CAPITULO 10

TAXONOMIAS DE ARQUITECTURAS DE COMPUTADORAS

10.1. - Introducción

La clasificación de Flynn no discrimina claramente las diferentes arquitecturas de multiprocesadores. Ya que se ha incrementado substancialmente la cantidad de arquitecturas de multiprocesadores, se ha vuelto más importante encontrar una manera útil de describirlas (una forma que distinga aquellas que son significativamente diferentes en tanto que muestre las similitudes subyacentes entre los diseños aparentemente divergentes).

Este capítulo presenta una taxonomía (Skillicorn 1988) para las arquitecturas de computadores que extiende la de Flynn, especialmente en lo que respecta a la categoría de los multiprocesadores.

Esta clasificación no se opone a lo expresado en el capítulo 6 sino que se utiliza, en esta oportunidad, un mecanismo de definición más formal.

Esta taxonomía es una jerarquía de dos niveles, en la cual el nivel superior clasifica las arquitecturas basadas en la cantidad de procesadores para datos y para instrucciones y las interconexiones entre ellos.

El nivel inferior que puede usarse para diferenciar con más precisión las variantes, está basado en una visión de los procesadores como máquinas de estado.

Existe una única taxonomía formal, debida a Flynn, que se usa en general y no alcanza a diferenciar la inmensa variedad de arquitecturas posibles de multiprocesadores. Flynn clasifica las arquitecturas por el número de conjuntos de instrucción y conjuntos de datos que pueden procesar simultáneamente.

La taxonomía que aquí se presenta está basada en una visión funcional de la arquitectura y en el flujo de información entre las unidades.

Esta taxonomía es una estructura de dos niveles que subsume la clasificación de Flynn. Las discriminaciones de grueso calibre pueden realizarse en el nivel superior, en tanto que las distinciones más finas se hacen en el segundo nivel.

Entre otras clasificaciones que hay en la literatura, es la clasificación de Feng (1972), quizás la más conocida. Es una clasificación orientada a performance que describe el paralelismo de un conjunto de procesadores en una máquina en términos de la cantidad de bits que pueden procesarse simultáneamente.

Las máquinas se describen por un par ordenado, el primer elemento indica el tamaño de la palabra, y el segundo la profundidad (la cantidad de palabras que pueden operar simultáneamente). Esto permite comparaciones de performance entre una amplia variedad de arquitecturas, pero, justamente, como relaciona arquitecturas diversas no hace resaltar sus diferencias.

Otra clasificación debida a Reddi y Feurstel (1976) utiliza la organización física, el flujo de información, y la representación y transformación de la información como base para la clasificación. El flujo de la información puede ser una herramienta muy poderosa y general para describir arquitecturas, pero sus otros atributos dependen en demasía del tipo de implementación concreta.

Otra clasificación muy popular es la de Händler (1977), que describe las arquitecturas mediante la cantidad de procesadores y la forma en que pueden ser pipelinizados, y el tamaño y profundidad de las unidades aritmético-lógicas.

En tanto que estas clasificaciones son buenas para las arquitecturas vectorizadas convencionales, no aportan claridad respecto de generalizar sobre las nuevas arquitecturas de multiprocesadores.

10.2. - Razones para un modelo arquitectural

Existen tres razones para clasificar arquitecturas. La primera es comprender el punto que se ha alcanzado actualmente. Hasta las pasadas dos décadas, casi todos los sistemas de computadoras usaron la arquitectura Von Neumann. Incluso cuando el hardware subyacente comenzó a tener cierta capacidad limitada de paralelismo (como el caso del CDC6600), este se ocultó generalmente al usuario. Sin embargo, a partir de entonces, el crecimiento en sistemas con diferentes clases de paralelismo fue explosivo, y no está del todo claro cuáles de estas arquitecturas tienen las mejores proyecciones hacia el futuro.

La segunda razón para tener una clasificación de arquitecturas es el hecho de que la misma puede revelar configuraciones posibles que podrían no habérsele ocurrido jamás a un diseñador. Una vez que los sistemas existentes se hubieran clasificado, las lagunas en la clasificación pueden sugerir otras posibilidades. Por supuesto que no todas estas posibilidades pueden implicar mejoras, bien pudieron haber sido probadas y descartadas por peores.

Sin embargo, hasta que el espacio de búsqueda haya sido claramente delimitado, no puede estarse seguro de que todas las posibilidades hayan sido examinadas.

La tercera razón para un esquema de clasificación es el hecho de que ella permite que se construyan y utilicen modelos más útiles de performance. Un buen esquema de clasificación podría revelar porqué una arquitectura en particular es mejor para proveer una determinada mejora de performance. Puede servir también como modelo para análisis de performance.

10.3. **Intento de clasificación**

Si consideramos la completa cuestión del cómputo y de las máquinas que lo proveen, parece útil pensar en ciertos niveles de abstracción.

En el más alto, el nivel abstracto, podemos considerar el modelo de cómputo que se está utilizando. Muchas computadoras aún emplean el modelo de cómputo denominado el modelo de Von Neumann.

Desde este punto de vista del cómputo, las operaciones primitivas consisten en las operaciones usuales de la matemática: suma, resta, multiplicación, comparación, y así siguiendo (en dominios convenientemente restringidos).

Los datos primitivos, sin embargo, permanecen en las denominadas posiciones (locations), y el comportamiento correcto del cómputo se cumplimenta con el programador que describe el orden exacto en el cual los cálculos deben realizarse.

De hecho, el orden de ejecución está superrestringido, en el sentido de que existen otras secuencias de ordenamiento triviales que calculan el mismo resultado, pero que no existe forma de expresarlas en el modelo de cómputo.

El modelo de cómputo de Von Neumann parece natural a aquellos cuya experiencia de programación fue desarrollada utilizando lenguajes que soportan este modelo. Sin embargo, simplemente no contiene ningún concepto de cómputo más que el de hacer una cosa a la vez (paralelismo), ni tampoco alguna forma mediante la cual sea posible expresar un ordenamiento del cómputo dinámicamente.

No es de sorprender que los primeros nuevos modelos de cómputo desarrollados fueran extensiones del modelo de Von Neumann, a los cuales se agregaran conceptos tales como la comunicación de primitivas. Luego se desarrollaron otros modelos más inusuales (tales como dataflow y reducción de grafos).

A medida que nos acercamos a un nivel más detallado, podemos considerar las **máquinas abstractas** como la implementación de los modelos de cómputo.

El mapeo desde el modelo de cómputo hacia la máquina abstracta no es necesariamente una relación uno-a-uno; algunos modelos de cómputo pueden implementarse en más de un tipo de máquina abstracta, aunque alguna de ellas se adapte mejor que las otras.

Una máquina abstracta captura la esencia de la forma de una arquitectura en particular, sin hacer distinción entre las diferentes tecnologías, tamaño de la implementación, o consideraciones semejantes.

Por el momento, la máquina abstracta tradicional de Von Neumann puede resumirse como un contador de programa, modos de direccionamiento, códigos de condición, una unidad de control, y una unidad aritmético y lógica.

Esta máquina abstracta no hace referencia a las diferentes implementaciones Von Neumann que existen, ni diferencia entre RISC y CISC. Este es sin embargo un nivel muy útil en el cual considerar las arquitecturas, y es el nivel que forma la clasificación más alta en esta taxonomía.

En el siguiente nivel, más detallado, consideramos las implementaciones de máquina. Esto no solo concierne a la tecnología utilizada para implementar la máquina. Más bien, este nivel representa una definición de la arquitectura del computador, la ilustración (o el dibujo) que se muestra al programador de lenguaje assembler. Este nivel corresponde al segundo de la taxonomía.

No tratamos aquí el papel que juegan los lenguajes de programación en esta clasificación, ya que, en cierto sentido el lenguaje utilizado en una máquina corresponde al modelo de cómputo que ella posee. Si no es este el caso, entonces existe un bastante mal ajuste entre el lenguaje y la implementación, y ya que nuestro objetivo es la performance, podemos ignorar esta clase de situaciones.

Por supuesto que cuando todas las máquinas que había eran Von Neumann, se utilizaron muchos diferentes lenguajes en la misma máquina. Sin embargo, ha existido una tendencia creciente hacia la construcción de la arquitectura y el lenguaje en forma conjunta, y existen muy buenas razones para tal tendencia. Aún en el caso de la arquitectura Von Neumann surge claramente que los lenguajes de tipo Algol se adaptan mejor que por ejemplo el Lisp.

10.4. - **Funciones de la arquitectura del computador**

En esta clasificación de las arquitecturas de los computadores utilizamos cuatro tipos de unidades funcionales a partir de las cuales se puede construir una máquina abstracta. Ellas son :

- Un procesador de instrucciones; una unidad funcional que actúa como un intérprete de las instrucciones, cuando tales elementos existen explícitamente en el modelo de cómputo.
- Un procesador de datos; una unidad funcional que actúa como un transformador de datos, normalmente de la forma que se corresponda a las operaciones aritméticas básicas.
- Una jerarquía de memoria; un dispositivo de almacenamiento inteligente que transmite datos de y hacia los procesadores.
- Un switch; un dispositivo abstracto que provee conectividad entre las otras unidades funcionales.

Veremos a continuación cómo estas unidades funcionales representan el comportamiento de una máquina abstracta utilizando la arquitectura de Von Neumann como un ejemplo.

10.5. - **LA MAQUINA ABSTRACTA VON NEUMANN**

La máquina abstracta Von Neumann consta de un solo procesador de instrucción (PI), un sólo procesador de datos (PD), y dos memorias jerárquicas. La unidad funcional switch (SW) no juega ningún papel en la máquina abstracta Von Neumann.

