiza mediante la variable compartida v que esta compuesta de las variables v1, v2,...,vn. La variable v denota un recurso dado. Las variables de v pueden ser accesadas solamente dentro de expresiones de una SCC que nombre a v. Una SCC es de la forma

cset v **do await** c:s

donde c es una expresion booleana que prueba la condicion necesaria para entrar a la lista de enunciados s.

Monitores. Las SCC s son costosas de implementar en sistemas con un solo procesador. Ademas, los enunciados que realizan operaciones sobre las variables compartidas estan dispersos por todo el programa. Esto quiere decir que uno debe estudiar todo el programa para conocer todas las formas en que un recurso es utilizado. Los monitores alivian estas dos dificultades; un monitor se forma al encapsular la definicion del recurso, las operaciones que lo manipulan y las inicializaciones necesarias. Dos lenguajes que utilizan monitores son: PASCAL Concurrente y Modula.

Expresiones de caminos. Las operaciones definidas por un monitor son ejecutadas con exclusion mutua. El resto de la sincronizacion se realiza mediante llamadas explicitas a operaciones del tipo **wait** y **signal**. Consecuentemente el codigo de sincronizacion se encuentra desperdigado por todo el monitor.

La idea de las expresiones de camino (path expressions) es agrupar todas las restricciones de la ejecucion de operaciones en un solo lugar. La sintaxis de las expresiones de caminos definidas por primera vez por Campbell y Habermann (1974) es:

path path-list **end**

Un pathlist contiene nombres de operaciones y operadores de caminos que son: "," para concurrencia, ";" para secuenciacion, "n:(path-list)" para n activaciones concurrentes de path-list, y "[path-list]" para especificar un numero no limitado de activaciones concurrentes de path-list.

5.8.2. Primitivos de sincronización basados en envio de mensajes.

A diferencia de los mecanismos reseñados arriba, que son una extension de los semaforos, A continuacion expondremos un enfoque basado en el envio de mensajes para lograr comunicacion y sincronizacion entre procesos [10].

Cuando se utiliza envio de mensajes, se logra comunicacion ya que cuando un proceso recibe un mensaje, obtiene valores del otro proceso que envio el mensaje. Se logra sincronizacion ya que un mensaje puede ser recibido solamente despues de que fue enviado.

Se envia un mensaje mediante la ejecucion de

send expression-list **to** destination-designator

El mensaje se recibe al ejecutar

receive varible-list **from** source-designator

Un lenguaje basado en esta nocion es CSP [34]. Debido a que una interaccion en la que se invoca un send seguido de un receive es muy comun, se han definido primitivos que realizan estas dos funciones y se denominan en general **llamadas remotas a procedimientos.** Se utiliza la instruccion

call service(value-args;result-args)

Por otro lado, la rutina de servicio puede ser

```
remote procedure service
        (in value-param;
         out result-param)
      body
end
```

Un lenguaje basado en este concepto es Ada.

5.8.3. Modelos de lenguajes de programación concurrente.

Los lenguajes de programacion concurrente utilizan una variedad muy grande de mecanismos para la interaccion entre procesos; sin embargo, cada uno se puede ver como un elemento de una de tres clases: orientados a procedimientos, orientados a mensajes y orientados a operaciones.

En los lenguajes **orientados a procedimientos**, la interaccion entre procesos esta basada en variables compartidas. Estos lenguajes contienen tanto objetos activos (procesos) como objetos compartidos, pasivos (modulos, monitores, etc.). Ejemplos de estos lenguajes son: PASCAL Concurrente, Modula, Mesa y Edison.

Los lenguajes orientados a mensajes y a operaciones estan ambos basados en el envio de mensajes pero reflejan diferentes visiones de la interaccion entre procesos. Los lenguajes **orientados a mansajes** proveen como principal medio de interaccion a las instrucciones send y receive. En este caso no existen objetos pasivos y compartidos, sino que cada objeto es administrado por un solo proceso que realiza todas las operaciones sobre el. CSP, Gypsy y PLITS son ejemplos de lenguajes orientados a mensajes.

En los lenguajes **orientados a operaciones** el medio primario para interaccion entre procesos son las llamadas remotas a procedimietos. Estos lenguajes combinan aspectos de las otras dos clases. De la misma forma que en los lenguajes orientados a mensajes, cada objeto tiene un administrado asociado a el; de igual manera que en los orientados a procedimientos, se realizan operaciones sobre los objetos mediante llamadas a procedimientos. La diferencia es que cuando un procedimineto llama a un administrador, estos se sincronizan durante la ejecucion del servicio, despues los dos prosiguen asincronamente. Algunos ejemplos de lenguajes que pertenecen a esta categoria son StartMod, Ada y SR.

Los lenguajes de cada una de estas clases son equivalentes en poder expresivo. Todos pueden ser usados para escribir programas concurrentes para uniprocesadores, multiprocesadores y sistemas distribuidos. Sin embargo, no todos se adaptan igual de bien a estos tres tipos de sistemas. En la figura 5.7 se muestra la evolucion y las relaciones entre los conceptos revisados en esta seccion.

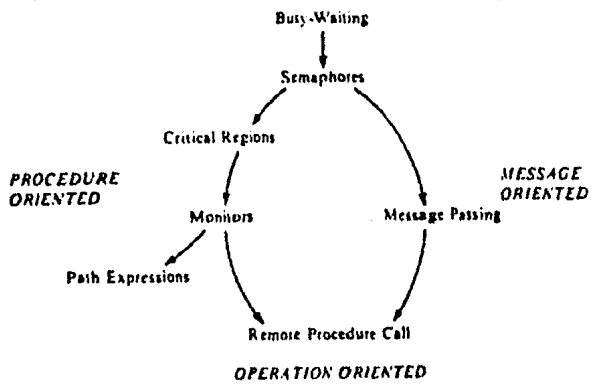


Figura 5.7. Programacion concurrente.[10]

5.9. Referencias Adicionales

En esta seccion se citan referencias de algunos otros temas importantes de software para multiprocesadores.

Abrazo mortal. Un problema interesante en programacion concurrente es el problema del abrazo mortal. Es un nuevo fenomeno ausente al nivel de la programacion secuencial. Su ocurrencia en los sistemas paralelos puede tener consecuencias muy graves, especialmente cuando el programa esta controlando los sistemas en tiempo real. Se dice que uno o mas procesos estan en abrazo mortal cuando esperan a que se cumpla alguna condicion o evento que no habra de cumplirse. Las condiciones necesarias para que ocurra un abrazo mortal y las politicas para resolver el problema se describen en cualquier libro de sistemas operativos, en la referencia [33] y en la [1].

Protección. La proteccion es un mecanismo para verificar si los procesos concurrentes no estan tratando, erronea o maliciosamente, de exceder sus derechos de accesar alguno de los recursos del sistema. La proteccion debe distinguirse de la **seguridad**, que es una politica utilizada para denotar mecanismos y tecnicas que controlan quien puede usar y modificar el sistema de computo. Los mecanismos de proteccion protegen un proceso de otro. En [1] y en [35] se describe el problema de la proteccion. El articulo [35] se desarrolla en tres partes que cubren el tema desde lo mas basico hasta un nivel de mayor profundidad.

Asignación de procesadores. La asignacion de procesadores (scheduling) implica la asignacion de tareas a procesadores para ser ejecutadas en un momento dado. La introduccion de multiples procesadores complica el problema de la asignacion de tareas. Se han desarrollado tanto modelos deterministicos como probabilisticos para evaluar los esquemas de asignacion de procesadores. Ambos enfoques se describen en [1]; en [36] se considera unicamente el problema en forma deterministica. En [7] se presenta un algoritmo paralelo de asignacion de procesadores.

Sistemas operativos. Existe poca diferancia conceptual entre los requerimientos de un sistema operativo para un multiprocesador y los de un sistema de computo grande con multiprogramacion. Sin embargo, hay una complejidad adicional cuando el sistema operativo esta diseñado para un multiprocesador. Existen tres organizaciones basicas que han sido utilizadas en el diseño de sistemas operativos para multiprocesadores: maestro esclavo, un sistema separado para cada procesador , y un tratamiento simétrico o anónimo de todos los procesadores. Consultar [1] y [12].

C A P I T U L O VI

Diseño de una Arquitectura

Diseño de una Arquitectura

En este capitulo se propone una arquitectura en paralelo basada en el microprocesador TMS32010. Hemos seleccionado este microprocesador debido a que se quiere aplicar la arquitectura en el analisis de señales de voz. En la primera parte de este capitulo se presenta un estudio de las redes Petri porque la arquitectura ha sido modelada mediante esta herramienta. Despues se presenta el modelo de la arquitectura y se detalla en su funcionamiento. Mas adelante se presenta el diagrama de bloques y se describe el funcionamiento de cada bloque desde lo general hasta lo particular. Finalmente se describe la simulacion de la arquitectura utilizando el lenguaje Modula-2 y se analizan los resultados.

6.1. Redes Petri

Las redes Petri son herramientas que sirven para modelar formalmente el flujo de informacion en los sistemas. Su principal caracteristica es que son muy poderosas para describir sistemas que presentan concurrencia y que son sincronos.

En la figura 6.1. se presenta una red Petri. Cuando una red Petri es dibujada mediante una grafica, entonces se están modelando las propiedades estaticas del sistema. En la figura 6.1. es posible distinguir dos tipos de nodos: circulos, llamados lugares, y barras, llamadas transiciones. Estos nodos se conectan mediante arcos que pueden ir de un lugar a una transicion o de una transicion a un lugar. Si un arco se dirige de un nodo i a un nodo j, entonces i será la entrada y j sera la salida. En la figura 6.1. el lugar P_1 es la entrada de la transicion t_2, mientras que los lugares P_3 y P_4 son las salidas de t_2.

Ademas de las propiedades estaticas de la grafica de una red Petri, se tienen las propiedades dinamicas resultantes de la ejecucion. La ejecucion de una red Petri esta representada mediante marcadores (tokens) que definen la posicion y el movimiento. Los marcadores se representan mediante puntos negros que se colocan dentro de los circulos de la grafica. Una grafica con marcadores se conoce como grafica marcada. Para el movimiento de los marcadores se tienen las siguientes reglas:

1. Un marcador se mueve cuando se dispara una transicion.
2. Para que una transicion se dispare es necesario que se encuentre habilitada.
3. Una transicion esta habilitada cuando todas sus entradas contienen un marcador.
4. El disparo de una transicion consume los marcadores de sus entradas y genera nuevos marcadores en sus salidas.

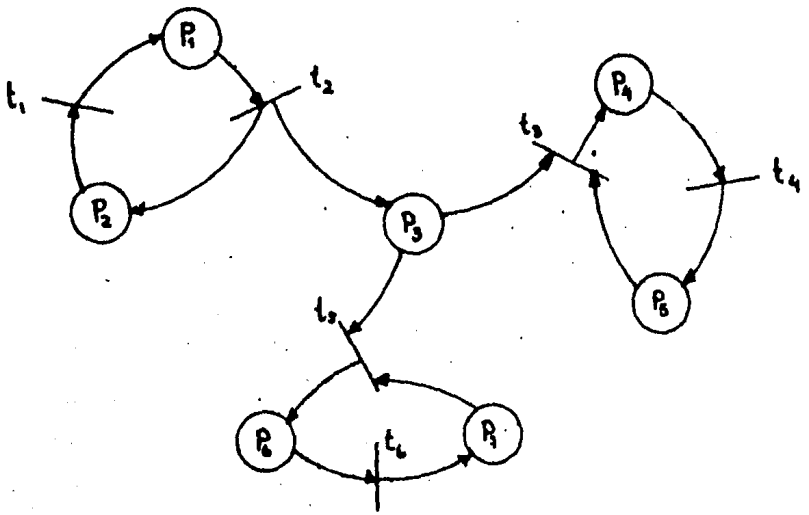


Figura 6.1. Una grafica simple que representa a una red Petri.[38]

. En la figura 6.2. se presenta una grafica marcada. Se puede ver que la transicion t_2 esta habilitada dado que tiene un marcador en su unica entrada P_1. Por otro lado la transicion t_5 esta deshabilitada dado que en P_3 no se tiene marcador. Cuando se dispara la transicion t_2, el marcador de P_1 se consume y se generan marcadores en P_3 y en P_2 que son las salidas de t_2.

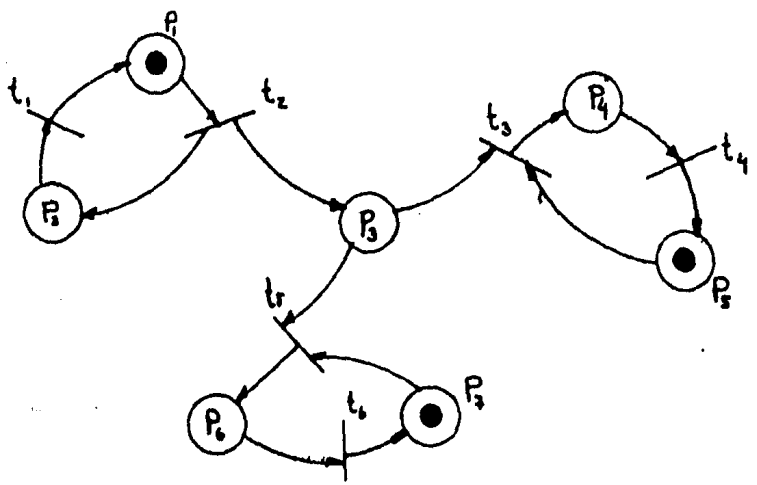


Figura 6.2. Grafica de una red Petri marcada.[38]

En la figura 6.3. se muestra la grafica despues de que t_2 se ha disparado. Ahora se tienen habilitadas las transiciones t_1, t_3 y t_5. Se tienen tres posibilidades dependiendo de cual transicion sea la siguiente en dispararse. En la figura 6.4. se presentan estas tres posiblidades.

