Capítulo 4

Otras arquitecturas avanzadas

Hasta ahora, la mayoría de sistemas que se han explicado pretendían resolver problemas generales de procesamiento, es decir, se trataba de arquitecturas que de alguna u otra manera permitían resolver de una forma eficiente una amplia gama de problemas de computación. Esto es especialmente interesante puesto que estas arquitecturas son las que se utilizan en los sistemas de propósito general que son los que ocupan la mayor parte del mercado de computadores. Hay diferencias entre unas arquitecturas y otras, por ejemplo es evidente que una máquina vectorial va a ser ideal para el cálculo científico, mientras que un multicomputador es más interesante para la realización de tareas sencillas por muchos usuarios conectados a el, pero en ambos casos el sistema es fácil de programar y resuelve prácticamente el mismo tipo de problemas.

Existen, sin embargo, otras arquitecturas que para un caso particular de problemas funcionan mejor que ninguna otra, pero no resuelven bien el problema de una programación sencilla o el que se pueda utilizar en cualquier problema. Estas arquitecturas son por tanto específicas para determinados problemas y no es fácil encontrarlas en sistemas de propósito general. Sin embargo, son necesarias para su utilización en sistemas *empotrados*, es decir, sistemas que realizan una tarea específica, que requieran una especial capacidad en una determinada tarea que un sistema de propósito general no puede tener. Los procesadores vectoriales y matriciales son ejemplos de arquitecturas que estarían en la frontera entre el propósito general y los sistemas empotrados. Ya como sistemas empotrados, o formando parte como un módulo en un sistema de propósito general, tenemos arquitecturas como los arrays sistólicos, los DSPs, las redes de neuronas, los procesadores difusos, etc.

Como en la mayoría de los casos estas soluciones requieren la realización de un chip a medida, a estas arquitecturas específicas se les llama también *arquitecturas VLSI*. La mejora y abaratamiento del diseño de circuitos integrados ha permitido que se pudieran implementar todas estas arquitecturas específicas en chips simples, lo que ha permitido el desarrollo y uso de estas arquitecturas.

Hemos visto entonces que hay problemas específicos que inspiran determinadas arquitecturas como acabamos de ver. Por otro lado, existen diferentes formas de enfocar la computación incluso de problemas generales, pero que por su todavía no probada eficiencia, o porque simplemente no han conseguido subirse al carro de los sistemas comerciales, no se utilizan en la práctica. Una arquitectura que estaría dentro de este ámbito sería la arquitectura de flujo de datos, donde el control del flujo del programa la realizan los datos y no las instrucciones. Hay otras arquitecturas como las basadas

en *transputers* que resultan también interesantes y de hecho se utilizan comercialmente, aunque en realidad no suponen una nueva filosofía de arquitectura sino más bien una realización práctica de conceptos conocidos.

En este capítulo veremos estas arquitecturas cuyo conocimiento va a resultar interesante para poder resolver problemas específicos que se pueden presentar y que quizá una arquitectura de propósito general no resuelve de forma eficaz. Se empezará por la máquina de flujo de datos y se seguirá con las arquitecturas VLSI.

4.1 Máquinas de flujo de datos

Hay dos formas de procesar la información, una es mediante la ejecución en serie de una lista de instrucciones y la otra es la ejecución de las instrucciones según las pidan los datos disponibles. La primera forma empezó con la arquitectura de Von Neumann donde un programa almacenaba las órdenes a ejecutar, sucesivas modificaciones, etc., han convertido esta sencilla arquitectura en los multiprocesadores para permitir paralelismo.

La segunda forma de ver el procesamiento de datos quizá es algo menos directa, pero desde el punto de vista de la paralelización resulta mucho más interesante puesto que las instrucciones se ejecutan en el momento tienen los datos necesarios para ello, y naturalmente se deberían poder ejecutar todas las instrucciones demandadas en un mismo tiempo. Hay algunos lenguajes que se adaptan a este tipo de arquitectura comandada por datos como son el Prolog, el ADA, etc. es decir, lenguajes que explotan de una u otra manera la concurrencia de instrucciones.

En una arquitectura de flujo de datos un instrucción está lista para ejecución cuando los datos que necesita están disponibles. La disponibilidad de los datos se consigue por la canalización de los resultados de las instrucciones ejecutadas con anterioridad a los operandos de las instrucciones que esperan. Esta canalización forma un flujo de datos que van disparando las instrucciones a ejecutar. Por esto se evita la ejecución de instrucciones basada en *contador de programa* que es la base de la arquitectura Von Neumann.