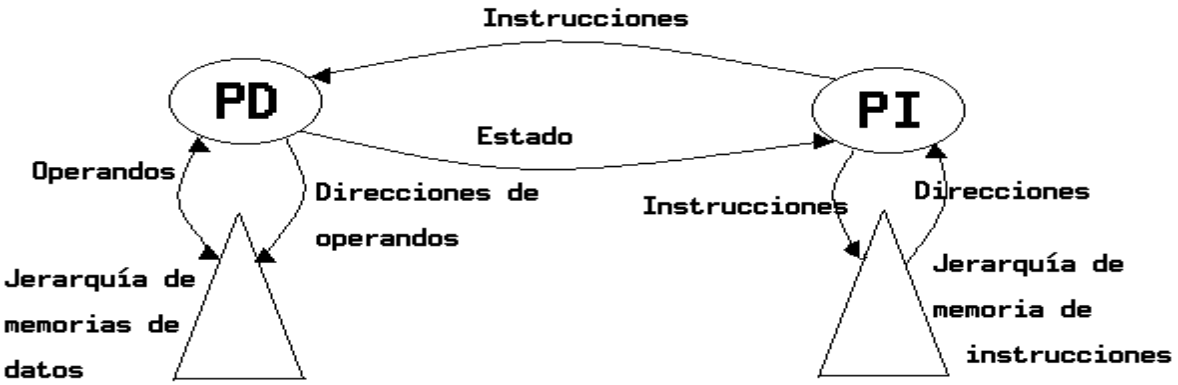


Fig. 10.1. - Disposición de las unidades funcionales de la máquina abstracta Von Neumann.

Estas unidades funcionales están dispuestas como muestra la Fig. 10.1. El procesador de instrucción está conectado a una de las jerarquías de memoria de donde toma las instrucciones. Para hacer esto él debe proveer e la memoria de un label o dirección que identifique la instrucción que él desea acceder.

Está conectado también al procesador de datos, al cual entrega información acerca de las operaciones a realizar y las direcciones de los valores sobre los cuales la operación debe realizarse, y recibe de éste la información de estado (códigos de condición).

Las funciones del procesador de instrucción son :

(1) Determinar la dirección de la próxima instrucción a ejecutar, utilizando la información de su propio almacenamiento interno y la información de estado que le fuera enviada desde el procesador de datos.

(2) Dar la instrucción a la memoria jerárquica para que esta le provea la instrucción con tal dirección.

(3) Recibir la instrucción y decodificarla. Esto implica determinar cuál operación, de existir, deberá ser requerida al procesador de datos para "ejecutar" la instrucción.

(4) Informar al procesador de datos sobre la operación requerida.

(5) Determinar las direcciones de los operandos y pasarlas al procesador de datos.

(6) Recibir la información de estado desde el procesador de datos (luego de completada la operación).

Cualquier arquitectura con un procesador de instrucciones requerirá una serie de pasos como los descriptos arriba.

No existe ningún supuesto en esta descripción sobre el hecho de que estos pasos se realicen en este orden o que alguno de ellos pueda saltearse.

En el primer nivel de la taxonomía se especifican solamente la existencia de ciertas unidades funcionales conectadas de una cierta forma.

En la máquina Von Neumann, el orden de las operaciones es normalmente el indicado antes. Este se denomina la secuencia de ejecución de la instrucción.

Si deseamos indicar precisamente el orden de los pasos debemos pasar al siguiente nivel de la taxonomía y representar estas diferentes operaciones como estados en un diagrama de estados.

En este segundo nivel el procesador de instrucciones puede representarse como se ve en la Fig. 10.2, en la cual cada estado representa una operación y las flechas indican las dependencias entre los estados. Las flechas

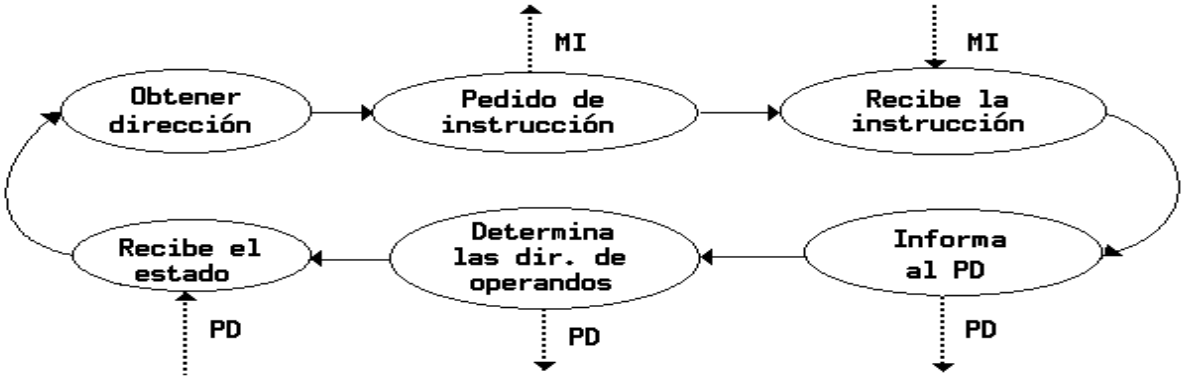


Fig. 10.2. - La estructura interna de un procesador de instrucción Von Neumann.

punteadas representan la comunicación entre el procesador de instrucción y las otras unidades funcionales del sistema.

Si podemos colocar un token para un determinado estado, entonces este diagrama de estados (conjuntamente con las ubicaciones de los tokens) da una completa descripción sobre qué está haciendo el procesador de instrucción en un momento dado. El token va siguiendo las flechas completando el circuito para cada instrucción ejecutada.

Podemos seguir haciendo diferenciaciones de acuerdo al diagrama de estados presentado indicando detalles a nivel de la implementación. Esto podría incluir mecanismos o controles temporales para cada estado, o la forma en que las entidades lógicas se mapean en entidades físicas.

Un ejemplo de esto es el siguiente: Muchos de los computadores Von Neumann implementan la selección de la próxima instrucción utilizando un contador de programa que se actualiza con la longitud de la instrucción actual (ignorando las instrucciones de bifurcación). Sin embargo, en algunos sistemas la dirección de la próxima instrucción está incluida en la instrucción en sí misma. Ninguna de estas estrategias es más Von Neumann que la otra; son elecciones de implementación y son indistintas a nivel de la máquina abstracta.

El procesador de datos lleva a cabo los siguientes pasos :

(1) Recibe un tipo de instrucción desde el procesador de instrucción.
(2) Recibe las direcciones de operandos desde el procesador de instrucción.
(3) Da las instrucciones a la memoria jerárquica (separada de la del procesador de instrucción) para que ésta le provea los valores de los operandos.
(4) Recibe los valores de los operandos desde la memoria jerárquica.
(5) Realiza la operación requerida (la fase de ejecución).
(6) Devuelve información de estado al procesador de instrucción.
(7) Provee un valor resultado a la memoria jerárquica.

Una vez más, esto puede representarse mediante un diagrama de estados consistente en un ciclo de estados.

La mayor diferencia con el procesador de instrucción es que la fase de ejecución del procesador de datos consiste potencialmente de una gran cantidad de estados paralelos que representan aquellas operaciones que el procesador de instrucción ve como una primitiva. El diagrama de estado se ve en la Fig. 10.3.

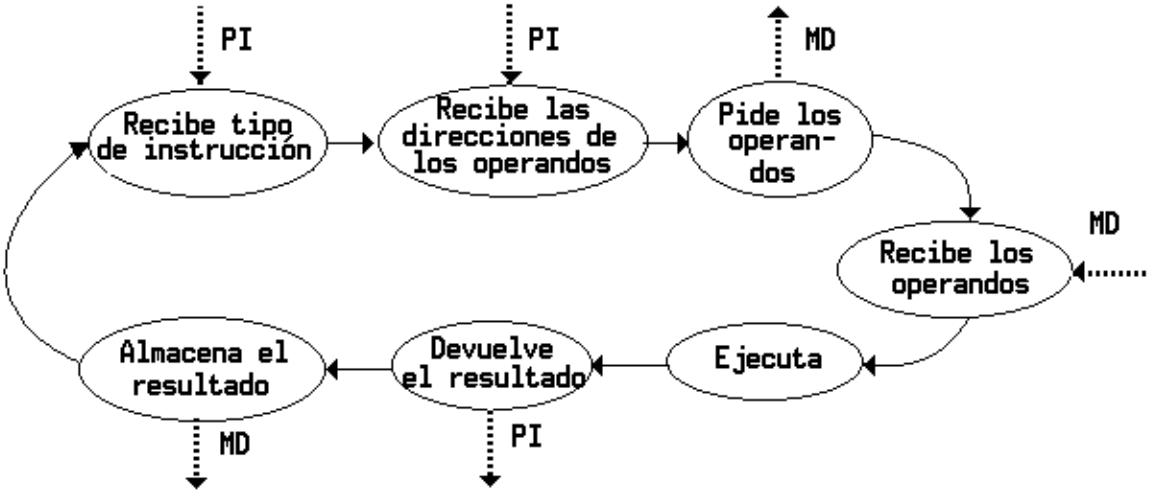


Fig. 10.3. - La estructura interna del procesador de datos de Von Neumann.

Ciertas acciones que se realizan en ambos procesadores deben ser sincrónicas. Los lugares donde esto ocurre se grafican utilizando líneas punteadas en el diagrama de estados, y representan la comunicación entre los procesadores.