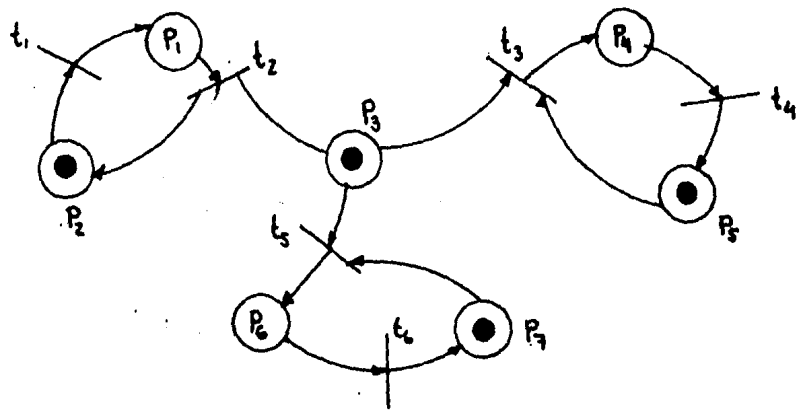
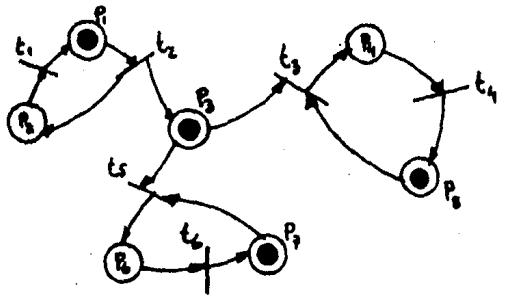


Figura 6.3. Grafica marcada como resultado de disparar t_2 en la figura 6.2. Notar que el marcador de P_1 ha sido consumido y se han generado marcadores en P_3 y P_2.[38]

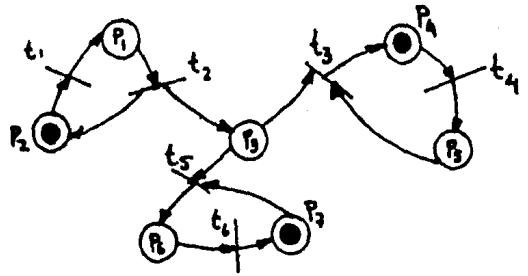
En la figura 6.4.a. se ha disparado la transición t_1 quedando habilitadas t_2, t_3 y t_5; si después t_2 se dispara, entonces habra 2 marcadores en p_3. En la figura 6.4.b. se disparo t_3 quedando habilitadas t_4 y t_1, sin embargo t_5 está deshabilitada debido a que se consumio el marcador de p_3. Por otro lado, en la figura 6.4.c. se disparo t_5 y se deshabilito a t_3, por lo que si se dispara t_3 ó t_5 se deshabilita a la otra, por esto se dice que están en conflicto.

Las redes Petri pueden ser estudiadas como automatas o como generadores de lenguajes formales y son buenos modelos de sistemas concurrentes particularmente de sistemas de computo concurrentes [38].

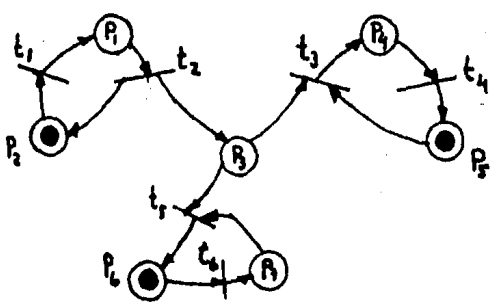
En la siguiente seccion vamos a presentar el modelo de la arquitectura utilizando esta herramienta.



(a)



(b)



(c)

Figura 6.4. Gráficas marcadas como resultado de distintas transiciones disparadas de la figura 6.3. a) dispero de t_1, b) disparo de t_3, c) disparo de t_6.

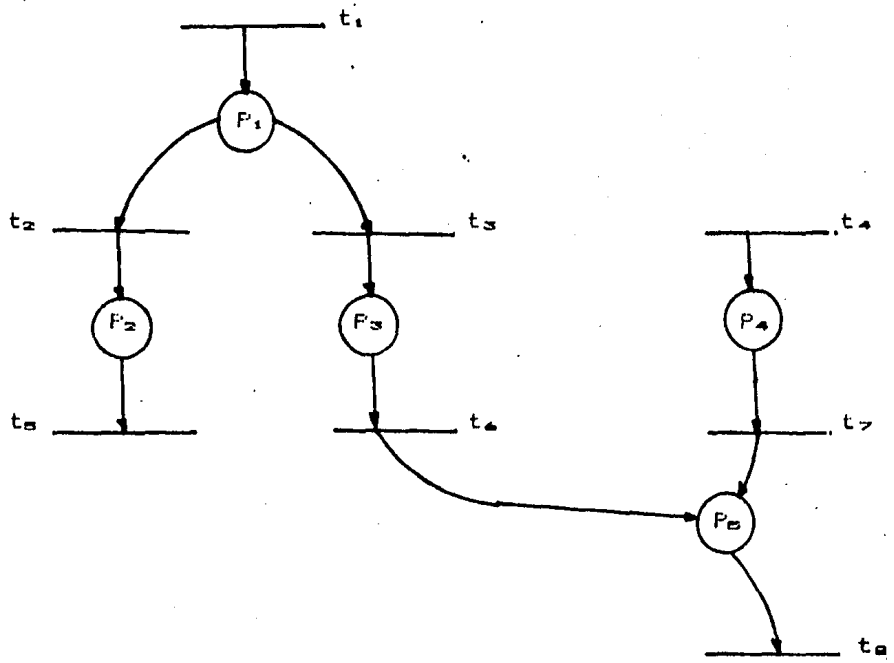
6.2. Modelado

La arquitectura tiene como base a la tecnología pipeline y a los procesadores de flujo de datos. Debido a esta característica es posible que se puedan configurar diferentes tipos de gráficas de redes Petri para diferentes aplicaciones. Sin embargo todas las posibles configuraciones tienen como base al modelo que vamos a presentar a continuación. En la figura 6.5. se tiene la gráfica de la red Petri que modela a la arquitectura. En esta gráfica las transiciones t_1 y t_5, t_4 y t_8 no pertenecen propiamente a la arquitectura, sin embargo forman parte de cualquier sistema y deben estar presentes en la gráfica. Estas transiciones representan a los procesos de entrada y salida. Las transiciones t_2 y t_3 son los procesos que se llevan a cabo dentro del procesador 1 y las transiciones t_6 y t_7 son los procesos que se ejecutan en el procesador 2. Podemos decir que dado que este es el modelo básico de nuestra arquitectura, cuando menos se deberán emplear 2 procesadores.

Las transiciones t_2 y t_7 representan procesos no concurrentes mientras que las transiciones t_3 y t_6 son procesos concurrentes. Podemos distinguir dos tipos de paralelismo en el modelo. El primero es el paralelismo que no comparte variables ni datos y esta dado gracias a las transiciones t_2 y t_7, es decir se pueden estar ejecutando simultaneamente 2 programas distintos, cada uno utilizando sus propios datos y variables y proporcionando sus propias salidas. El segundo es el paralelismo donde se comparte información y es debido a las transiciones t_3 y t_6. Aquí el procesador 1 realiza una parte del proceso y le envía sus resultados al procesador 2, este ultimo toma los datos del primero y concluye la tarea arrojando el resultado por su salida. Desde este punto de vista, podemos decir que el paralelismo obtenido es similar a un pipeline, sin embargo, hay que recordar que este es tan solo el modelo básico y que al configurar redes más grandes es posible llegar a tener una máquina de flujo de datos.

Los dos tipos de paralelismo que se encuentran en el modelo no son necesariamente excluyentes, es decir, se pueden ejecutar programas concurrentes y no concurrentes al mismo tiempo. Para aclarar este concepto hagamos un análisis dinámico para la gráfica de la figura 6.5.

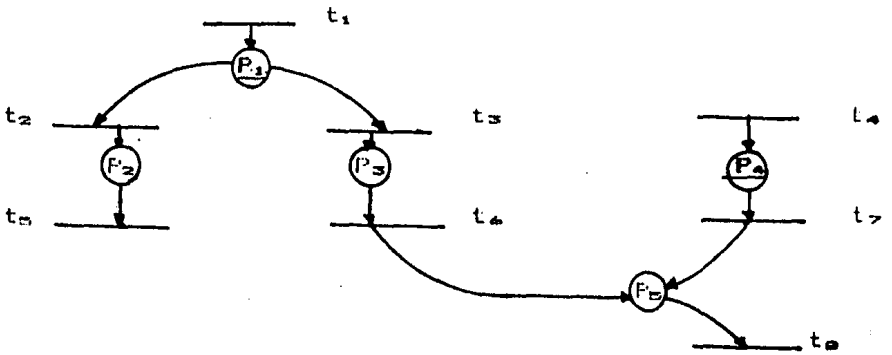
Supongamos primero que llegan datos a los dos procesadores, es decir, se disparan las transiciones t_1 y t_4, entonces los lugares P_1 y P_4 estarán marcados, como se muestra en la figura 6.6.a. En tal caso las transiciones t_2, t_5 y t_7 están listas para dispararse. Supongamos que se disparan las transiciones t_7 y t_2, ver figura 6.6.b. en este caso se han ejecutado dos procesos en paralelo sin compartir información. Si después se disparan las transiciones t_8 y t_9 se obtendrán los resultados de los procesos 1 y 2 respectivamente.



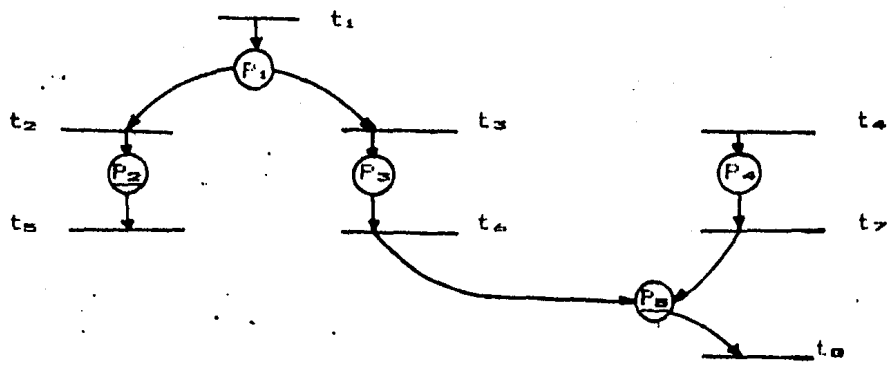
Donde:

t_1= Entrada de datos al procesador 1.
t_2= Proceso no concurrente 1.
t_3= Proceso concurrente 1.
t_4= Entrada de datos al procesador 2.
t_5= Salida de datos del procesador 1.
t_6= Proceso concurrente 2.
t_7= Proceso no concurrente 2.
t_8= Salida de datos del procesador 2.
P_1= Cola de entrada 1.
P_2= Cola de salida 1.
P_3= Comunicación.
P_4= Cola de entrada 2.
P_5= Cola de salida 2.

Figura 6.5. Modelo basico de la arquitectura empleando una grafica de la red Petri.



(a)

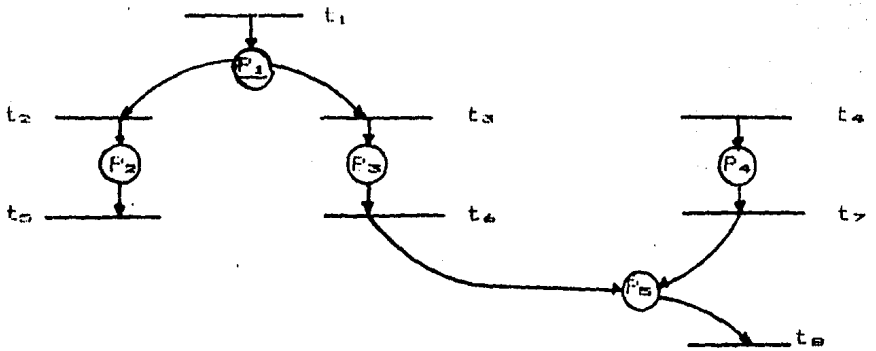


(b)

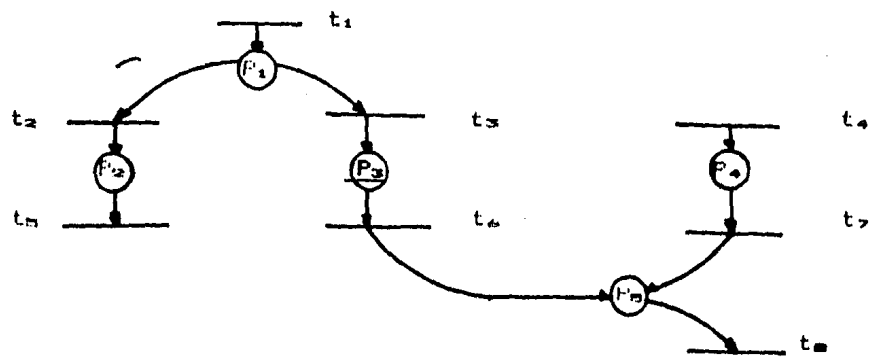
Figura 6.6. Análisis dinámico de la arquitectura para la ejecucion en paralelo de dos programas que no comparten variables.