Su comportamiento es el siguiente : debe existir un token al comienzo y en la finalización del camino de comunicación. Ningún token puede abandonar el estado hasta que la comunicación se haya completado (es decir, la comunicación es sincrónica).

La memoria jerárquica es un dispositivo inteligente que intenta proveer los datos que le requieren los procesadores unidos a ella, lo más rápido posible.

El sistema de almacenamiento puede ser visto como una pirámide, o más exactamente como un zigurat, ya que los datos almacenados son discretos y las capas sucesivas se amplían en etapas.

Una memoria jerárquica trata de tener la próxima porción de datos que le será requerida por el procesador que se comunica con el tope de la jerarquía donde el tiempo de acceso es el menor. Una jerarquía perfecta de memoria alcanzaría siempre este objetivo. Sin embargo, no existe una forma óptima de determinar qué porción de datos será la próxima requerida. De allí, que todas las jerarquías de memoria deben aproximar tal objetivo mediante heurística.

Lógicamente, el promedio de tiempo de acceso se minimiza cuando los datos más referenciados permanecen en la jerarquía más alta y aquellos referenciados menos a menudo permanecen en un nivel inferior.

Cuando los datos accedidos tienen algún tipo de localidad, ya sea temporal o espacial, la performance del tiempo promedio de acceso puede mejorarse manteniendo los datos que hayan sido recientemente accedidos o los que se encuentren cerca de lo accedido en la jerarquía más alta.

Podemos ver ahora porqué una máquina abstracta Von Neumann tiene dos jerarquías de memorias. Los datos en la memoria de instrucción fluyen hacia el procesador; las instrucciones son consumidas por el procesador de instrucciones y no se modifican a sí mismas. Por otra parte, la jerarquía de memoria de datos debe manejar el movimiento de los datos en ambas direcciones. De allí que la implementación de las dos jerarquías de memoria debe tener propiedades substancialmente diferentes.

Normalmente las jerarquías de memoria se implementan utilizando una sola jerarquía de almacenamiento excepto en el tope (la cache) en el cual se diferencian los datos e instrucciones. La cache está en el tope de la jerarquía de memorias, por encima de la memoria principal, la cual se continúa a su vez en el almacenamiento en discos y en términos de mayor capacidad hacia los dispositivos en cinta. Las E/S se producen en el nivel más bajo de la jerarquía de memorias. En un cierto sentido, la E/S es un acceso a un mecanismo de almacenamiento de gran capacidad, el mundo externo.

10.6. - **FORMAS DE INCREMENTAR PERFORMANCE.**

Muchos de los otros modelos de cómputo fueron motivados por el deseo de mejorar la performance, por tanto es instructivo en este punto considerar cómo la máquina abstracta Von Neumann puede rendir una mejor performance.

Existen tres grandes formas :

- reordenar el diagrama de estados a fin de que el token pueda circular en el menor tiempo;
- permitiendo que más de un token, y por tanto más de un paso activo, existan simultáneamente, y
- duplicando unidades funcionales para permitir la actividad concurrente.

Todas estas tres variantes incrementan la performance y en particular la tercera es muy fructífera y ha llevado a una amplia variedad de clases de arquitecturas.

El primer método para incrementar la velocidad (reordenando el diagrama de estados), no genera realmente una nueva clase de arquitectura, aún así el esquema de clasificación permite distinguir ciertas diferencias en la descripción de los estados del procesador.

Como un ejemplo, en el procesador de datos las operaciones de informar al procesador de instrucciones sobre el estado de la información derivada del resultado y el almacenamiento del resultado en las memorias jerárquicas son independientes. Sin embargo, pueden ocurrir simultáneamente. Esto nos lleva al diagrama revisado que puede verse en la Fig. 10.4, el cual nos muestra, desde el punto de vista del tiempo de cada etapa, que el nuevo ciclo será más veloz.

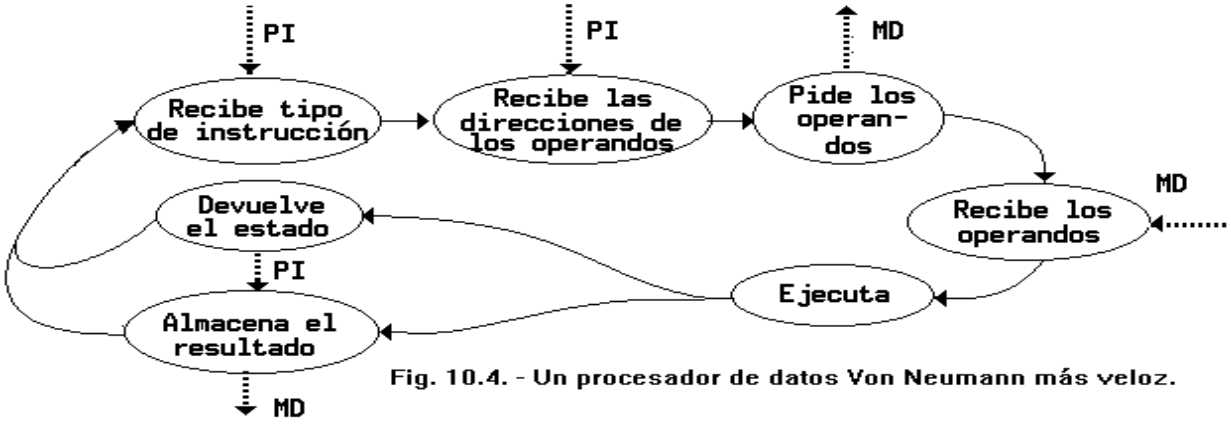


Fig. 10.4. - Un procesador de datos Von Neumann más veloz.

10.6.1. - **Pipelining**

La segunda posibilidad de incrementar velocidad permite que existan más de un token en el diagrama de estados, y se denomina Pipelining.

Supongamos que el número de estadios en el diagrama de estados es n. Si cada estadio demora la misma cantidad de tiempo, por ejemplo t, el tiempo de ejecutar una instrucción será n * t. Si permitimos que n tokens estén presentes simultáneamente, estamos representando n diferentes instrucciones en varios estadios de ejecución.

El tiempo que lleva ejecutar una instrucción sigue siendo n. Sin embargo, el tiempo promedio en el cual las instrucciones se completan es de 1/t instrucciones por unidad de tiempo.

Luego, hemos alcanzado un mecanismo acelerador de n-partes en nuestra máquina, de manera tal que tengamos siempre n tokens presentes en el diagrama de estados.

Por supuesto que existen dificultades prácticas para realizar esto mismo (instrucciones tales como bifurcaciones y saltos, interrupciones externas, y necesidades simultáneas de acceder a los mismos datos) que provocan que la performance real del pipeline sea mucho peor de lo que nuestro análisis sugiere. Sin embargo, sirve para

demostrar porqué el pipelining fue de interés a los diseñadores de sistemas que intentaban mejorar la performance de las máquinas Von Neumann.

Los pipelines tienen otra interesante propiedad. Si subdividimos cada uno de los estados en subestados, de manera que cada uno lleve un tiempo de t/2, y permitimos que existan 2n tokens en el diagrama de estados, entonces una instrucción se completa cada t/2 unidades de tiempo, incrementando el promedio de completitud de instrucciones a 2 / t instrucciones por unidad de tiempo. Luego, tenemos una aceleración total de 2n.

Haciendo más pequeño cada estadio nos permite incrementar aún más el promedio de finalización, pero a expensas de complicar el manejo de interacciones imprevistas.

Dentro de nuestro modelo de máquina abstracta el comportamiento pipeline puede describirse sin agregar ninguna unidad funcional nueva. En cambio, rotulamos cada unidad funcional con uno de dos tipo : Simple, o Pipelined.

10.6.2 - **Procesadores Array**

Replicar unidades funcionales es la tercera forma de incrementar performance. Existen muchas formas de replicar, y comenzamos por la más sencilla : los procesadores Array.

Un procesador Array típico consta de una unidad de instrucción que envía instrucciones a un conjunto de procesadores esclavos. Cada procesador esclavo ejecuta la instrucción que le fuera enviada interpretando las direcciones de los operandos como si fueran de su propia memoria.

Normalmente, se proveen instrucciones para permitir el intercambio de datos entre los procesadores en forma directa o indirecta.

Es común también que cada procesador tenga acceso a sus propios datos, permitiendo que diferentes direcciones sean accedidas por diferentes procesadores.

Un procesador array está directamente clasificado como una máquina abstracta con procesador de única instrucción, una sola memoria de instrucciones, múltiples procesadores de datos y múltiples memorias de datos.

Debido a que existen múltiples unidades funcionales debemos describir las formas en que están conectadas. Las conexiones entre las unidades funcionales se hacen mediante conmutadores (switches) abstractos, que pueden implementarse de diferentes forma : Buses, Switches dinámicos, Redes de interconexión estáticas.

Existen 4 formas de conectar unidades funcionales con switches abstractos, a saber :

*) **1-a-1**, una sola unidad funcional de un tipo está conectada a otra sola unidad funcional de otro tipo. Por supuesto que existen a nivel físico más de una conexión y la información fluye en ambas direcciones, como en la máquina Von Neumann descripta anteriormente. Esto no es realmente un switch, pero lo incluimos por completitud.

*) **n-a-n**, en esta configuración la i-ésima unidad de un conjunto de unidades funcionales está conectada con la i-ésima unidad de otro. Este tipo de switch es una conexión 1-a-1 que se repite n veces.

*) **1-a-n**, en esta configuración una unidad funcional está conectada a los n dispositivos de otro conjunto de unidades funcionales.