Como segundo caso supongamos que se dispara t_1, entonces solo el lugar P_1 estara marcado, ver figura 6.7.a. En este momento las transiciones t_2 y t_3 se pueden disparar. Supongamos

que se dispara t_3. En este caso se ejecutará el proceso concurrente 1 y quedará habilitada la transición t_4, como se muestra en la figura 6.7.b. Cuando se dispara la transición t_4 se ejecuta el proceso concurrente 2 y la salida se obtendrá al dispararse t_8. La información tuvo un flujo del procesador 1 al procesador 2. Se puede ver que si al mismo tiempo se disparan las transiciones t_1,t_3,t_4 y t_8, entonces se tendrá un procesamiento pipeline lineal (ver capítulo 2).



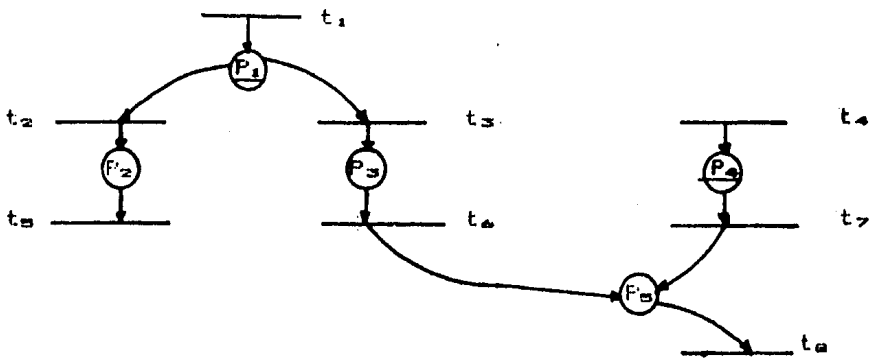
(a)



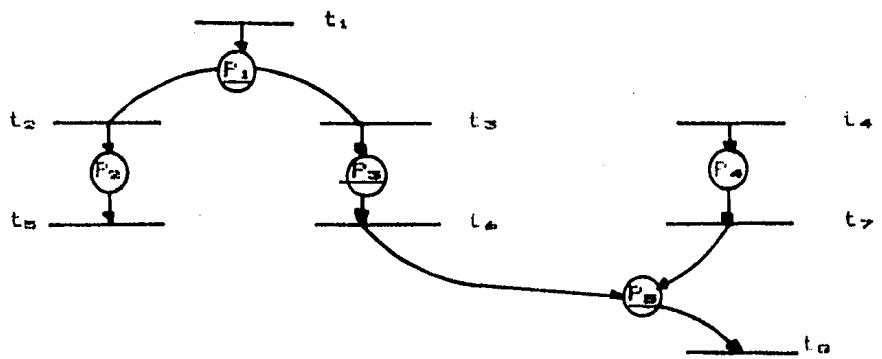
(b)

Figura 6.7. Análisis dinámico de la arquitectura para la ejecución de un proceso concurrente.

Un tercer caso está dado cuando se disparan t_1 y t_4 (fig. 6.8.a.), entonces tanto P_1 como P_4 estan marcados y se encuentran habilitados t_2, t_3 y t_7, supongamos que se disparan t_3, t_7 y t_1 entonces se han ejecutado simultaneamente un proceso concurrente y un proceso no concurrente , ver figura 6.8.b., ademas se han habilitado las transiciones t_2 y t_3. Supongamos ahora que se disparan t_8,t_4 y t_2 (figura 6.8.c.), entonces se habra terminado de ejecutar el proceso concurrente, se habrá obtenido la salida del primer proceso no concurrente y se habra ejecutado un segundo proceso no concurrente. Cuando se disparan t_5 y t_6 se obtienen las salidas del proceso concurrente y del segundo proceso no concurrente. De esta forma se puede ver que el modelo permite tanto paralelismo de procesos independientes como paralelismo de procesos que comparten informacion. Ademas se ha demostrado que no son excluyentes.

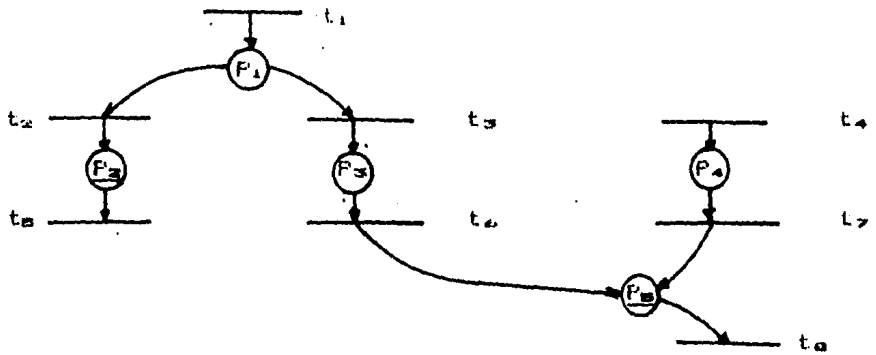


(a)



(b)

Figura 6.8. Analisis dinamico de la arquitectura para la ejecucion de un proceso concurrente y dos procesos no concurrentes. (continua en la siguiente pagina)



(c)

Figura 6.8. Analisis dinamico de la arquitectura para la ejecucion de un proceso concurrente y dos procesos no concurrentes. (continuacion)

Ya que ha quedado bien definido el modelo basico, podemos ejemplificar algunas posibles configuraciones de redes que utilicen mas de dos procesadores. El modelo basico puede crecer en dos sentidos, el primero es en forma vertical y el segundo es en forma horizontal, ademas puede darse el caso de un crecimiento combinado. Para el primer tipo de crecimiento, el paralelismo que se esta explotando es el temporal (tipo pipeline). Para el segundo caso estaremos explotando un paralelismo espacial. De esta forma vemos que dependiendo de las necesidades del problema el modelo se puede reconfigurar. A continuacion vamos a dar ejemplos de crecimiento vertical y horizontal.

En el crecimiento vertical queremos aumentar un nivel mas, para poder segmentar una tarea en más subprocesos. Esto lo logramos, facilmente, si a nuestro modelo basico le aumentamos en la salida del procesador 2 otro procesador como se muestra en la figura 6.9. Las transiciones t_3 y t_{10}, t_6 y t_7, t_{10} y t_{11} representan a los procesadores 1,2 y 3 respectivamente. t_1, t_4 y t_8 son entradas mientras que t_{10}, t_9 y t_{12} son las salidas. Los procesos concurrentes son t_3, t_6, y t_{10}. Los procesos no concurrentes son t_2, t_7 y t_{11}.

En el crecimiento horizontal queremos aumentar un procesador dentro de un mismo nivel para repartir el trabajo dentro de ese nivel. En la figura 6.10 se muestra este crecimiento, las transiciones t_1, t_2 y t_6 son las entradas mientras que t_{12}, t_9 y t_{13} son salidas. El procesador 1 contiene a las transiciones t_3, t_4 y t_5, mientras que t_7 y t_8, t_{10} y t_{11} son los procesadores 2 y 3 respectivamente. Procesos concurrentes son t_3, t_8, t_5 y t_{10}, puede observarse que ahora el procesador 1 tiene dos procesos concurrentes. Los procesos no concurrentes son t_7, t_4 y t_{11}.

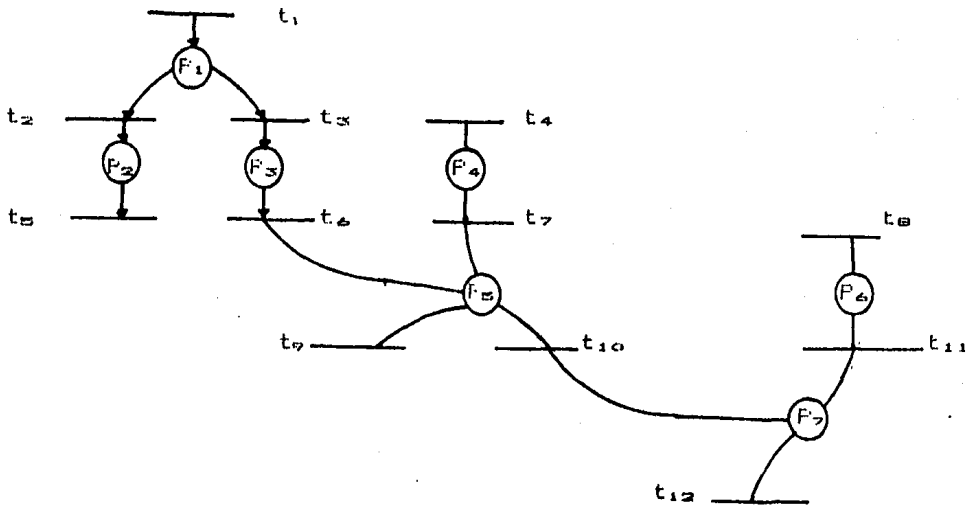


Figura 6.9. Crecimiento vertical del modelo basico.

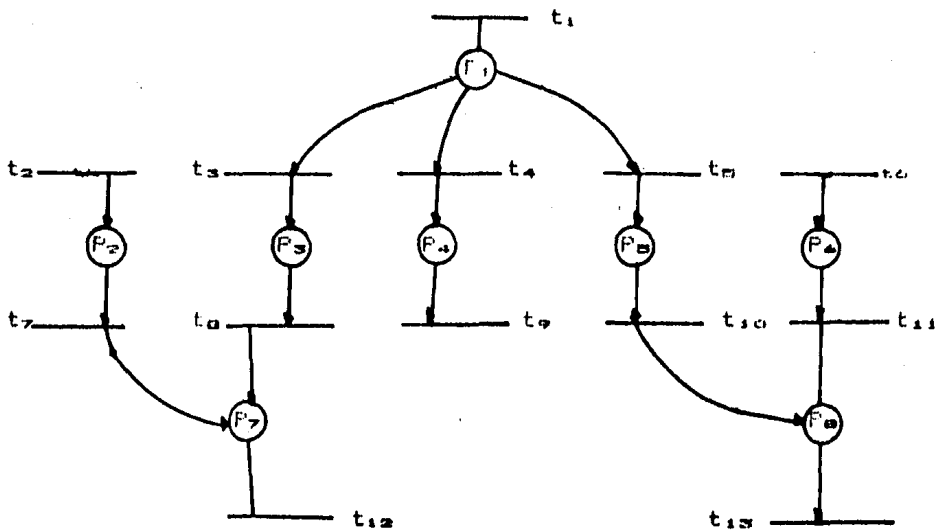


Figura 6.10. Crecimiento horizontal del modelo basico.

Estos dos tipos de crecimiento no son excluyentes, por lo que se pueden generar redes combinadas, sin embargo, el costo aumentara en forma significativa por lo que solo es recomendable realizar redes combinadas en aplicaciones que verdaderamente lo requieran. En la siguiente seccion vamos a desarrollar una arquitectura empleando el modelo basico.

6.3. La Arquitectura.

La arquitectura que se presenta en esta seccion utiliza al microprocesador TMS32010. Se ha empleado este microprocesador porque ha sido diseñado para el analisis de senales. El conjunto de instrucciones que maneja permite realizar en un solo ciclo de reloj muchas de las operaciones mas comunmente empleadas en el analisis de señales de voz. Es, ademas, uno de los microprocesadores mas rapidos que se tienen en el mercado. Su ciclo de reloj es de 200 ns y debido a que esta construido con procesamiento pipeline, puede ejecutar instrucciones en 1, 2 y 3 ciclos de reloj. Puesto que el TMS32010 utiliza procesamineto pipeline para procesar las instrucciones, la arquitectura que aqui presentamos tiene paralelismo a nivel instruccion, a nivel proceso y a nivel programa. Hay que remarcar que el microprocesador se ha escogido para que se adapte mejor a las necesidades de nuestros problemas, sin embargo, desde el punto de vista del modelo basico presentado en la seccion anterior, el microprocesador que se escoja no altera las propiedades de dicho modelo.

En esta seccion se describe en forma general una arquitectura que satisface el modelo basico de la figura 6.5., despues se detalla mas a fondo en cada uno de los bloques de la arquitectura.

6.3.1 Diagrama de bloques general

La forma como se realiza la descripcion es de lo general a lo particular. El diagrama de bloques general que satisface al modelo basico se compone de tres bloques, como se muestra en la figura 6.11. En este diagrama podemos ver que se tienen 2 bloques (modulo 1 y modulo 2) unidos mediante un bloque de comunicacion. La comunicacion se presenta en un solo sentido, es decir, de un modulo a otro, ademas pueden estar presentes ambos sentidos. Sin embargo, para cada sentido es necesario que se tenga un bloque de comunicacion. En el diagrama general unicamente tomaremos el sentido propuesto por la figura 6.10. es decir, de arriba hacia abajo. Una caracteristica muy importante de este diagrama es que ambos modulos son independientes, es decir, no comparten ni el bus de datos, ni el bus de control, ni

el bus de direcciones, ni localidades de memoria ni puertos de E/S,ni el reloj. Esta caracteistica hace que la implementación se facilite considerablemente.