*) **n-por-n**, en esta configuración cada dispositivo de un conjunto de unidades funcionales puede comunicarse con cualquier dispositivo de un segundo conjunto de unidades y viceversa.

Todo procesador array tiene un switch de conexión de tipo 1-a-n que conecta el único procesador de instrucción a los procesadores de datos.

Se distinguen dos diferentes subfamilias, basándose en el tipo de switch utilizado para conectar los procesadores de datos a la jerarquía de memoria de datos.

La primera clase puede verse en la Fig. 10.5. Aquí la conexión del procesador-de-datos-memoria-de-datos

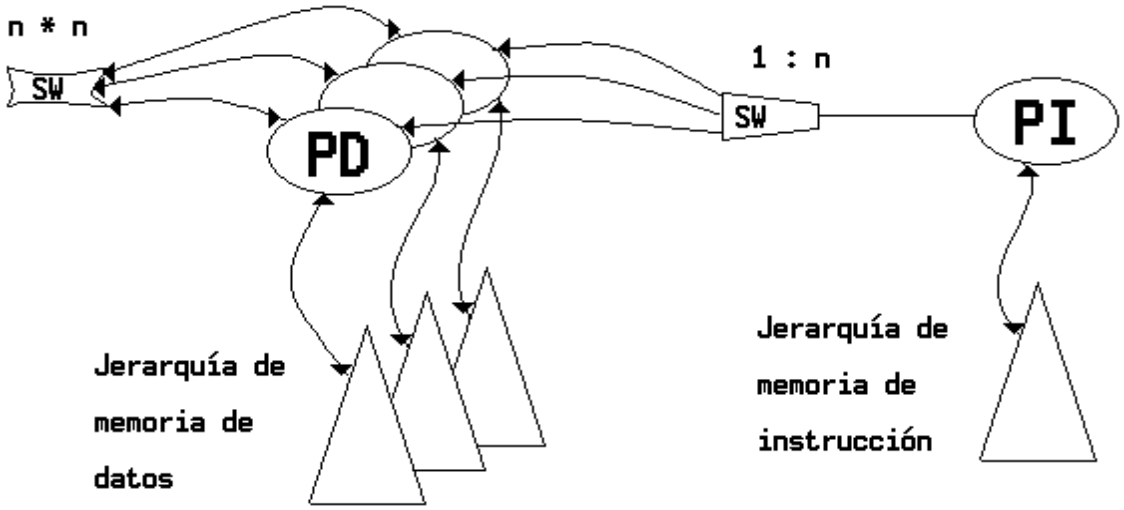


Fig. 10.5. - Un procesador array de Tipo 1.

es del tipo n-a-n y la conexión del procesador-de-datos-procesador-de-datos es del tipo n-por-n.

Este esquema de conexión se utilizó en máquinas tales como la Máquina de Conexión (The Connection Machine de Hillis, 1985).

El segundo tipo se ve en la Fig. 10.6, en este caso, el procesador-de-datos-memoria-de-datos están conectados con una conexión de tipo n-por-n y no existe conexión entre los procesadores de datos. Esta forma de switch es usualmente denominada una red de sincronización (alignement network) Un ejemplo de ésta lo constituye el Procesador Científico Burroughs (BSP).

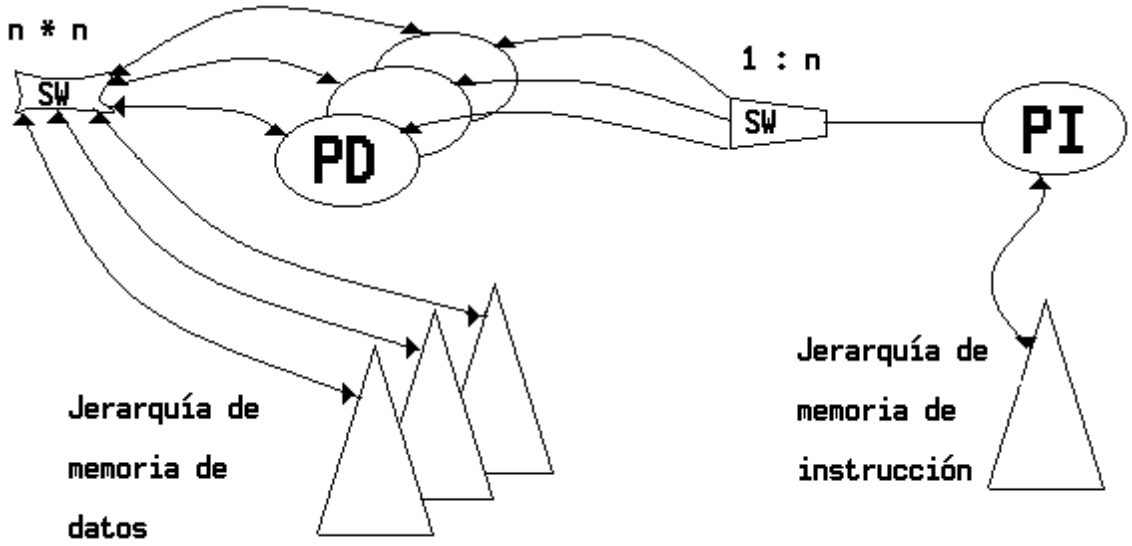


Fig. 10.6. - Un procesador array de Tipo 2.

Podemos introducir ahora nuestra taxonomía informalmente.

Clasificamos arquitecturas por el número de procesadores de instrucción que tienen, y por el número de procesadores de datos; por la cantidad de jerarquías de memorias que existen y por la forma en que estas unidades funcionales están conectadas por switches abstractos. Luego, un monoprocesador Von Neumann puede clasificarse como :

- La cantidad de procesadores de instrucción es uno.
- La cantidad de memorias de instrucción es uno.
- El switch entre el procesador de instrucciones y la memoria de instrucciones es de tipo 1-a-1.
- La cantidad de procesadores de datos es uno.
- La cantidad de memorias de datos es uno.
- El switch entre el procesador de datos y la memoria de datos es de tipo 1-a-1.
- El switch entre el procesador de datos y el procesador de instrucciones es del tipo 1-a-1.

Un procesador array de tipo 1 puede clasificarse como :

- La cantidad de procesadores de instrucción es uno.
- La cantidad de jerarquías de memoria de instrucción es uno.
- El switch entre el procesador de instrucciones y la memoria de instrucciones es de tipo 1-a-1.
- La cantidad de procesadores de datos es n.
- La cantidad de memorias de datos es n.
- El switch entre los procesadores de datos y las memorias de datos es de tipo n-a-n.
- El switch entre el procesador de instrucciones y los procesadores de datos es del tipo 1-a-n.
- El switch entre los procesadores de datos es del tipo n-por-n.

Veremos descripciones más complejas en la próxima sección. A medida que la cantidad de unidades funcionales crece, las posibilidades de interconexión crecen conjuntamente.

10.7. - **MAQUINAS PARALELAS VON NEUMANN**

Los procesadores arrays dependen de la existencia de que muchos datos puedan ser manejadas de la misma forma (igual instrucción) y al mismo tiempo. Muchos problemas no se encuadran adecuadamente en este paradigma.

Aún así, es natural considerar la replicación del procesador de instrucciones así como la del procesador de datos y crear múltiples líneas de control.

Esta es la idea que se plantea en las máquinas procesadores paralelas de Von Neumann.

De este supuesto surgen dos diferentes tipos de arquitectura : Sistemas fuertemente acoplados, y Sistemas débilmente acoplados.

Los sistemas fuertemente acoplados consisten en un conjunto de procesadores conectados a un conjunto de memorias mediante un switch dinámico. Cualquier procesador puede acceder cualquier posición de memoria con aproximadamente la misma latencia. La comunicación y sincronización entre los procesos se realiza mediante el uso de variables compartidas. Ejemplo de tales máquinas son la BBN Butterfly, la IBM 3081 y 3090 y la C.mmp.

Los sistemas débilmente acoplados consisten en un conjunto de procesadores, cada uno con su memoria local. La comunicación se produce por pedidos explícitos desde un procesador hacia otro a través de una red de interconexión. A través de intercambio de mensajes o por medio de llamadas a procedimientos remotos. Ejemplos de estas lo constituyen las máquinas hipercubo de Intel y el NCubo, los sistemas basados en el Transputer tales como el SuperNode (Proyecto Esprit 1985) y el Meiko MK40, así como viejos sistemas tales como el CM*.

En los sistemas fuertemente acoplados, tanto los procesadores de datos como los de instrucciones están duplicados, pero los procesadores de datos comparten una memoria común.

La correspondiente máquina abstracta puede verse en la Fig. 10.7. La mostramos aquí con múltiples memorias de datos y un switch de n-por-n entre los procesadores de datos y las memorias de datos debido a que es ligeramente más sugestiva que la implementación usual. Funcionalmente hablando ésta es indistinguible de una memoria común compartida.

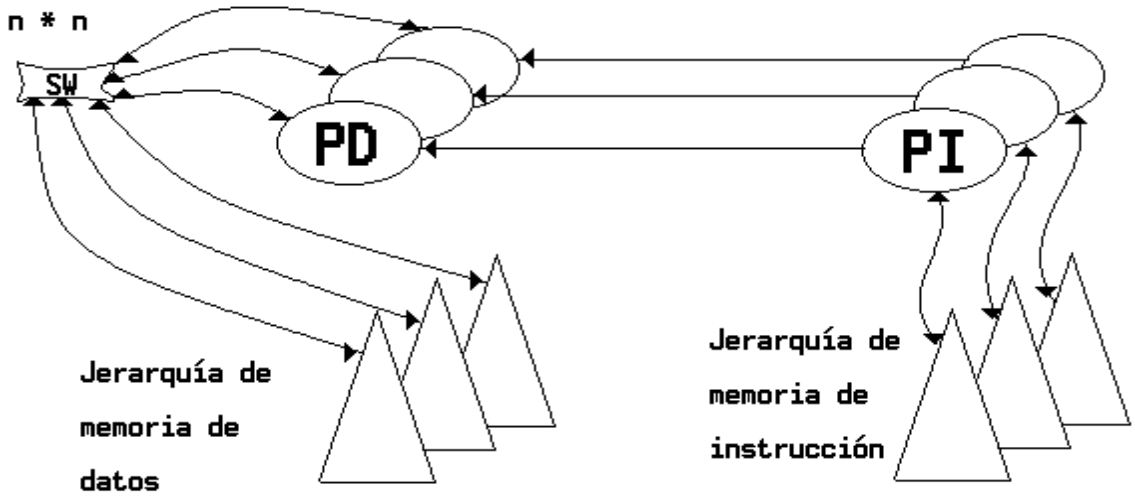


Fig. 10.7. - La máquina abstracta para un típico multiprocesador fuertemente acoplado.

Los sistemas débilmente acoplados también tienen duplicación de los procesadores de datos e instrucciones, pero los switches en el subsistema de datos difieren. Un débilmente acoplado típico puede verse en la Fig. 10.8. La conexión entre los procesadores de datos y las memorias de datos es de tipo n-a-n, y la conexión entre los procesadores de datos es del tipo n-por-n.

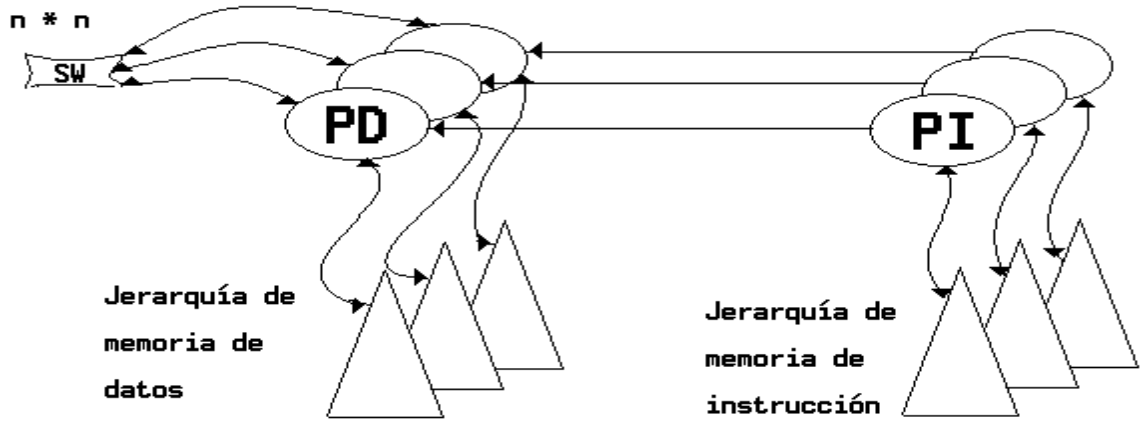


Fig. 10.8. - La máquina abstracta para un típico multiprocesador débilmente acoplado.

10.8. - MAQUINAS QUE UTILIZAN OTROS MODELOS DE COMPUTO

Las máquinas Von Neumann están todas basadas en el modelo de cómputo con líneas de instrucciones ejecutadas secuencialmente, excepto cuando el orden se altere explícitamente. Hablamos ya de que este ordenamiento es superrestrictivo. Esta dificultad empeora cuando se utilizan modelos de cómputo con múltiples líneas de control, ya que el programador debe considerar no solo el ordenamiento de las instrucciones en un conjunto en particular sino también las posibles secuencias de instrucciones en diferentes conjuntos interactuantes.

No sorprende entonces que aquellos involucrados en el diseño de las máquinas paralelas examinaran otros modelos de cómputo que no tuvieran esta tosca propiedad.

Estos nuevos modelos de cómputo están caracterizados por una completa ausencia de la descripción del programador en cuanto al orden de evaluación, que no sea el implícito por las dependencias de los datos. Esto permite que cualquier orden posible de evaluación para la ejecución pueda ser considerado, y el que otorgue la mejor performance pueda ser seleccionado por el compilador o el environment de runtime.

Los modelos de cómputo con esta propiedad se expresan usualmente en lenguajes funcionales.

10.8.1. - **Máquinas de reducción de grafos Graph Reduction Machines**

En reducción de grafos un cómputo se codifica como un grafo de funciones y argumentos que (recursivamente) tienen la propiedad de que el nodo raíz es un Aplicar, el subárbol izquierdo es la descripción de una función, y el subárbol derecho es la descripción de un argumento.

La descripción de una función o un argumento puede ser tanto un valor o una descripción de como obtener el valor.

Si ambos, la función y el argumento, han sido ya evaluados, entonces el subárbol (denominado Redex) se encuentra disponible para ejecución.

Cuando éste ha sido evaluado, sus valores reemplazan al subárbol. Si aún no ha sido evaluado, entonces algún subárbol debe ser Redex o el cómputo no puede proseguir.

Luego, en general las redex disponibles para ejecución se encuentran en las hojas del árbol. Típicamente existen múltiples redex disponibles para ser evaluadas en cualquier momento, por tanto pueden involucrarse múltiples procesadores en la evaluación simultáneamente.

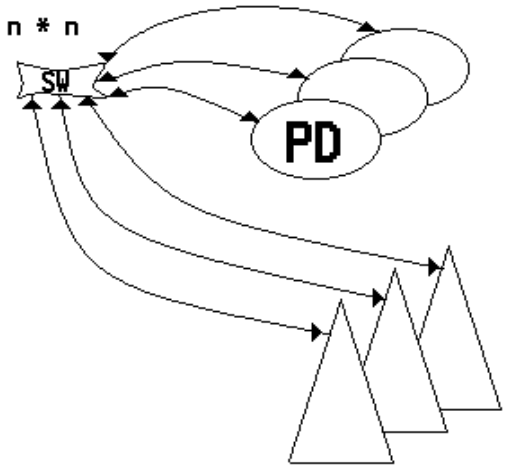


Fig. 10.9. - La máquina abstracta para reducción de grafos.

La máquina abstracta para reducción de grafos puede verse en la Fig. 10.9. Difiere primordialmente de la máquina abstracta de Von Neumann en la ausencia de una unidad de instrucciones y una memoria de instrucciones. El grafo que está siendo reducido juega aquí ambos roles, el de instrucciones (en su estructura) y el de datos (en sus contenidos).

El procesador de datos y sus memorias de datos asociadas son más parecidas a aquellos de un multiprocesador fuertemente acoplado, lo cual no debe sorprender ya que el grafo es una amplia estructura de datos compartidos.

Ya que no existe un procesador de instrucciones que provee las direcciones de los datos al procesador de datos, éste debe generarlas él solo.

Existen dos formas equivalentes de lograr esto.

El primero consiste en proveer al procesador de datos de un selector que seleccione cualquier redex disponible desde la memoria de datos.

El otro, es obtener una memoria de datos que actúe como un dispositivo activo que empuje los redex disponibles hacia el procesador de datos.

Los trabajos en reducción de grafos han tenido extremada importancia en Europa y el Reino Unido y han sido bastante exitosos. Varias máquinas han sido diseñadas y construidas, algunos ejemplos son Alice (1981) y Flagship (1988).

10.8.2. - **Máquinas Dataflow**

Las máquinas dataflow son otra clase de máquinas que no cuentan con un mecanismo secuenciador de instrucciones sino que está implícito por las dependencias de los datos.

Su modelo de cómputo es un grafo con arcos dirigidos, a lo largo de los cuales fluyen los datos y los vértices representan las operaciones que transforman los datos.

Ciertos arcos tienen vértice sólo de un lado; estos representan los inputs o outputs del cómputo.

Un vértice, u operador, se dispara de acuerdo a alguna regla de encendido (firing rule) especificada por la forma particular de dataflow.

En general, la detonación consume un grupo de datos de cada arco de input y produce un resultado que parte luego a través de los arcos de salida.

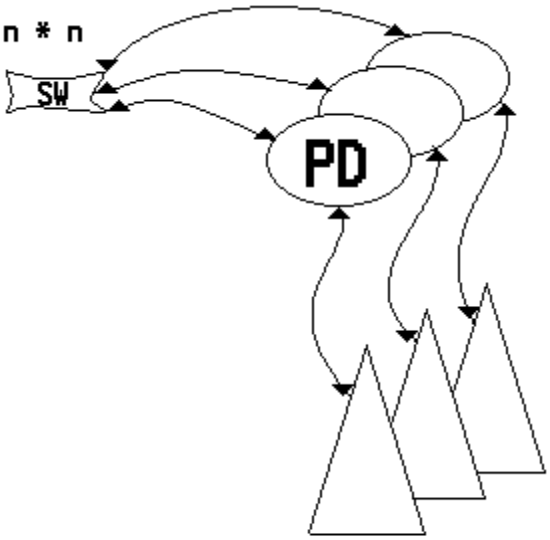


Fig. 10.10. - La máquina abstracta para un Dataflow (de tipo Dennis).

Muchas de las máquinas de dataflow están basadas en un anillo que consiste de un almacenamiento comparativo (donde los valores de los datos esperan hasta que el conjunto de todos los operandos estén presentes), una memoria que contiene los operadores y una serie de elementos de procesamiento que ejecutan los operadores, produciendo los valores del resultado que fluyen alrededor del anillo hacia el almacenamiento comparativo.