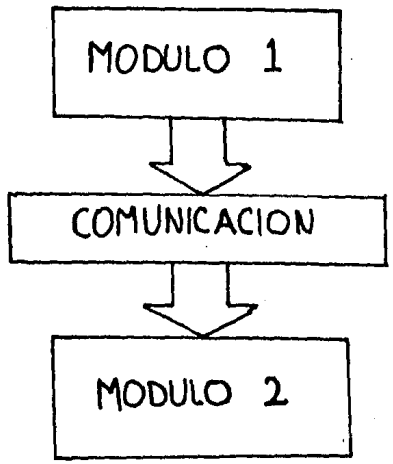


Figura 6.11. Diagrama de bloques general de la arquitectura.

En la figura 6.12. se ha dividido el bloque de comunicacion en dos partes, ademas se han detallado las lineas de control. Estos dos subloques son el manejador de bandera y el almacenamiento. El manejador de bandera es un bloque muy importante dentro del sistema, esta compuesto por circuitos logicos y su funcionamiento se explica mas adelante. Esencialmente, la funcion de esta bloque es mantener el estado del almacenamiento, dicho de otra forma, el manejador de bandera informa a los dos modulos si es posible leer o escribir en el almacenamiento. Esto se logra mediante un solo bit, si el bit es un "1", indica que el almacenamiento está lleno y que el modulo 2 puede tomar el dato, en este caso el modulo 1 solo podra escribir otro dato hasta que el almacenamiento haya sido leido. Cuando el bit es "0", indica que el almacenamiento esta vacio y que el modulo 1 puede depositar otro dato en el, el modulo 2 debera esperar a que el almacenamiento este lleno para poder leer el dato. El manejador de bandera tiene propiedades internas que facilitan y agilizan la comunicación, estas propiedades se detallan en las siguientes secciones. Por su parte, el bloque de almacenamiento es un registro de 16 bits. Este registro tiene separadas las entradas de las salidas para que ambos modulos permanezcan aislados. El control de la escritura se realiza independientemente del control de la lectura. Sin embargo, no se

puede realizar las dos funciones al mismo tiempo debido a que los datos serán poco confiables. Para evitar una colisión entre la lectura y la escritura, se utiliza el manejador de bandera. Podemos observar que la función del registro de almacenamiento es muy similar al la función que desempeñan los registros en los procesadores pipeline (ver capitulo 2.). Existe una diferencia básica, en los procesadores pipeline el registro está controlado por una señal externa (clock) que establece la frecuencia de cambio en los datos del registro, en este caso los procesadores están sincronizados y si un proceso toma más tiempo que los demás se genera un cuello de botella dentro del sistema, provocando que algunos procesadores estén ociosos en espera de un nuevo dato. En esta arquitectura, el paso de la información se realiza cuando es posible hacerlo, por lo que el sistema es completamente asíncrono, esta característica permite que los procesadores puedan realizar tanto ejecuciones de programas compartiendo variables como de programas independientes, reduciéndose los tiempos de ocio. Resumiendo, la arquitectura permite establecer un procesamiento pipeline que puede llegar a convertirse en un flujo de datos al no trabajar sincronamente.

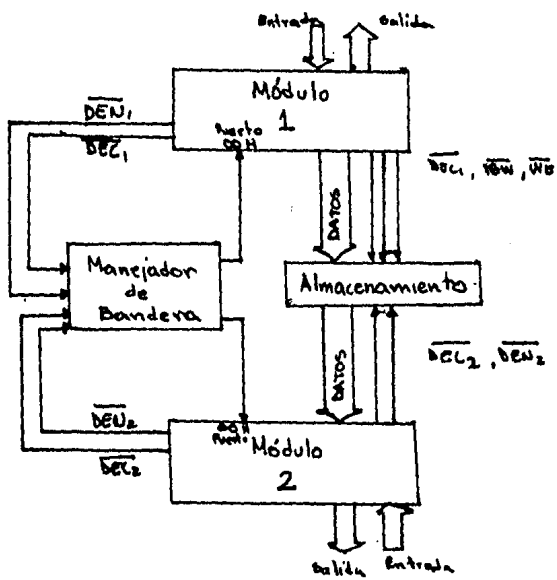


Figura 5.12. Subdivisión del bloque de comunicación.

Las líneas de entrada y de salida de ambos módulos pueden ser tanto para señales analógicas (si tiene los convertidores necesarios) como para señales digitales y el número de estas señales está determinado por el microprocesador que se utilice; si se cuenta con interfases adecuadas es posible tener terminal de video, almacenamiento en disco y comunicación remota, pero estas interfases no forman parte de la arquitectura general.

6.3.2. El almacenamiento.

En esta parte proponemos una implementación para el registro de almacenamiento. El registro está formado por cuatro módulos que almacenan 4 bits cada uno, en la figura 6.13. se tiene un diagrama de la conexión de estos 4 módulos. Se ha seleccionado al circuito integrado SN74LS670 debido a que satisface con todos los requerimientos. Este CI tiene un tiempo de acceso típico de 20 ns. Tiene dos entradas para la selección de la escritura y dos entradas para la selección de lectura, W_A , W_B y R_A R_B respectivamente. Estas cuatro líneas serán comunes en los cuatro módulos teniendo como entradas las señales siguientes: [1] $\overline{DEC1}$ a W_A y W_B, $\overline{DEC2}$ a R_A y R_B. La primera señal es del módulo 1 y controla la escritura, la última señal es del módulo 2 y controla la lectura. Para habilitar al CI para lectura o escritura se tienen las líneas de G_W y G_R. Estas dos líneas estarán conectadas a las señales $\overline{WE_1}$ y $\overline{DEN_2}$ provenientes de los módulos 1 y 2 respectivamente. Las 16 salidas Q_1-Q_{16} estarán conectadas en paralelo al bus de datos del módulo 2. Se puede ver que en ningún momento se han mezclado los buses de datos y que tampoco se han compartido líneas de control. En la figura 6.14. se presenta el diagrama funcional del CI SN74LS670, la señal G_R controla las 4 salidas quedando en alta impedancia cuando G_R tiene un nivel alto. En esta implementación únicamente se estará utilizando la palabra 0 (word 0), dejándose las 3 restantes para posibles mejoras del diseño.

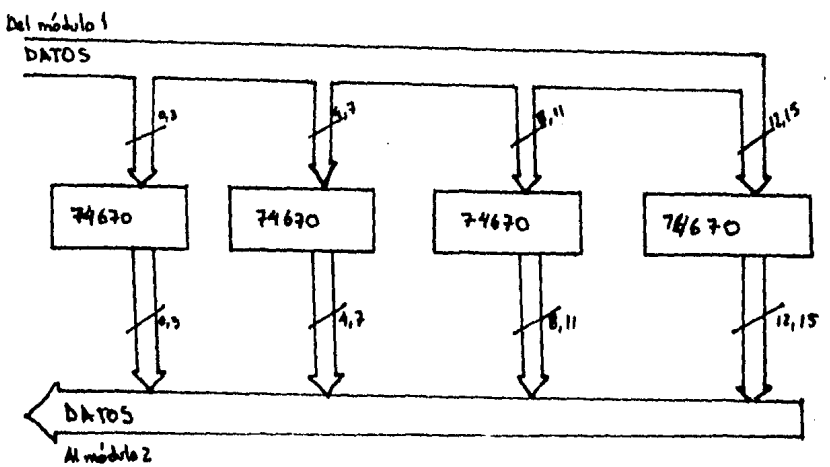


Figura 6.13 Diagrama de bloques del registro para comunicación.

[1] Las variables lógicas subrayadas indican el negado.

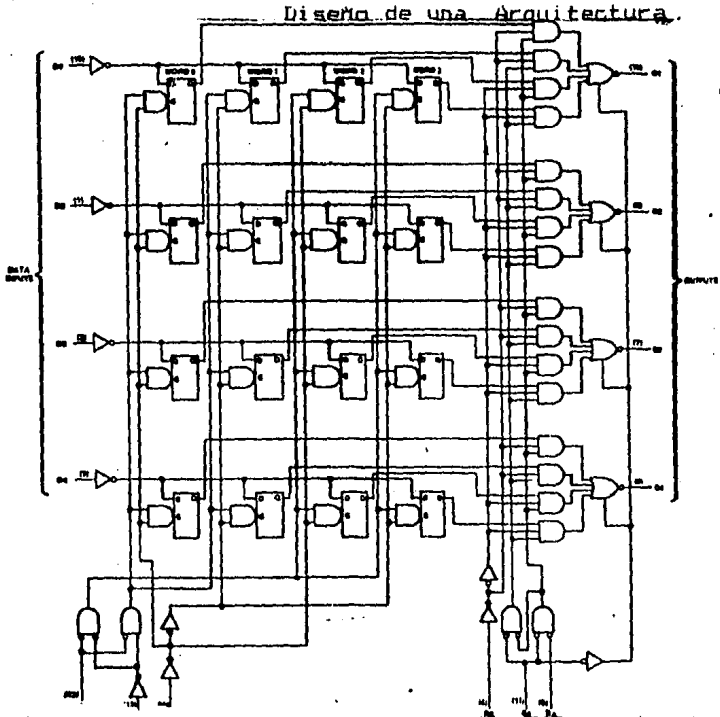
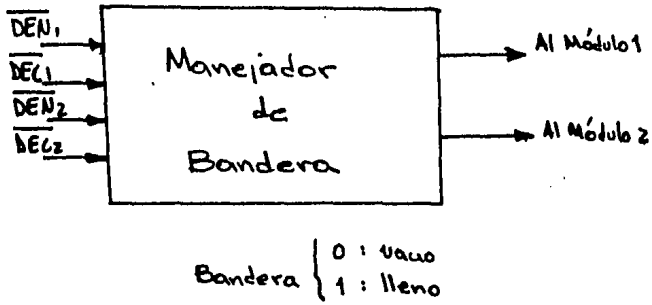


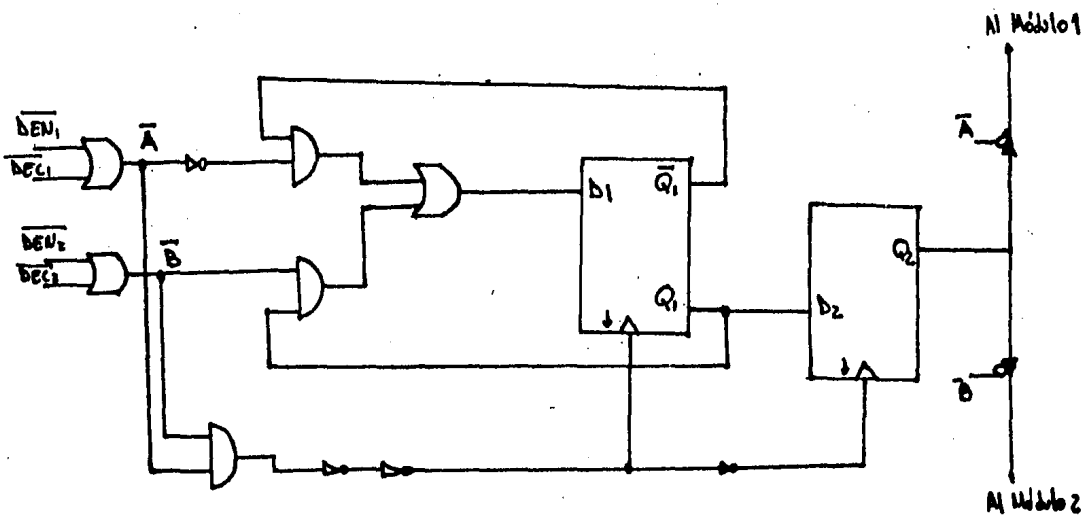
Figura 6.15. Diagrama funcional del 74LS670.[39]

6.3.3. El manejador de bandera

El manejador de bandera es un bloque con cuatro entradas y dos salidas, como se muestra en la figura 6.15. La finalidad es controlar el acceso del registro, evitar las colisiones y cambiar el estado de la bandera al leer o escribir en el registro. La forma en como opera es la siguiente: cuando el modulo 1 quiere escribir en el registro, primero pregunta por el estado de la bandera, bajando las señales de $\overline{DEN_1}$ y $\overline{DEC_1}$, el circuito de tres estados permite el paso de la señal Q_2 al modulo 1; simultaneamente, mediante la lógica que entra al D_1 se escribe un 1 (lleno) en el flip-flop 1, que al subir las señales DEN_1 y DEC_2 sera escrito en el flip-flop 2. Entonces el modulo 1 tiene el estado Q_{n-1} del manejador de bandera y Q_n=1. Con esto se nos presentan dos posibilidades, si Q_{n-1}=1 no es posible que se escriba en el registro y se debe preguntar por el estado Q_n hasta que este sea igual a cero, por otra parte si Q_{n-1}=0 se puede escribir en el registro sin temor a perder información. Una caracteristica importante es que con una sola instruccion de lectura se puede conocer el estado del registro y modificar la bandera, dicho en otras palabras, mediante la utilización de una sola instruccion de lectura estamos realizando también una escritura, esto nos genera un ahorro de dos ciclos de reloj que son muy importantes cuando se transmiten bloques de datos. Además nos da la posibilidad de que el modulo 2 pregunte por la bandera cuando el modulo 1 escribe en el registro y viceversa.



(a)



(b)

Figura 6.15. El manejador de bandera, (a) diagrama de bloques; (b) diagrama funcional.

El manejador de bandera trabaja asincronamente, pudiendo recibir señales de los dos modulos al mismo tiempo. Cuando se presentan ambas señales se genera un conflicto, es decir, si el modulo 1 pregunta por la bandera quiere dejar el estado Q_n en 1 y si el modulo 2 pregunta por la bandera al mismo tiempo, quiere cambiar el estado Q_n a cero. Es evidente que se genera un conflicto y que el estado de la bandera es incierto. Para resolver este problema se genero la siguiente tabla de verdad.

Q_n	A	B	D_1	
0	1	0	1	*
0	1	1	1	
0	0	0	0	
0	0	1	0	
1	1	0	0	*
1	1	1	1	
1	0	0	0	
1	0	1	1	

Marcados con asteriscos estan los estados que generan conflicto. Si A=1 indica que el modulo 1 quiere escribir si $\overline{B}$=0 indica que el modulo 2 quiere leer. La forma en como se resolvio el conflicto es utilizando el estado de la bandera para decidir, es decir, si el estado actual de la bandera es Q_1=0 indica que el registro actualmente está vacio y que se debe dar prioridad a llenarse, por esto es que D_1=1. En el caso contrario Q_1=1 indica que el registro esta lleno y que se debe vaciar. Hay que notar que el trabajo del flip-flop2 es fundamental, mientras en el flippflop1 se esta modificando el estado de la bandera, el flip-flop2 mantiene el estado anterior y la lectura hecha por ambos modulos es confiable y segura, una vez que el dato del flip-flop1 es capturado, se pasa al flip-flop2.

Ambos flip-flops son disparados con borde de bajada, y este disparo es generado por el primer modulo que lea la bandera, es por esto que se realiza la funcion AND entre $\overline{A}$ y $\overline{B}$. Supongase que la señal $\overline{A}$ se genera primero (el modulo quiere leer la bandera), entonces se cumple 0 AND 1 = 0 , por lo que se genera un borde de bajada en la entrada del flip-flop1, si un instante despues la señal $\overline{B}$=0 (el modulo 2 quiere leer), entonces se cumple que 0 AND 0 = 0 y no se genera borde de bajada. La señal $\overline{A}$ debe subir primero y no se genera borde de subida, pero cuando $\overline{B}$=1 entonces se cumple que 1 AND 1 = 1 y se genera un borde de subida en el flip-flop1 mientras que en el flip-flop2 se genera un borde de bajada y el dato de la bandera se transfiere del FF1 al FF2.

El doble inversor sirve para retardar la señal del reloj y evitar que se tome un dato falso. Con esto demostramos que el manejador de bandera responde a una sola señal a la vez aunque se presentarán ambas exactamente al mismo tiempo. Esta caracteristica enfatiza que el sistema es asincrono.

Actualemte ya se ha implementado este modulo, puesto que, para fines de la simulacion se necesita saber que es posible realizar un bloque con capacidad de leer la bandera y al mismo tiempo actualizar su estado. Es por esto que se ha detallado en su funcionamiento. Sin embargo, puede ser que exista una solucion mejor o que se deba realizar alguna modificacion al diseño actual. Por lo pronto, solo se pretende demostrar que si es factible un bloque manejador de bandera con las caracteristicas que se proponen.

6.3.4. El módulo

El modulo es un bloque cuya complejidad depende de las posiblidades materiales y economicas con que se cuenta. Este bloque representa a un microprocesador y en realidad no se especifica que microprocesador utilizar. Sin embargo, como se pretende usar esta arquitectura en el procesamiento de senales se ha seleccionado al TMS32010 porque cumple con muchas caracteristicas, entre otras que es muy veloz. En la figura 6.16. se presenta el minimo modulo necesario para poder trabajar con la arquitectura. Podemos ver que esta basado en el microprocesador seleccionado. Este microprocesador en su conjunto de instrucciones tiene la instruccion IN <dir>, que realiza una lectura del puerto direccionado. Esta instruccion tarda en ejecutarse tan solo 2 ciclos de reloj, sin embargo, solamente se pueden direccionar ocho puertos P_0-P_7 con las tres lineas de direccionamiento menos significativas. Esta restriccion limita la implementacion de la arquitectura pero facilita el manejo de la bandera, como se explica a continuacion: cuando se ejecuta una instruccion de lectura de puerto, el microprocesador enciende la senal $\overline{DEN}$, esta senal es baja solamente con la instruccion IN por lo que resulta facil determinar cuando se quiere leer la bandera, si proporcionamos ademas la decodificacion del puerto. Entonces las lineas para controlar el manejador de bandera son unicamente $\overline{DEN}$ y $\overline{DEC}$ (decodificacion de la direccion del puerto). Para controlar el registro se utilizan las señales de decodificacion de la direccion del puerto y $\overline{WE}$, cuando el modulo es de escritura, y cuando el modulo es de lectura se usa la decodificacion $\overline{DEN}$, hay que notar que en el primer caso se puede usar un solo puerto (00H) para la lectura de la bandera y la escritura del registro, en el segundo caso es necesario utilizar dos puertos uno para la lectura de la bandera (00H) y otro para la lectura del registro (01H), por lo que en ultimo caso las lineas de $\overline{DEC}$ seran distintas. Los puertos que sobran pueden ser utilizados para entrada y salida de datos analogicos y/o digitales. En el diagrama de la figura 6.16. se han dibujado dos decodificadores para facilitar el diagrama. Sin embargo, no necesariamente tiene que ser distintos. El bloque marcado como puertos tiene dentro un MUX de 3 a 8, las 16 lineas de entrada son el bus de datos; este bloque puede trabajar en forma bidireccional para permitir la E/S de los datos. Se tiene un arreglo de memoria que maximo puede ser de 4 Kbytes debido a que es la capacidad maxima de direccionamiento del TMS32010, se ha dividido en dos secciones, la parte superior es una memoria EPROM que almacenara el programa que se quiera ejecutar (concurrente o no concurrente), en las localidades inferiores se ha colocado una memoria RAM que almacenara los datos y variables que se esten utilizando. El tiempo de acceso en estas memorias debe de ser de cuando mas 100 ns debido a que es la velocidad que requiere el microprocesador. Esta caracteristica hace que el precio de las memorias sea elevado.

Un bloque muy importante es el del reloj, cada modulo tiene su propio reloj (para fines de implementacion se puede usar uno solo) con lo que se demuestra una vez mas que el sistema no

requiere de sincronizacion.

Esto es lo minimo necesario que debera tener el modulo, sin embargo, se puede pensar en aumentar interfases para poder manejar terminales de video, impresoras , unidades de disco suave, convertidores A/D y D/A, etc. Tambien es factible que un modulo sea mas complicado que el otro pero esto no afecta la eficiencia de la arquitectura. Se ha propuesto utilizar, para fines de implementacion, los modulos de evaluacion hechos en base al TMS32010 que cuentan con todo lo necesario para la ejecucion de programas, desde el punto de vista de hardware y software (incluso un ensamblador).

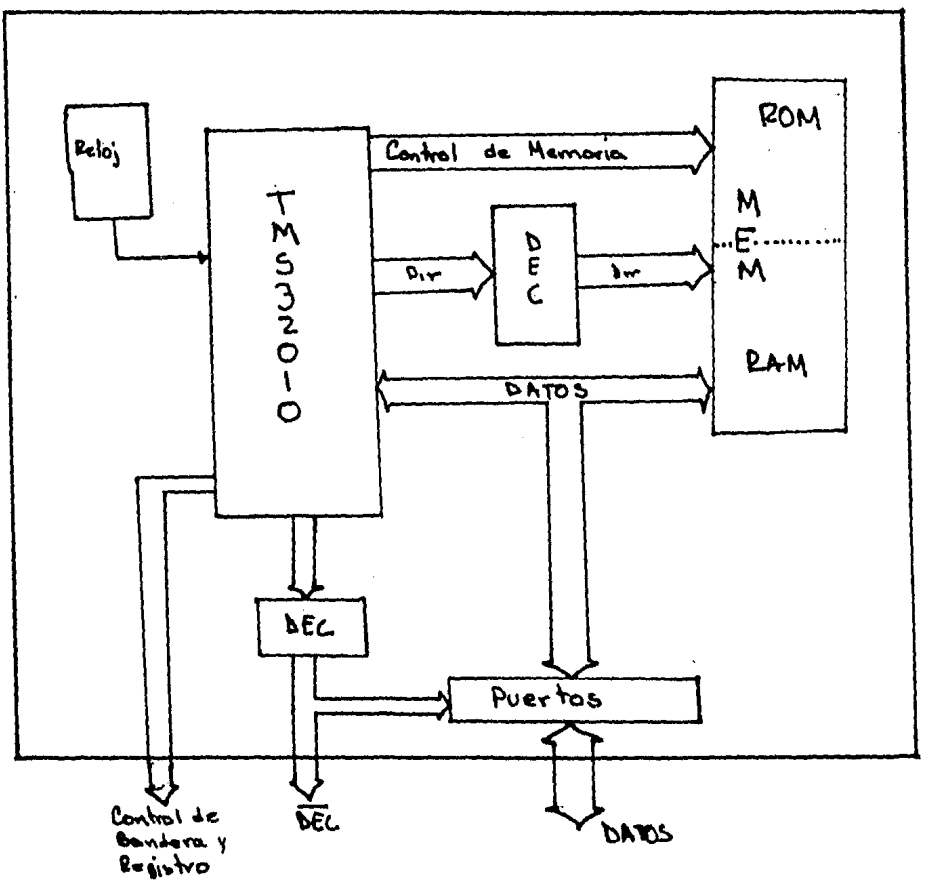


Figura 6.16. Diagrama de bloques del modulo.

6.4 Simulación de la Arquitectura

Se realizo una simulacion de la arquitectura en el lenguaje de programacion Modula-2 en una computadora PC compatible con los siguientes objetivos:

+ Evaluar la arquitectura.
+ Tener una herramienta para el desarrollo de software.

Aunque es posible evaluar la arquitectura, dadas sus especificaciones, teoricamente, la disponibilidad de un simulador facilita el estudio de la arquitectura, principalmente en el caso de programas más complejos.

Un simulador en software de un sistema en hardware como el nuestro es una herramienta muy util para la evaluacion, depuracion y creacion de programas, sobre todo en un caso como este en el que se trata de programas escritos en ensamblador.

Se eligio Modula-2 para implementar el simulador debido a que es un lenguaje con las mismas cualidades de Pascal ademas de contar con tres propiedades para nosotros interesantes: concurrencia, modulos y facilidades de bajo nivel.

El manejo de concurrencia lo realiza Modula-2 utilizando monitores (ver capitulo 5) y lo simula en sistemas con un solo procesador. El concepto de modulo es introducido con el objeto de lograr un nivel adicional de estructuracion en los programas; son bloques de mas alto nivel que los procedimientos y pueden ser compilados independientemente. Una descripcion del lenguaje se encuentra en [30]; el compilador utilizado fue LOGITECH MODULA-2/86.

a) Descripción del programa.

El programa consta de los siguientes modulos: correarq, arqui, banderafi], registrofi] y tmsfi].

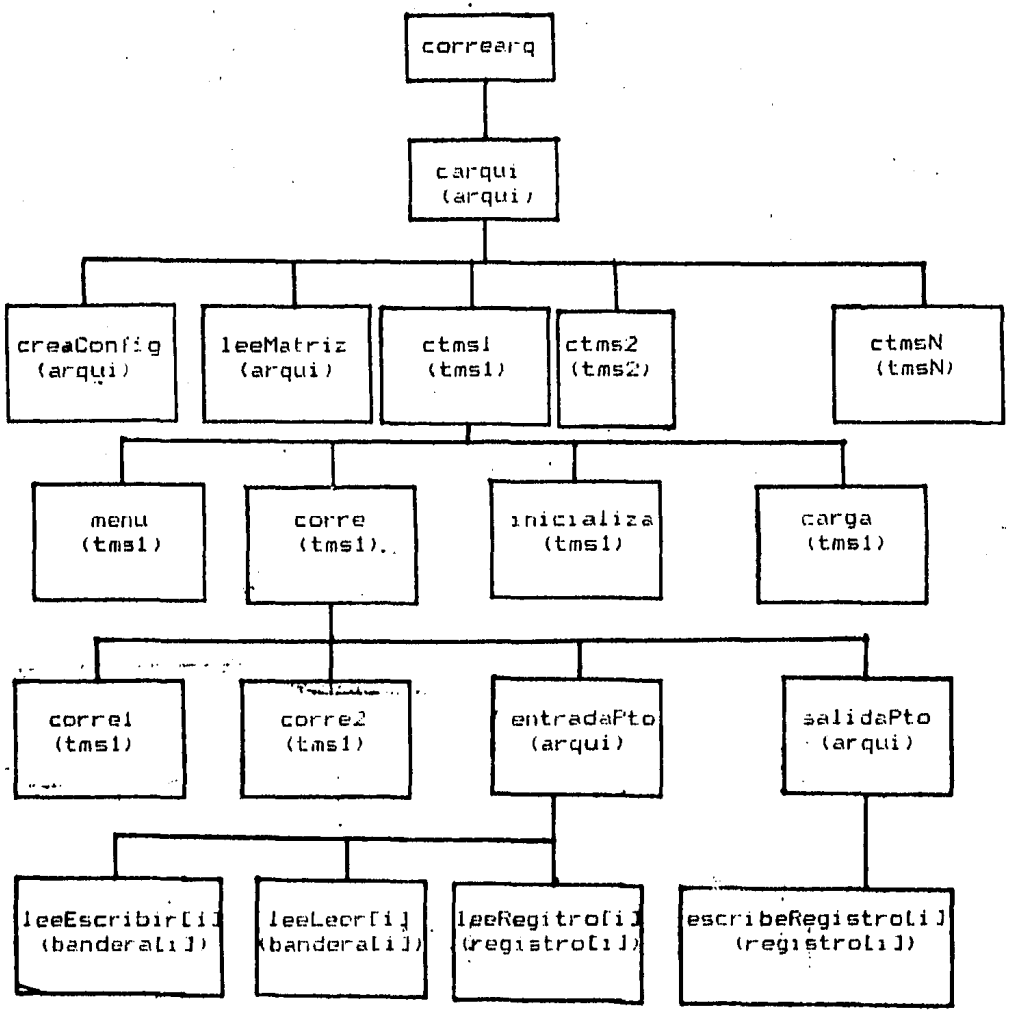
El modulo correarq (76 bytes de codigo) sirve unicamente para activar todo el sistema. En su version ejecutable (86550 bytes con 4 tms's y 3 comunicaciones), correarqui.lod es en donde se encuentra el codigo ejecutable de todo el sistema.