La máquina abstracta para un procesador dataflow puede tener dos formas, que se corresponden a las dos formas en que pueden disponerse los procesadores de datos en un procesador array.

En la primera forma, todas las memorias de datos son visibles a todos los procesadores. Esta visión es la que se aplicó en la máquina Dataflow Manchester (1980) y en la máquina de Arvind.

Esta forma de procesamiento es funcionalmente idéntica a las máquinas paralelas de reducción de grafos y, por tanto, tienen el mismo diagrama de máquina abstracta (Fig. 10.9).

En la segunda forma, que puede verse en la Fig.10.10, cada procesador tiene su propia memoria local, y los datos requeridos por otro procesador fluyen a través de switches entre-procesadores-de-datos.

La segunda forma es quizás más intuitiva; en el límite cada procesador ejecuta sólo un único operador.

10.9. **EL MODELO FORMAL**

Nuestro esquema de clasificación consiste de dos niveles de detalle y discriminación. El primer nivel define una arquitectura especificando :

- cantidad de procesadores de instrucción (PI)
- cantidad de memorias de instrucción (MI)
- el tipo de switch de conexión de los PIs a las MIs
- la cantidad de procesadores de datos (PD)
- la cantidad de memorias de datos (MD)
- el tipo de switch de conexión de los PDs y las MDs.
- el tipo de switch de conexión entre los PIs y los PDs
- el tipo de switch de conexión entre los PDs.

El primer nivel hace un refinamiento de la clasificación de Flynn expandiendo las clases de SIMD y MIMD a un subconjunto de subclases que capturan importantes arquitecturas existentes.

El segundo nivel permite refinamientos ulteriores describiendo si se puede o no pipelinizar el procesador y hasta qué punto, y dando el comportamiento del diagrama de estado de los procesadores. Es posible un tercer nivel que describa detalles de implementación. Esta clasificación se muestra en la Fig. 10.11.

Como hemos visto esta clasificación captura las diferencias entre arquitecturas que son significativamente diferentes, ignorando diferencias que son una cuestión primordial en la implementación.

La Tabla de la Fig. 10.12 muestra un conjunto sencillo de arquitecturas posibles basándonos en el supuesto de que la cantidad de memorias coincide con la cantidad de procesadores para cada subsistemas.

A pesar de que existen arquitecturas interesantes en las cuales este supuesto no se cumple, por ahora, en aras de la claridad, las ignoraremos.

Las clases 1 a 5 son las máquinas dataflow/reducción que no tienen instrucciones en el sentido usual de la palabra. La clase 6 es el monoprocesador de Von Neumann. Las clases 7 a 10 son procesadores array.

Nótese que las clases 5 a 10 representan extensiones de las arquitecturas a las que pertenecen de forma tal que cuentan con una doble conectividad para sus procesadores de datos.

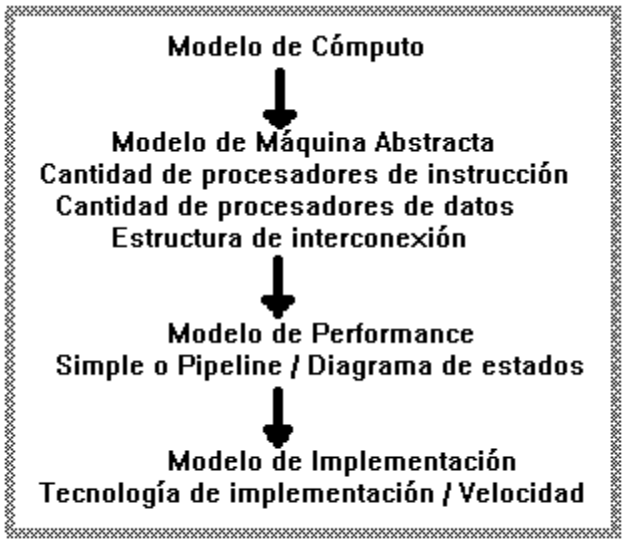


Fig. 10.11. - El método de clasificación

Las clases 11 y 12 son las arquitecturas MISD. A pesar de que estas clases han sido tratadas como una aberración, existen lenguajes para los cuales este tipo de ejecución es apropiada. Por ejemplo, en NIAL (Nested Interactive Array Language, 1986) se puede escribir :

[f g h] x

que es la aplicación paralela de las funciones f, g, y h sobre x.

No estamos enterados de ninguna arquitectura existente de este tipo, sin embargo el concepto no es totalmente ridículo.

Las clases 13 a 28 son los multiprocesadores de varios tipos. Las clases 13 a 20 son más o menos los multiprocesadores convencionales con diferentes formas de la estructura de conexión.

Las clases 21 a 28 son arquitecturas más exóticas en las cuales las conexiones entre los procesadores de instrucciones es del tipo n-por-n. Estas clases se encuentran completamente inexploradas.

Las arquitecturas que tienen una conexión n-por-n entre los procesadores de instrucciones y las memorias de instrucciones son aquellas en donde la decisión del procesador de ejecutar una instrucción particular se realiza tarde (justo antes de la ejecución).

Las arquitecturas Dataflow y de Reducción de grafos tienen a menudo esta propiedad, pero la única máquina Von Neumann de la que tengamos conocimiento que se comporta de esta forma es el Procesador de Elementos Heterogéneos (1981) de la compañía Denelcor.

Algunas máquinas Von Neumann que existen cuentan con más de un sólo procesador de datos. Por ejemplo, algunos sistemas pipeline de alta performance (Cray I, Cyber 205) tienen un procesador de datos para instrucciones vectoriales y otro para instrucciones escalares. Algunos tienen inclusive un procesador escalar separado para las instrucciones de este tipo.

Las arquitecturas de este tipo no concuerdan con el esquema sencillo presentado antes, pero pueden representarse fácilmente incluyendo las unidades de procesamiento vectorial conjuntamente con los procesadores de datos (la cantidad de procesadores de datos es 1 + m).

Fig. 10.12. - TABLA DE ARQUITECTURAS POSIBLES

Clase	PIs	PDs	PI-PD	PI-MI	PD-MD	PD-PD	Nombre
1	0	1	ninguna	Ninguna	1-1	ninguna	Monoprocesador reducción/dataflow
2	0	n	ninguna	Ninguna	n-n	ninguna	Máquinas separadas
3	0	n	ninguna	Ninguna	n-n	n*n	Débilmente acoplado reducción/dataflow
4	0	n	ninguna	Ninguna	n-n	ninguna	Fuertemente acoplado reducción/dataflow
5	0	n	ninguna	Ninguna	n*n	n*n	
6	1	1	1-1	1-1	1-1	ninguna	Monoprocesador Von Neumann
7	1	n	1-n	1-1	n-n	ninguna	
8	1	n	1-n	1-1	n-n	n*n	Array processor Tipo 1
9	1	n	1-n	1-1	n*n	ninguna	Array processor Tipo 2
10	1	n	1-n	1-1	n*n	n*n	
11	n	1	1-n	n-n	1-1	ninguna	
12	n	1	1-n	n*n	1-1	ninguna	
13	n	n	n-n	n-n	n-n	ninguna	Monoprocesadores separados Von Neumann
14	n	n	n-n	n-n	n-n	n*n	Von Neumann débilmente acoplados
15	n	n	n-n	n-n	n*n	ninguna	Von Neumann fuertemente acoplados
16	n	n	n-n	n-n	n*n	n*n	
17	n	n	n-n	n*n	n-n	ninguna	
18	n	n	n-n	n*n	n-n	n*n	
19	n	n	n-n	n*n	n*n	ninguna	Procesador de elementos Heterogéneos de Denelcor
20	n	n	n-n	n*n	n*n	n*n	
21	n	n	n*n	n-n	n-n	ninguna	
22	n	n	n*n	n-n	n-n	n*n	
23	n	n	n*n	n-n	n*n	ninguna	
24	n	n	n*n	n-n	n*n	n*n	
25	n	n	n*n	n*n	n-n	ninguna	
26	n	n	n*n	n*n	n-n	n*n	
27	n	n	n*n	n*n	n*n	ninguna	
28	n	N	n*n	n*n	n*n	n*n	

10.10. - **OTRA PROPUESTA TAXONOMICA**.

La clasificación anterior adolece de ciertos defectos que fueron observados por Subrata Dasgupta (1990) quien propuso a su vez un esquema taxonómico más completo utilizando parte de las características definidas por Skillicorn.

10.10.1. - **TAXONOMIA JERARQUICA**.

Una clasificación de tipo jerárquica provee las bases para comparar y discriminar distintos objetos. Esta clasificación permite además determinar puntos de convergencia a través de los niveles de categorías, según se puede apreciar en el diagrama de la Fig. 10.13.

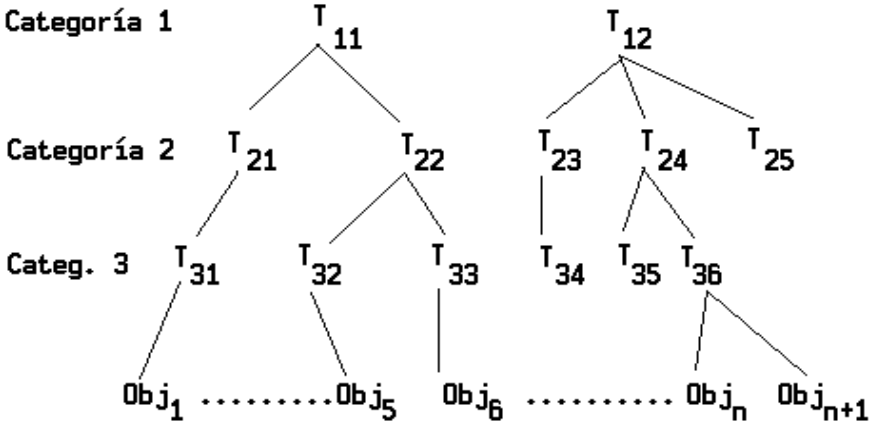


Fig. 10.13.