El modulo arqui (6863 bytes de codigo) se encarga de establecer la interconexion entre los registros, banderas y tms's ademas de controlar la concurrencia y activar los procesos concurrentes.

Los modulos que simulan a los registros (273 bytes cada uno) a traves de los cuales se pasan datos los tms s, se llaman registro, registro1, registro2, etc. De la misma forma, a cada registro le corresponde una bandera (329 bytes cada uno) que indica si el registro contiene un dato que no ha sido leido (lleno) o no (vacio). Estos modulos se llaman bandera, bandera1, bandera2, etc.

Finalmente tenemos los modulos que se encargan de simular la operacion de un microprocesador TMS32010 con reloj, memoria y los decodificadores necesarios (3111 bytes cada uno). Estos modulos se llaman tms1, tms2, tms3, etc.

A continuacion se muestra un diagrama de bloques en el que se puede observar la jerarquia entre los principales procedimientos del simulador. Entre parentesis se indica el modulo en el cual esta definido el procedimiento respectivo.



A continuacion se listan las definiciones y encabezados de los modulos que conforman el sistema con el objeto de aclarar el anterior diagrama de bloques, de mostrar las variables utilizadas

y de servir como referencia a la explicacion mas detallada que sigue del sistema.

El módulo arqui.

Este modulo realiza tres funciones, su definicion es la siguiente:

```
(* Definicion de la arquitectura del sistema *)
DEFINITION MODULE arqui;
 FROM Processes IMPORT SIGNAL;
 EXPORT QUALIFIED tuOyo,carqui,entradaPto,salidaPto;

 VAR tuOyo:SIGNAL;  (* para que corran concurrentemente todos los tms *)
 PROCEDURE carqui;  (* activa todo el sistema *)
 PROCEDURE entradaPto(quien,cual:INTEGER;VAR que:INTEGER);(* El tms 'quien' *)
                                                          (* hace un lN del puerto *)
                                                          (* 'cual'. 'que' se le manda *)
                                                          (* como dato de entrada. *)
 PROCEDURE salidaPto(quien,cual,que:INTEGER); (* El tms 'quien' hace un OUT *)
                                              (* al puerto 'cual' del dato *)
                                              (* 'que'. *)

END arqui.
```

La primera funcion es la de activar el número requerido de simuladores de TMS; número que el usuario especifica mediante la variable "numerotms". A continuación se lista el procedimiento en el que se realiza esta funcion.

```
PROCEDURE carqui;
BEGIN
  WriteString("desea crear una nueva configuracion del sistema?(s/n) ");
  Read(c);WriteLn;
  IF c="s" THEN creaConfig END;
  leeMatriz; escribeMatriz;
  Init(tuOyo);
  IF numerotms>0 THEN StartProcess(ctms1,5000) END;
  IF numerotms>1 THEN StartProcess(ctms2,5000) END;
  IF numerotms>2 THEN StartProcess(ctms3,5000) END;
  IF numerotms>3 THEN StartProcess(ctms4,5000) END;
  WHILE Awaited(tuOyo) DO
    SEND(tuOyo)
  END
END carqui;
```

Como se aprecia en el listado, esta funcion se logra mediante dos pasos: primero se activan los tms's mediante la instruccion StartProcess y luego se transfiere el control a cada uno de los tms's mediante la instrucción SEND(tuOyo). Cada tms tiene instrucciones WAIT(tuOyo) de tal forma que el control se alterna entre todos los tms's activados.

La segunda función es la de leer y escribir en disco la interconexion que se desea que haya entre los tms's, las banderas y los registros. La matriz matEnt[i,j] indica de donde proviene la entrada del tms[i], en el puerto j; es decir, si proviene de terminal, de un registro, de una bandera o de algun archivo. La matriz matSal tiene un objetivo similar pero para las salidas de

los tms's. En un archivo se encuentra el número de tms's de la arquitectura y las matrices de entradas y salidas.

Finalmente, el módulo arqui tiene la función de establecer propiamente la interconexión del sistema, para lo cual exporta dos procedimientos que los tms's utilizan: entradaPto y salidaPto. al realizar las instrucciones de IN y OUT respectivamente. Como se puede ver en el siguiente listado, estos procedimientos dirigen las entradas y salidas de los tms's de la forma indicada por el usuario mediante las matrices matEnt y matSal.

```
PROCEDURE salidaPto(quien,cual,que:INTEGER); (* quien: que tms *)
VAR tmp:INTEGER;                              (* cual: que puerto *)
BEGIN                                         (* que: lo que hay que sacar *)
  WriteString("OUT: tms ");WriteInt(quien,2);WriteString("   puerto ");
  WriteInt(cual,2);WriteString("   dato>");WriteInt(que,5);
  IF (quien<=mtms)AND(cual<mpuertos)THEN tmp:=matSal[quien,cual]
  ELSE tmp:=0; WriteString("ERROR: OUT, quien o cual ");
       WriteInt(quien,3);WriteInt(cual,3);WriteLn
  END;
  CASE tmp OF
    1:
       escribeRegistro(que);WriteString(" ( a reg. 1 )")  |
    2:
       escribeRegistro1(que);WriteString(" ( a reg. 2 )") |
    3:
       escribeRegistro2(que);WriteString(" ( a reg. 3 )")
    ELSE
       WriteString(" ( a terminal )")
    END;
    WriteLn
END salidaPto;

PROCEDURE entradaPto(quien,cual:INTEGER;VAR que:INTEGER);
VAR tmp:INTEGER;
BEGIN
  que:=0;

  WriteString("IN: tms ");WriteInt(quien,2);WriteString("   puerto ");
  WriteInt(cual,2); WriteString("   dato>");
  IF (quien<=mtms)AND(cual<mpuertos)THEN tmp:=matEnt[quien,cual]
  ELSE tmp:=0; WriteString("ERROR: IN, quien o cual ");
       WriteInt(quien,3);WriteInt(cual,3);WriteLn
  END;
  CASE tmp OF
    1:
       que:=leeEscribir();WriteString(" ( de bandera 1, para esc. )")  |
    2:
       que:=leeLeer()     ;WriteString(" ( de bandera 1, para leer )")  |
    3:
       que:=leeRegistro();WriteString(" ( de registro 1 )")             |
    4:
       que:=leeEscribir1();WriteString(" ( de bandera 2, para esc. )") |
    5:
       que:=leeLeer1()    ;WriteString(" ( de bandera 2, para leer )")  |
    6:
       que:=leeRegistro1() ;WriteString(" ( de registro 2 )")           |
    7:
       que:=leeEscribir2();WriteString(" ( de bandera 3, para esc. )") |
```

```
    8:
       que:=leeLeer2()     ;WriteString(" ( de bandera 3, para leer )")  |
    9:
       que:=leeRegistro2();WriteString(" ( de registro 3 )")
    ELSE
      ReadInt(que);WriteString(" ( de terminal )")
    END;

    WriteInt(que,5); WriteLn
END entradaPto;
```

Una consecuencia importante de este módulo es que no es necesario modificar y recompilar programas cuando se modifican las interconexiones entre modulos, ya que esto se hace mediante nuevos valores en las matrices matEnt y matSal.

Los módulos de banderas y registros.

Los modulos de banderas (tipo monitor) indican mediante la variable "estado" si su respectivo registro está lleno (estado=1) o vacio (estado=0). Estos modulos exportan dos procedimientos que son utilizados por el modulo arqui: leeLeer y leeEscribir. El primero regresa el contenido de la bandera cuando el tms quiere leer el contenido del registro; leeEscribir también regresa el contenido de la bandera pero el tms lo llama cuando lo que quiere hacer es colocar un nuevo dato en el registro. Su funcionamiento se puede ver en los listados de definición e implementacion del modulo:

```
DEFINITION MODULE bandera;

 EXPORT QUALIFIED leeLeer,leeEscribir;

 (* 0: vacio; 1: lleno *)

 PROCEDURE leeEscribir():INTEGER;

 PROCEDURE leeLeer():INTEGER;

END bandera.
```

```
IMPLEMENTATION MODULE bandera01;

VAR estado:INTEGER;

PROCEDURE leeEscribir():INTEGER;
VAR t:INTEGER;
BEGIN
  t:=estado;
  estado:=1;
  RETURN t
END leeEscribir;
```

```
PROCEDURE leeLeer():INTEGER;
VAR t:INTEGER;
BEGIN
  t:=estado;
  estado:=0;
  RETURN t
END leeLeer;

BEGIN
  estado:=0
END bandera
```

Los moudos de registros administran las variables que contienen los datos de los registros. Exportan los procedimientos necesarios para leer y escribir los registros como se puede ver en el siguiente listado.

```
DEFINITION MODULE registro;

 EXPORT QUALIFIED leeRegistro,escribeRegistro;

 PROCEDURE leeRegistro():INTEGER;

 PROCEDURE escribeRegistro(que:INTEGER);

END registro.

IMPLEMENTATION MODULE registro;

VAR contenido:INTEGER;

PROCEDURE leeRegistro():INTEGER;
BEGIN
 RETURN contenido
END leeRegistro;

PROCEDURE escribeRegistro(que:INTEGER);
BEGIN
  contenido:=que
END escribeRegistro;

BEGIN
 contenido:=0
END registro.
```

Los módulos tms.

Estos modulos se llaman tms1, tms2, etc. Cada uno de estos modulos simula la operación de un TMS32010 con memoria, decodificaciones y reloj. Como se puede ver en el menu de un tms que se lista a continuación, el simulador permite ver y modificar el contenido de los registros internos y de la memoria del procesador asi como colocar puntos de alto (breakpoints) dentro del programa, ejecución paso a paso y modificar el contenido del reloj.

Para indicar que está esperando su turno para correr, cada tms tiene una instrucción WAIT(tuyo); el módulo arqui manda continuamente señales con "tuyo".

```
PROCEDURE menu;
BEGIN
  WriteString("         MENU ");WriteString(nombtms);WriteLn;
  WriteString("m...menu");WriteLn;
  WriteString("c...alterar/ver pc");WriteLn;
  WriteString("a...alterar/ver acumulador");WriteLn;
  WriteString("A...ver registros auxiliares");WriteLn;
  WriteString("d...imprimir memoria de datos");WriteLn;
  WriteString("p...imprimir memoria de programa");WriteLn;
  WriteString("P...ejecucion paso a paso; 'c' para dejarla");WriteLn;
  WriteString("s...imprimir stack");WriteLn;
  WriteString("r...corre");WriteLn;
  WriteString("R...alterar/ver reloj");WriteLn;
  WriteString("L...alterar/ver limite del reloj");WriteLn;
  WriteString("n...cargar un nuevo programa");WriteLn;
  WriteString("f...terminar");WriteLn;
  WriteString("1...alto #1");WriteLn;
  WriteString("2...alto #2");WriteLn
END menu;
```

En la siguiente sección del procedimiento "corre" se muestra como se logra la interacción entre un módulo tms y el resto del sistema mediante llamadas a los procedimientos "entradaPto" y "salidaPto".

```
IF bits(tbuf,11,14)=0 THEN                                    (* IN  *)

  WAIT(tuOvo); (* instruccion de 2 ciclos *)

  entradaPto(quetms.puerto,dato);

  IF b7=0 THEN
    Mdata[dma]:=dato
  ELSE
    Mdata[Ar[Arp]]:=dato;
    indirecto(b0,b3,b4,b5)
  END; reloj:=reloj+1
 END;
IF bits(tbuf,11,14)=9 THEN                                    (* OUT *)

  WAIT(tuOvo); (* instruccion de 2 ciclos *)

  IF b7=0 THEN
    dato:=Mdata[dma]
  ELSE
    dato:=Mdata[Ar[Arp]];
    indirecto(b0,b3,b4,b5)
  END;

 salidaPto(quetms.puerto,dato);

 reloj:=reloj+1
END;
```

```
IMPLEMENTATION MODULE arqui;
FROM InOut IMPORT WriteString,WriteLn,WriteInt,ReadInt,EOL,OpenInput,
                  CloseInput,OpenOutput,CloseOutput,Done,WriteCard,Read,
                  ReadCard;
FROM Processes IMPORT SEND,StartProcess,Init,Awaited;
FROM tms1 IMPORT ctms1;
FROM tms2 IMPORT ctms2;
FROM tms3 IMPORT ctms3;
FROM tms4 IMPORT ctms4;
FROM registro IMPORT leeRegistro,escribeRegistro;
FROM bandera  IMPORT leeLeer,leeEscribir;
FROM registr1 IMPORT leeRegistro1,escribeRegistro1;
FROM bandera1 IMPORT leeLeer1,leeEscribir1;
FROM registr2 IMPORT leeRegistro2,escribeRegistro2;
FROM bandera2 IMPORT leeLeer2,leeEscribir2;

(* EXPORT QUALIFIED tuOvo; *)

(* VAR tuOvo:SIGNAL; *)

CONST mpuertos=8; (* numero de puertos *)
      mtms=4; (* maximo numero de tms's *)
      mcomunic=3; (* maximo numero de comunicaciones: parejas de banderas y *)
                  (* registros.                                              *)
```

Ampliación del sistema.