10.10.2. - **Limitaciones de la taxonomía de Skillicorn**.

Vamos a denominar **TC** (Caracteres Taxonómicos) a los PI, PD, etc.

Las limitaciones de la taxonomía de Skillicorn están dadas por :

Predecir Capacidad (Poder) : De acuerdo a la clasificación de Skillicorn, no es posible determinar, de antemano, cuando una arquitectura es más poderosa que otra, o similar, sin comparar los valores de sus respectivos TC. Esto se debe a que esta clasificación no es jerárquica y entonces tiene una sola categoría.

Explicar Capacidad (Poder): Si bien, dando las características precisas de los PI y PD, existe la posibilidad de explicar las capacidades de las arquitecturas, esto no se realiza completamente.

La clasificación no incluye el concepto de Pipelining en el mismo nivel de abstracción que lo hace con las memorias, procesadores y switches. Sólo aparece en el diagrama de estados. Pero, los diagramas de estados no identifican formalmente taxonomías y no constituyen categorías.

Además, como la clasificación no es de tipo jerárquica, existen más elementos en el nivel superior que en el inferior, ver Fig. 10.14.

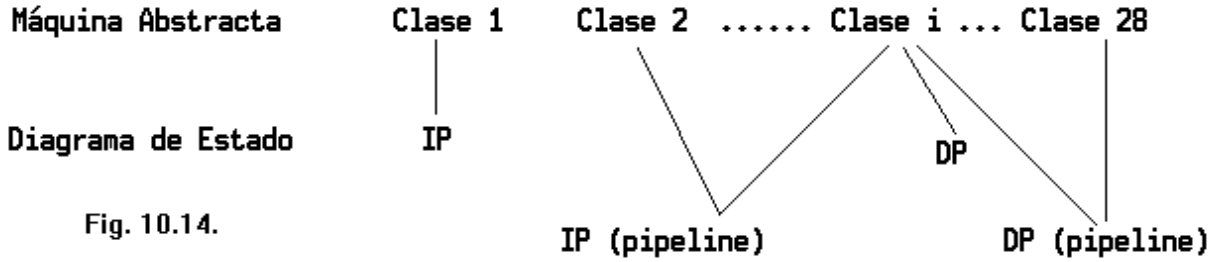


Fig. 10.14.

En cuanto a las memorias (MI, MD), se dice que son jerárquicas, pero no se discrimina entre memoria principal, cache, etc. y no se tiene en cuenta su importancia en el multiprocesamiento.

10.10.3. - **Nueva Taxonomía**.

Se toma como punto de partida el esquema de Skillicorn, pero teniendo en cuenta sus limitaciones se llegará a un resultado distinto.

La nueva taxonomía se construye con siete (7) **primitivas** de endoarquitectura que llamaremos **átomos**.

Atomos del mismo tipo pueden ser combinados en entidades más complejas que llamaremos **radicales atómicos**, que a su vez pueden ser combinados en conceptos más complicados llamados **radicales no-atómicos**.

Los radicales no-atómicos pueden ser combinados en **moléculas**, las que denotarán entidades de endoarquitectura completas.

10.10.3.1. - **Atomos**.

Usaremos los siguientes símbolos:

iM :	memoria interleaving
sM :	memoria simple
C :	cache
sI :	unidad de instrucciones simple
pI :	unidad de instrucciones pipeline
sX :	unidad de ejecución simple
pX :	unidad de ejecución pipeline

Las letras (M, I, C, X) son **iones**.

Las letras minúsculas (s, i, p) son **prefijos** de los iones.

Las funciones de los iones son las conocidas, pero aquí entenderemos que las del ión I son :

- Determinar próxima instrucción.
- Cargar instrucción de M o C.
- Decodificar instrucción.
- Calcular dirección de operandos.
- Transferir direcciones de operandos a X.
- Recibir estados de X.

Y las funciones del ión X son :

- Recibir direcciones de operandos y operadores (instrucciones) de I.
- Cargar operandos desde M o C.
- Ejecutar instrucciones.
- Almacenar resultados en M o C.
- Informar estado a I.

De lo que resulta que las funciones de I y X son equivalentes a las de los PI y PD de Skillicorn.

10.10.3.2. - **Radicales Atómicos**.

Son las instancias replicadas de un átomo, lo que se escribirá como A_n donde el subíndice n, que llamaremos el número de radical (**RN**), puede ser una constante o variable pero siempre dentro del rango de los enteros positivos.

Por ejemplo:

*) iM_2 denota dos átomos de memoria interleaving
*) sM_4 denota 4 átomos de memoria
*) pI_m denota m instancias de procesadores. de instrucciones con pipeline

Si RN = 1 se dice que es un radical monoatómico y es equivalente escribir C o C_1.
si RM >> 1 se dice que es un radical multiatómico.

10.10.3.3. - **Radicales No-atómicos**.

Es la combinación de átomos distintos. Por convención se escribirá siempre más a la izquierda los correspondientes a memoria (M), luego los de cache (C), y por último los correspondientes a procesador (P).

Por ejemplo CP indica un procesador con cache, que es una combinación de un radical C con un I, o un X.

También aquí puede haber replicaciones, y obtenerse ejemplos como este :

$iM.(C.sI_2)_k$

que indica una memoria de instrucciones combinada con k instancias de dos procesadores de instrucciones con cache. En este caso el radical es de tipo (MCP-).

10.10.3.4. - **Moléculas**.

Una molécula I- es un radical MCP- que representa un subsistema completo de "preparación de instrucciones" dentro del nivel de endoarquitectura.

Una molécula X- es un radical MCP que representa un subsistema completo de "ejecución de instrucciones" dentro del nivel de endoarquitectura.

Una macromolécula es una combinación de moléculas I- o X- que representan una computadora completa a nivel de endoarquitectura. Por convención se escribirá I a la izquierda de X.

10.10.3.5. - **Sintaxis**.

M' ::= $iM \mid sM \mid iM_n \mid sM_n$
C' ::= $C \mid C_n$
I' ::= $sI \mid pI \mid sI_n \mid pI_n$
X' ::= $sX \mid pX \mid sX_n \mid pX_n$
CP' ::= $C'.I' \mid C'.X' \mid C'.CP' \mid (CP)'_n$
MCP' ::= $M'.I' \mid M'.X' \mid M'.MCP' \mid (MCP)'_n$
I'' ::= MCP'
X'' ::= MCP'
MM'' ::= $(I'')\ (X'') \mid (\ (I'')(X'')\ )_n$

Atomos : iM, sM, C, sI, pI, sX, pX.
Instancias : subíndice n
$(.)_n$: instancias de radicales o moléculas (replicación)
.' : radical
." : moléculas

10.10.4. - **<u>Estructuras de Radicales y Moléculas</u>**.

Si decimos que R es un radical, podemos escribir cabeza(R) y cola(R). Por ejemplo si R1 = C.sI, entonces cabeza (R1) = C y cola (R1) = sI.

Si R2 = $(C.pI)_n$, resulta cabeza(R2) = cola(R2) = $(C.pI)_n$.

Si R3 = $iM_m.(C.pI)_n$, resulta cabeza(R3) = iM_m y cola(R3) = $(C.pI)_n$.

La estructura de un radical (y obviamente de una molécula I o X) puede ser determinada por dos operadores **Rep** y **Link** que se definen así:

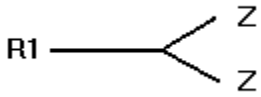
Rep(R) causa
R
.
.
R

Link(R1,R2) causa las siguientes estructuras:

- si cola(R1) y cabeza(R2) tienen una sola instancia, entonces

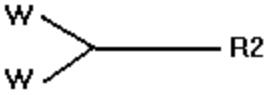
o sea un camino dedicado

- si cola(R1) tiene una sola instancia pero cabeza(R2) tiene varias (en particular tomemos que sean 2), entonces



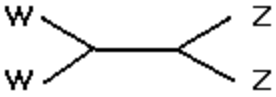
caminos compartidos

- si cola(R1) tiene varias instancias (w) y cabeza(R2) solo una, entonces



caminos alternativos

- si cola(R1) tiene varias instancias (w) y cabeza(R2) también (z), entonces



caminos compartidos entre todas las instancias

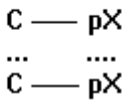
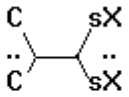
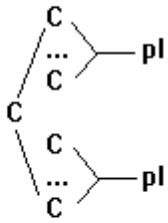
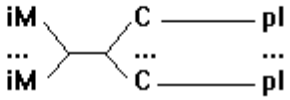
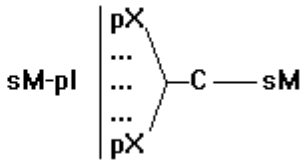
Usando Rep y Link, la estructura de cualquier macromolécula puede ser determinada de acuerdo a las reglas 1) a 9) que se muestran a continuación.

Nota: St(x) significa estructura de x y $St^{-1}(x)$ es la imagen espejada de la estructura de x.