En la implementacion actual del sistema están cargados cuatro tms's y tres comunicaciones (parejas de banderas y registros), de tal forma que se puede especificar cualquier interconexión entre estos elementos sin necesidad de modificar el programa. Se puede indicar el número de tms's que se quieren correr en paralelo y se puede ir deteniendo la operación de cada uno de ellos mientras que los demás siguen corriendo.

Si se tiene necesidad de simular una arquitectura con más tms's o más comunicaciones es necesario copiar el módulo deseado, cambiarle de nombre y compilarlo. Es necesario también cambiar el modulo arqui para indicar la nueva disponibilidad de módulos, compilarlo y ligar todo el sistema nuevamente. Para crear un nuevo tms es necesario modificar el nombre del módulo, las constantes que contienen el nombre y el número de módulo ("nombtms" y "quetms") y el nombre del procedimiento "ctms".

Para crear una nueva comunicación se cambian los nombres de los modulos y de los procedimientos que exportan.

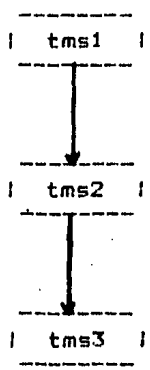
En el modulo "arqui" se debe indicar la disponibilidad de un nuevo elemento en la lista de módulos importados, modificando el máximo número de tms's (mtms) y/o de comunicaciones (mcomunic), ampliando las listas de los procedimientos salidaPto y entrada Pto, y si es necesario incluyendo una nueva instrucción de StartProcess en el procedimiento "carqui". Solamente es necesario recompilar los módulos que hayan sido modificados.

b) Resultados.

A continuación se presentan algunos ejemplos de programas que fueron corridos en el simulador de la arquitectura, junto con algunos resultados que dan una idea del desempeño del sistema.

Ejemplo 1: Comunicacion de datos.

En este ejemplo se pretende obtener el número de ciclos de reloj que toma pasar datos de un tms a otro, y además mostrar como se comunican entre si los módulos. En primer lugar se utilizo una arquitectura tipo pipeline como la de la siguiente figura:



con tres programas diferentes; uno para el primer tms:

```
7001 *LARK 1   TRANSMISION DE DATOS ( etapa inicial )
4188 *IN *,1   ( dato de terminal )
4200 *IN 0,2   ( bandera )
2000 *LAC 0
FE00 *BNZ 2
0002
48AB *OUT *+,0   ( al siguiente tms )
F900 *B 1
0001
```

uno para los tms's intermedios:

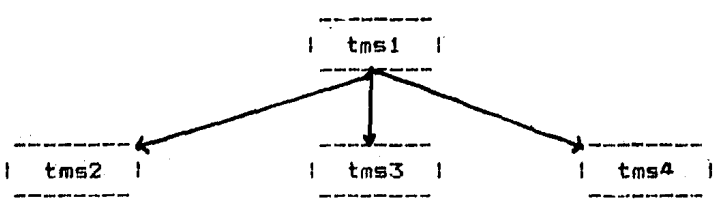
```
7001 *LARK 1   TRANSMISION DE DATOS ( etapa intermedia )
4000 *IN 0,0 ( bandera )
2000 *LAC 0
FF00 *BZ 1
0001
4188 *IN *,1   ( de registro )
4200 *IN 0,2   ( de bandera )
2000 *LAC 0
FE00 *BNZ 6
0006
48AB *OUT *+,0    ( a registro )
```

y otro para la etapa final:

```
7001 *LARK 1   TRANSMISION DE DATOS ( etapa final )
4000 *IN 0,0    ( bandera )
2000 *LAC 0
FF00 *BZ 1
0001
4188 *IN *,1    ( registro )
```

```
45A8 *OUT *+,0    ( terminal )
F900 *B 1
0001
```

La segunda configuracion utilizada fue la siguiente:



Para este caso el procedimiento del tms1 fue:

```
7001 *LARK 1 TRANSMISION DE DATOS ( en paralelo )
4188 *IN *,1 ( terminal )
4200 *IN 0,2 ( bandera 1 )
2000 *LAC 0
FE00 *BNZ 2
0002
4888 *OUT *,0   ( a tms 1 )
4300 *IN 0,3    ( bandera 2 )
2000 *LAC 0
FE00 *BNZ 7
0007
4988 *OUT *,1   ( a tms 2 )
4400 *IN 0,4    ( bandera 3 )
2000 *LAC 0
FE00 *BNZ 12
```

```
000C
4AA8 *OUT *+,2    ( a tms 3 )
F900 *B 1
0001
```

y el del resto de los tms s fue el de la etapa final de la primera configuracion.

Los resultados obtenidos fueron:

Configuración I:

a) Dos tms's.
   15 ciclos para pasar el primer dato.
   11 ciclos por cada dato adicional.

b) Tres tms's.
   25 ciclos para el primer dato.
   16 ciclos por dato adicional.

c) Cuatro tms's.
   35 ciclos para el primer dato.
   16 ciclos por dato adicional.

Configuracion II:

a) Cuatro tms's.
   26 ciclos para pasar el primer dato.
   25 ciclos por dato adicional.

Como se puede ver, para la configuración I, la utilización de los procesadores es del 100% (si tomamos en cuenta los ciclos para preguntar por la bandera, la utilización sería de alrededor del 50%) ya que estos se sincronizan y nunca esperan por un dato. Se obtienen las frecuencias de operación máximas de 11 ciclos/dato y 16 ciclos/dato para dos procesadores y más de dos procesadores respectivamente. Para la configuración II la utilización de los procesadores no es más que del 64% (en este caso si hay ocaciones en que los tms's deben esperar por un dato) para los tms2, tms3 y tms4 ya que estos reciben datos solamente cada 25 ciclos.

Ejemplo 2: Filtros digitales.

Se implementó un filtro digital de orden 17 basado en el diseño de [31]. Los listados de los programas utilizados son los siguientes.

tms1

```
7E01 *LACK 1
5023 *SACL 1,0
7F89 *ZAC
1023 *SUB 1,0
5024 *SACL MINUS,0
6A23 *LT 1
8003 *MPYK
7F8E *PAC
7010 *LARK 0,16
7111 *LARK 1,CX1
6881 *LARP1:RCONST
67A0 *TBLR *+,0
0023 *ADD 1
F400 *BANZ RCONST
001E
7F80 *NOP
7F80
4100 *IN X1,1
7010 *LARK 0,X17
7121 *LARK 1,CX17
7F89 *ZAC
6A91 *LT *-,1
6D90 *MPY *-,0
6881 *LTD *,1:LOOP
6D90 *MPY *-,0
F400 *BANZ LOOP
002B
7F8F *APAC
0023 *ADD 1,14
5022 *sacl y
407F *in 7F,0 bandera
207F *lac 7F
FE00 *bnz 50
0032
4822 *OUT Y,0
F900 *B WAIT
0023
```

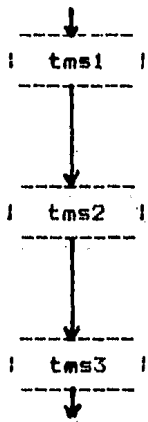
tms2

```
7E01 *LACK 1
5023 *SACL 1,0
7F89 *ZAC
1023 *SUB 1,0
5024 *SACL MINUS,0
6A23 *LT 1
8003 *MPYK
7F8E *PAC
7010 *LARK 0,16
7111 *LARK 1,CX1
6881 *LARP1:RCONST
67A0 *TBLR *+,0
0023 *ADD 1
F400 *BANZ RCONST
001E
F800 *CALL 35
0039 *
4100 *IN X1,1
7010 *LARK 0,X17
7121 *LARK 1,CX17
7F89 *ZAC
6A91 *LT *-,1
6D90 *MPY *-,0
6881 *LTD *,1:LOOP
6D90 *MPY *-,0
F400 *BANZ LOOP
002B
7F8F *APAC
0023 *ADD 1,14
5022 *sacl y
427F *IN BANDERA
207F *LAC
FE00 *BNZ WAIT
0032
4822 *OUT Y,0
F900 *B WAIT
0023
4050 *IN 50,0
2050 *LAC
FF00 *BZ
0039
7F80 *RET
```

tms3

```
7E01 *LACK 1
5023 *SACL 1,0
7F89 *ZAC
1023 *SUB 1,0
5024 *SACL MINUS,0
6A23 *LT 1
8003 *MPYK
7F8E *PAC
7010 *LARK 0,16
7111 *LARK 1,CX1
6881 *LARP1:RCONST
67A0 *TBLR *+,0
0023 *ADD 1
F400 *BANZ RCONST
001E
F800 *CALL 35
0035 *
4100 *IN X1,1
7010 *LARK 0,X17
7121 *LARK 1,CX17
7F89 *ZAC
6A91 *LT *-,1
6D90 *MPY *-,0
6881 *LTD *,1:LOOP
6D90 *MPY *-,0
F400 *BANZ LOOP
002B
7F8F *APAC
0023 *ADD 1,14
5022 *sacl y
4922 *OUT Y,1
F900 *B WAIT
0023
4039 *IN 39,0
2039 *LAC
FF00 *BZ
0035
7F80 *RET
```

con la configuración pipeline, que conecta a los filtros en cascada, representada en la figura que sigue:



Los resultados obtenidos fueron:

a) Un tms.
78 ciclos/dato.

b) Dos tms s.
87 ciclos/dato.

c) Tres tms's.
92 ciclos/dato.

d) Cuatro tms's.
92 ciclos/dato.

De estos datos se obtienen aceleraciones (speedups) de alrededor de 1.8, ya que hay que tomar en cuenta que en casi el mismo número de ciclos se realiza un filtrado del doble de orden. Por ejemplo, con dos tms's se realiza un filtro de orden 34 en 87 ciclos.

Ejemplo 3: Búsqueda de vectores.

En este programa se trata de localizar el vector al que más se parezca un vector de datos leído. La medida de distancia utilizada es la usual y se hace la búsqueda exaustiva. Cada tms tiene almacenados la mitad de los patrones. El algoritmo funciona de la siguiente forma: El tms1 localiza el vectos más parecido al vector leído y le manda al tms2 la direccion y la distancia que encontro; el tms2 a su vez busca el vector, obtiene una dirección y una distancia entre sus vectores patrón. Compara las dos distancias y da como resultado la dirección apropiada. Los programas utilizados y los resultados obtenidos (en número de ciclos para hallar un dato, en promedio) se encuentran a continuación.

tms1

```
7F80 *BUSCA UN VECTOR ENTRE PATRONES
7F80 *
7E0A *LACK 10
5014 *SACL 20
F800 *CALL 66   (INICIALIZA)
0042
F800 *CALL 109    (LEEDATO)
006D
2007 *LAC 07
5000 *SACL 00
3804 *LAR 04
6881 *LARP 1
3905 *LAR 05
6880 *LARP 0
20A1 *LAC *+,1
10A0 *SUB *+,0
F800 *CALL 95     (COMPARA)
005F
FE00 *BNZ 47
002F
2000 *LAC 00
1001 *SUB 1
5000 *SACL 0
FE00 *BNZ 13    (OTRO ELEMENTO)
000D
2016 *LAC 22
1015 *SUB 21
FA00 *BLZ  35   (ES MAYOR)
0023
2015 *LAC 21
5016 *SACL 22
3117 *SAR 23
2017 *LAC 23
1007 *SUB 7
5002 *SACL 02
7E00 *LACK 0
5015 *SACL 21
3804 *LAR 4,0
2003 *LAC 03
1007 *SUB 7
5003 *SACL 3
FC00 *BGZ  13    (OTRO ELEMENTO)
000D
F800 *CALL 134   (SALIDA)
0086
F900 *B    02     (NUEVO VECTOR)
0002
2007 *LAC 7
1000 *SUB 0
0001 *ADD 1
3119 *SAR 25
0019 *ADD 25
5019 *SACL 25
3919 *LAR 25
3804 *LAR 4
7E00 *LACK 0
5015 *SACL 21
2003 *LAC 3
```