1) St(M')	=	if iM	then iM
		if sM	then sM
		if iM_n	then Rep(iM)
		if sM_n	then Rep(sM)
2) St(C')	=	if C	then C
		if C_n	then Rep(C)
3) St(I')	=	if sI	then sI
		if pI	then pI
		if sI_n	then Rep(sI)
		if pI_n	then Rep(pI)
4) St(X')	=	if sX	then sX
		if pX	then pX
		if sX_n	then Rep(sX)
		if pX_n	then Rep(pX)
5) St(CP')	=	if C'.I'	then Link(St(C'),St(I'))
		if C'.X'	then Link(St(C'),St(X'))
		if C'.CP'	then Link(St(C'),St(CP'))
		if $(CP')_n$	then Rep(St(CP'))
6) St(MCP')	=	if M'.I'	then Link(St(M'),St(I'))
		if M'.X'	then Link(St(M'),St(X'))
		if M'.CP'	then Link(St(M'),St(CP'))

		if M'.MCP'	then Link(St(M'),St(MCP'))
		if $(MCP')_n$	then Rep(St(MCP'))
7) St(I'')	=	St(MCP')	
8) St(X'')	=	St(MCP')	
9) St(MM'')	=	if (I'')(X'')	then St(I'') | St^{-1}(X'')
		if (I'')$(X'')_n$	then Rep(St(I'') | St^{-1}(X''))

Para comprender mejor lo anterior y ver que los switches de Skillicorn están implícitos en las fórmulas veamos algunos ejemplos:

Fórmula	Estructura
C.sI	
[C.pX]n	
Cm.sXn	
C.[C2.pI]2	
iMm.[C.pI]n	
[sM.pI][sM.C.pX4]	

10.10.5. - **Categorías Jerárquicas del sistema taxonómico**.

Según este esquema tendremos 3 principales jerarquías.
1)- La categoría MCP, es la molecular y la más baja.
2)- La categoría CP, la siguiente.
3)- La categoría P, es la más alta, correspondiendo a los procesadores.
Veamos como ejemplo (Fig. 10.15) el caso de la VAX 8600, Vax 11/780 y VAX 11/750.

Para asignar un lugar a una computadora en esta clasificación, es necesario establecer primero su fórmula molecular correspondiente a su endoarquitectura.

Esta se obtiene en términos de memoria, cache, procesadores (X e I), si hay interleaving de memoria o no, si los procesadores son de tipo pipeline y si la cache retiene datos, instrucciones o ambos.

Es necesario conocer la cantidad de estos elementos y sus interconexiones.

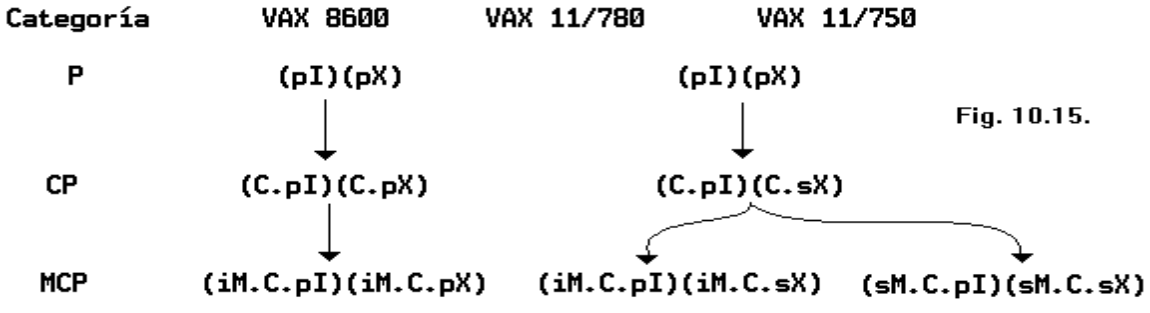


Fig. 10.15.

Toda esta información puede ser obtenida en forma directa de la misma fórmula. Tomemos como ejemplo el caso de la VAX 8600, cuya fórmula es:

(iM.C.pI)(iM.C.pX)

De aquí se infiere que es monoprocesador, que ejecuta en pipeline el ciclo completo de sus instrucciones, que el flujo del pipe se ve ayudado por memoria cache y memoria interleaving (repetidas ambas).

La categoría CP es una abstracción de la MCP (molecular) e identifica a las computadoras sin tener en cuenta los componentes de memoria.

La categoría P es una abstracción de la CP y clasifica procesadores en función de cómo ejecutan instrucciones, independiente de la presencia de caches.

10.10.5.1. - **Algunos ejemplos**.

Illiac IV	$(sM_{64}.sI)(sM.sX)_{64}$
Cray X-MP	$(iM_m.(C.pI)_n)(iM_m.(C_r.pX_s)_q)$
IBM 3838	$(sM.pI)(sM.pX_7)$
Manchester	$(sM.sI)(sM.sX_n)$

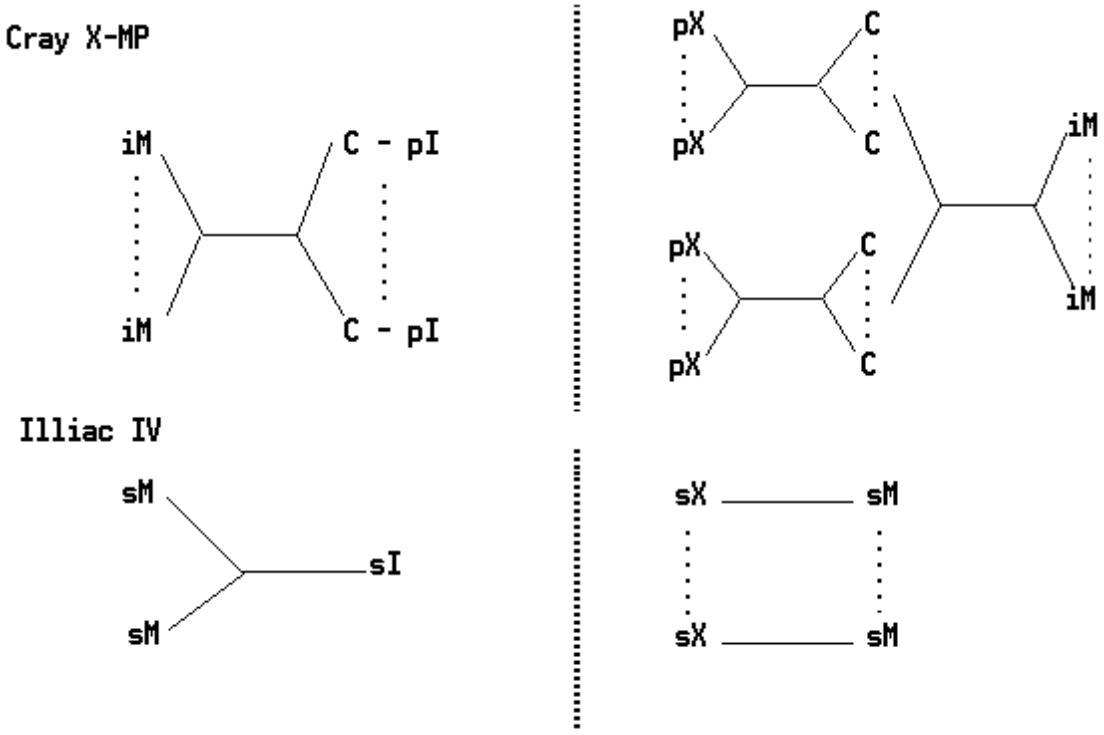


Fig. 10.16. - Gráficos de algunos ejemplos.

En el caso de la Manchester es discutible la presencia de sI, pero si se elimina, no queda clara la búsqueda de instrucciones, o sea que en esta clasificación las máquinas DF (Dataflow) tampoco quedan bien caracterizadas.

EJERCICIOS

1) Cuál es el problema específico de la clasificación de Flynn que llevar a plantear un nuevo método de clasificación de las arquitecturas de computadores.
2) Qué es la máquina abstracta y cuál es su relación con el modelo de cómputo ?
3) Cuáles son los tipos de unidades funcionales que se emplean en esta clasificación y cuál es su función ?
4) Cuáles son las funciones del Procesador de Instrucción en la máquina abstracta Von Neumann ?
5) Qué es un diagrama de estados y cuando se hace necesario su uso ?
6) Cuáles son las funciones del Procesador de Datos en la máquina abstracta de Von Neumann ?
7) Qué significa que la comunicación entre el PI y el PD debe ser sincrónica ? Justifique.
8) Cómo se conceptualiza en esta clasificación la característica de las jerarquías de memorias ?
9) Cuáles son las formas de incrementar performance ? Explique cada una.
10) Cuáles son las formas de conectar unidades funcionales ?
11) Explique las diferencias entre las dos clases posibles de procesadores array.
12) Cuál es la diferencia conceptual de la máquina abstracta de un sistema computador fuertemente acoplado con la de un sistema débilmente acoplado ?
13) Porqué una arquitectura de reducción de grafos carece de PI y de MI ? Justifique.

14) Cómo se proveen las direcciones de los datos en un sistema de reducción de grafos ?
15) Relacione el "anillo comparativo" de la máquina dataflow en este capítulo con lo visto en el capítulo 9.
16) Porqué se pueden tener dos modelos de máquinas abstractas para la arquitectura Dataflow ? Justifique.
17) Formalmente, qué se especifica en el 1er nivel de esta clasificación ?
18) Idem 17) para el 2do y 3er nivel.
19) Cómo se pueden encuadrar en esta clasificación las arquitecturas con procesadores vectoriales ? Justifique.