tms2

```
7F80 *BUSCA UN VECTOR ENTRE PATRONES
7F80 *
7E0A *LACK 10
5014 *SACL 20
F800 *CALL 66   (INICIALIZA)
0042
F800 *CALL 109    (LEEDATO)
006D
2007 *LAC 07
5000 *SACL 00
3804 *LAR 04
6881 *LARP 1
3905 *LAR 05
6880 *LARP 0
20A1 *LAC *+,1
10A0 *SUB *+,0
F800 *CALL 95     (COMPARA)
005F
FE00 *BNZ 47
002F
2000 *LAC 00
1001 *SUB 1
5000 *SACL 0
FE00 *BNZ 13    (OTRO ELEMENTO)
000D
2016 *LAC 22
1015 *SUB 21
FA00 *BLZ  35   (ES MAYOR)
0023
2015 *LAC 21
5016 *SACL 22
3117 *SAR 23
2017 *LAC 23
1007 *SUB 7
5002 *SACL 02
7E00 *LACK 0
5015 *SACL 21
3804 *LAR 4,0
2003 *LAC 03
1007 *SUB 7
5003 *SACL 3
FC00 *BGZ  13    (OTRO ELEMENTO)
000D
F800 *CALL 120   (SALIDA)
0078
F900 *B    02     (NUEVO VECTOR)
0002
2007 *LAC 7
1000 *SUB 0
0001 *ADD 1
3119 *SAR 25
0019 *ADD 25
5019 *SACL 25
3919 *LAR 25
3804 *LAR 4
7E00 *LACK 0
5015 *SACL 21
2003 *LAC 3
```

```
1007 *SUB 7
5003 *SACL 03
FC00 *BGZ  13      (OTRO ELEMENTO)
000D
F800 *CALL  134  (SALIDA)
0086
F900 *B   02      (NUEVO DATO)
0002
7F80 *<<< INICIALIZACION >>>>
7E01 *LACK 1
5001 *SACL 1
7E0A *LACK 10
5004 *SACL 4
7E1E *LACK 30
5005 *SACL 5
7E0B *LACK 11
5003 *SACL 03
2014 *LAC 20
5016 *SACL 22
2003 *LAC 3
5002 *SACL 2
3805 *LAR 5,0
7E96 *LACK 150
5008 *SACL 8
2008 *LAC 8
67A8 *TBLR *+
0001 *ADD 1
5008 *SACL 8
2002 *LAC 2
1001 *SUB 1
5002 *SACL 2
FE00 *BNZ  82  (LEEOTRO)
0052
7E00 *LACK 0
5002 *SACL 2
5008 *SACL 8
7F8D *RET
7F80 *<<< COMPARA >>>
5008 *SACL 8
6A08 *LT 8
6D08 *MPY 8
7F8E *PAC
0015 *ADD 21
5015 *SACL 21
1014 *SUB 20
FB00 *BLEZ  107
006D
7E01 *LACK 1
7F8D *RET
7E00 *LACK 0
7F8D *RET
3804 *LAR 4,0   <<< LEEDATO >>>
7E02 *LACK 2
5000 *SACL 0
5007 *SACL 7
401A *IN 26,0 ;L1
201A *LAC 26
FE00 *BZ  L1
0071
41A8 *IN *+,1
2000 *LAC 0
1001 *SUB 1
5000 *SACL 0
```

```
1007 *SUB 7
5003 *SACL 03
FC00 *BGZ  13      (OTRO ELEMENTO)
000D
F800 *CALL  120  (SALIDA)
0078
F900 *B   02      (NUEVO DATO)
0002
7F80 *<<< INICIALIZACION >>>>
7E01 *LACK 1
5001 *SACL 1
7E0A *LACK 10
5004 *SACL 4
7E1E *LACK 30
5005 *SACL 5
7E0B *LACK 11
5003 *SACL 03
2014 *LAC 20
5016 *SACL 22
2003 *LAC 3
5002 *SACL 2
3805 *LAR 5,0
7E96 *LACK 150
5008 *SACL 8
2008 *LAC 8
67A8 *TBLR *+
0001 *ADD 1
5008 *SACL 8
2002 *LAC 2
1001 *SUB 1
5002 *SACL 2
FE00 *BNZ  82  (LEEOTRO)
0052
7E00 *LACK 0
5002 *SACL 2
5008 *SACL 8
7F8D *RET
7F80 *<<< COMPARA >>>
5008 *SACL 8
6A08 *LT 8
6D08 *MPY 8
7F8E *PAC
0015 *ADD 21
5015 *SACL 21
1014 *SUB 20
FB00 *BLEZ  107
006D
7E01 *LACK 1
7F8D *RET
7E00 *LACK 0
7F8D *RET
3804 *LAR 4,0   <<< LEEDATO >>>
7E02 *LACK 2
5000 *SACL 0
5007 *SACL 7
42A8 *IN *+,2
2000 *LAC 0
1001 *SUB 1
5000 *SACL 0
FE00 *BNZ  113
0071
7F8D *RET
7F80 *<<< SALIDA >>>
```

```
FE00 *BNZ  L13-L1
0071
401A *IN 26,0 :L2
201A *LAC 26
FF00 *BZ L2
0078
411B *IN 27,1
401A *IN 26,0 :L3
201A *LAC 26
FF00 *BZ L3
0080
411C *IN 28,1
7F8D *RET
7F80 *<<< SALIDA >>>
2016 *LAC 22
101C *SUB 28
FA00 *BLZ L4
00BF
4801 *OUT 0,1
481B *OUT 0,27
481C *OUT 0,28
7F8D *RET
7E02 *LACK 2  :L4
501D *SACL 29
481D *OUT 0,29
4802 *OUT 0,2
4816 *OUT 0,22
7F8D *RET
7F80 *NOP
003C *60
003D *61
0046 *70
0047 *71
0050 *80
```

Resultados:

```
3804 *LAR 4,0    <<< ESCRIBE DATO
2007 ** LAC 7
5000 ** SACL 0
7F80
401A **IN 0,26:LOOP
201A **LAC 26
FE00 **BNZ LOOP
007D
48A8 **OUT *+,0
2000 *LAC 0
1001 *SUB 1
5000 *SACL 0
FE00 *BNZ  LOOP
007D
401A *IN 0,26 :L2
201A *LAC 26
FE00 *BNZ L2
0087
4802 *OUT 0,2
401A *IN 0,26  :L3
201A *LAC 26
FE00 *BNZ L3
008C
4816 *OUT 0,22
7F8D *RET
7F80 *NOP
7F80 *NOP
7F80 *NOP
7F80 *NOP
000A *10
000B *11
0014 *20
0015 *21
```

a) Un tms.

435 ciclos/dato (con 10 patrones).

b) Dos tms's.

460 ciclos/dato (con 20 patrones).

c) Conclusiones.

La simulacion es una herramienta útil para el estudio de la arquitectura y para el desarrollo de software para ésta. Debido a que los resultados teoricos y los obtenidos en la simulación coinciden, deducimos que la simulacion se apega al sistema real.

Los resultados de la simulación son prometedores ya que para programas pequeños como los de los ejemplos anteriores ya logramos aceleraciones aceptables, lo que nos hace pensar que para programas mayores los desempeños serán muy buenos.

Conclusiones Generales

1. Consideramos, después de haber estudiado al Procesamiento en Paralelo, que debe de formar parte de los estudios de la licenciatura de Ingenieria en Computacion, ya sea como una materia o cuando menos incluirse como parte de alguna otra materia actual como Organización de Computadoras, Diseño de Sistemas Digitales o Temas Selectos de Computación. La razón es que este campo de la computacion es uno de los más recientes e importantes.

2. La simulación desarrollada demostró que la arquitectura diseñada es capaz de comunicar eficientemente a varios procesadores trabajando en paralelo.

3. La utilización del simulador del TMS32010 que hemos desarrollado en esta tesis proporciona grandes ventajas para el desarrollo de programas tanto para un solo microprocesador como para un sistema con varios microprocesadores.

4. El costo para comunicar a los procesadores está determinado por los componentes del manejador de bandera y por los del registro; como puede verse de las implementaciones propuestas resulta ser muy bajo en comparación con el del resto del sistema (TMS32010 y memorias principalmente).

5. El modelo de envio de mensajes, en el cual está basada la arquitectura, se adapta bien al procesamiento de señales digitales y a los problemas de reconocimiento de patrones como se puede inferir de los programas corridos en el simulador.

Referencias.

[1] K. Hwang y F. Briggs, **Computer Architecture and Parallel Processing**, Mc Graw-Hill, Nueva York, 1984.

[2] L. C. Hobbs, **Parallel Processor Systems, Technologies, and Applications**, Spartan Books, Londres, 1970.

[3] C. U. Ramamoorthy y H. F. Li, "Pipeline Architecture", ACM Computing Surveys, Vol. 9, No. 1, Marzo 1977.

[4] K. Hwang, "Multiprocessor Supercomputers for Scientific/Engineering Applications", Computer, Vol. 18, No. 6, Junio 1985.

[5] J. Allen, "Computer Architecture for Digital Signal Processing", Proc. IEEE, Vol. 73, No. 5, Mayo 1985.

[6] J.T. Rayfield y H. F. Silverman, "An Approach to DFT Calculations Using Standard Microprocessors", IBM J. Res. Develop., Vol. 29, No.2, Marzo 1985.

[7] E. Dekel y S. Shani, "Binary Trees and Parallel Schedualing Algorithms", University of Minnesota, Technical Report 80-19.

[8] H. Lorin, **Parallelism in Hardware and Software**, Prentice-Hall, Inc., Nueva Jersey, 1972.

[9] D. Kuck, "A Survey of Parallel Machine Organization and Programming", ACM Computing Surveys, Vol.9, No. 1, Marzo 1977.

[10] G. R. Andrews y F. B. Schneider, "Concepts and Notations for Concurrent Programming", ACM Computing Surveys, Vol. 15, No. 1, Marzo 1983.

[11] S. Rosen, "Electronic Computers: A Historical Survey", ACM Computing Surveys, Vol. 1, No. 1, Marzo 1969.

[12] P. H. Enslow, "Multiprocessor Organization - A Survey", ACM Computing Surveys, Vol. 9, No. 1, Marzo 1977.

[13] C. P. Dalton, "Multiprocessor: Architecture and Applications", Computer, Vol. 18, No. 6, Junio 1985.

[14] D.D. Gajski, J. Peir, "Essential Issues in Multiprocessor Systems", Computer, Vol. 18, No. 6, Junio 1985.

[15] IEEE Computer Society, "Computing at the Frontiers of Science and Engineering", Computer, Vol. 18, No. 11, Noviembre 1985.

[16] S.Y. Kung, **"VLSI Array Processors"**, IEEE ASSP Magazine, Julio 1985.

[17] K. Murakami, T. Kakuta, R. Onai, N. Ito, **"Research on Parallel Machine Architecture for Fifth-Generation Computer Systems"**, Computer, Vol. 18, No. 6, junio 1985.

[18] Schneck, Austin, Squires, Lehmann, Mizell, Wallgren, **"Parallel Processor Programs in the Federal Government"**, Computer, Vol. 18, No. 6, junio 1985.

[19] A. Guzman, **"A Heterarchical Multi-Microprocessor LISP Machine"**, Reporte Tecnico AHR-81-17, IIMAS, Mexico 1981.

[20] Assosiation of Computing Machinery, **"Special Issue on Supercomputers- Their Impact on Science and Technology"**, Proc. IEEE, Vol. 72, No. 1, enero 1984.

[21] L. Kleinrock, **"Distributed Systems"**, Communications of the ACM, Vol. 28, No. 11, noviembre 1985.

[22] O. Lubeck, J. Moore, R. Mendez, **"A Benchmark Comparition of Three Supercomputers: Fujitsu UP-200, Hitachi S810/20, and Cray X-MP/2"**, Computer, Vol. 18, No.12, Diciembre 1985.

[23] P. C. Treleaven, D. R. Brownbridge, R. P. Hopkins, **"Data-Driven and Demand-Driven Computer Architecture"**, ACM Computing Surveys, Vol. 14, No. 1, marzo 1982.

[24] W. B. Ackerman, **"Data Flow Languages"**, Computer, Vol.15, No. 2, febrero 1982.

[25] A. L. Davis, R. M. Keller, **"Data Flow Program Graphs"**, Computer, Vol. 15, No. 2, febrero 1982.

[26] D. D. Gajski, D.A. Padua, D. J. Kuck, R. H. Kuhn, **" A Second Opinion on Data Flow Machines and Languages,** Computer, Vol. 15, No. 2, febrero 1982.

[27] I. Watson, J. Gurd, **" A Practical Data Flow Computer"**, Computer, Vol. 15, No. 2, febrero 1985.

[28] B. Hansen, **"Concurrent Programming Concepts"**, ACM Computing Surveys, Vol. 5, No. 4, diciembre 1973.

[29] J. L. Baer, **"A Survey of Some Theoretical Aspects of Mutiprocessing"**, ACM Computing Surveys, Vol. 5, No. 1, marzo 1973.

[30] N. Wirth, **Programming in Modula-2**, Springer-Verlag, third, corrected edition, Berlin, 1985.

[31] **TMS32010 User's Guide**, Texas Instruments, EUA, 1983.

[32] S. J. Allan, A. E. Oldehoeft, **"A Flow Analysis Procedure for the Translation of High Level Languages to a Data Flow Language"**, Proc. of the 1979 Int. Conf. on Parallel Processing.

[33] H. Oktaba, **"Programación Concurrente"**, Monografías, No. 86, IIMAS, 1985.

[34] C. A. R. Hoare, **"Communicating Sequential Processes"**, Communications of the ACM, Vol. 21, No. 8, agosto 1978.

[35] J. H. Saltze, **"The Protection of Information in Computer Systems"**, Proceedings of the IEEE, Vol. 63, No. 9, septiembre 1975.

[36] M. J. Gonzalez, JR., **"Deterministic Processor Scheduling"**, Computing Surveys, Vol. 9, No. 3, septiembre 1977.

[37] S. S. Yau, H. S. Fung, **"Associative Processor Architecture-A Survey"**, ACM Computing Surveys, Vol. 9. No. 1, marzo 1977.

[38] J. L. Peterson, **"Petri Nets"**, ACM Computing Surveys, Vol. 9, No. 3, septiembre 1977.

[39] **The TTL Data Book for Design Engineers"**, Texas Instruments, E.U.A., 1981.