Las instrucciones en un flujo de datos son puramente autocontenidas; es decir, no utilizan variables en una memoria compartida global, sino que llevan los valores de las variables en ellas mismas. En una maquina de este tipo, la ejecución de una instrucción no afecta a otras que estén listas para su ejecución. De esta manera, varias instrucciones pueden ser ejecutadas simultáneamente lo que lleva a la posibilidad de un alto grado de concurrencia y paralelización.

4.1.1 Grafo de flujo de datos

Para mostrar el comportamiento de las máquinas de flujo de datos se utiliza un grafo llamado *grafo de flujo de datos*. Este grafo de flujo de datos muestra las dependencias de datos entre las instrucciones. El grafo representa los pasos de ejecución de un programa y sirve como interfaz entre la arquitectura del sistema y el lenguaje de programación.

Los nodos en el grafo de flujo de datos, también llamados *actores*, representan los operadores y están interconectados mediante arcos de entrada y salida que llevan etiquetas conteniendo algún valor. Estas etiquetas se ponen y quitan de los arcos de acuerdo con unas reglas de disparo. Cada actor necesita que unos determinados arcos

de entrada tengan etiquetas para que pueda ser disparado (o ejecutado). Cuando están presentes las etiquetas en los arcos necesarios, el actor se habilita y entonces se dispara. Al dispararse, quita una etiqueta de cada uno de los arcos de entrada requeridos para el disparo, realiza la función requerida según las etiquetas que hubiera, y pone las etiquetas de resultado en los arcos de salida correspondientes.

Cada nodo de un grafo de flujo de datos puede representarse (o guardarse) como una *copia de actividad*. Una copia de actividad consiste en unos campos para el tipo de operación, el almacenamiento de las etiquetas de entrada, y para las direcciones de destino. Como en un grafo de flujo de datos, una colección de copias de actividad se puede utilizar para representar un programa.

Hay dos tipos de etiquetas: etiquetas de datos y etiquetas booleanas. Para distinguir el tipo de entradas a un actor se utilizan flechas simples para llevar etiquetas de datos y flechas dobles para las etiquetas booleanas. El *actor de conmutación* pone la etiqueta de entrada en uno de los arcos de salida dependiendo de la etiqueta que llega por la entrada booleana. El *actor de combinación* pone una de las etiquetas de entrada en el arco de salida dependiendo de la etiqueta booleana. La *puerta T* pasa la etiqueta de la entrada a la salida cuando la etiqueta booleana es *True*, y la *puerta F* cuando es *False*.

La figura 4.1 muestra el grafo de flujo de datos para el cálculo de $N!$ cuyo código secuencial se da a continuación. Hay que hacer notar que los arcos booleanos de entrada de los dos actores que combinan están inicializados con etiquetas *False*. Esto causa que en el inicio del programa, la etiqueta de datos N vaya a la salida de los dos combinadores.

```
Input N;
  i=N;
  While i>1
  {   N=N(i-1);
      i=i-1;
  }
Output N;
```

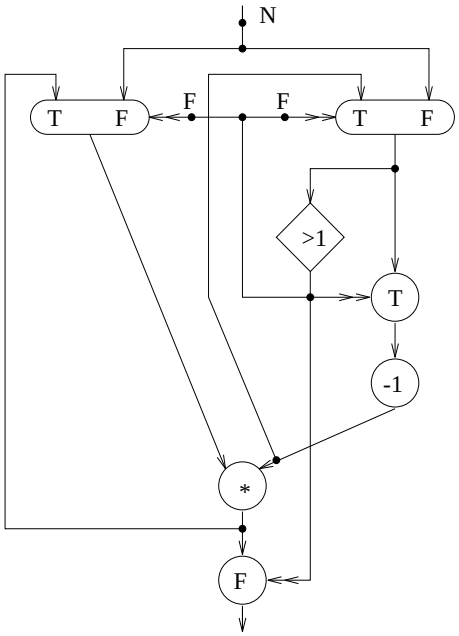

Figura 4.1: Grafo de flujo de datos para calcular $N!$

4.1.2 Estructura básica de un computador de flujo de datos

Se han realizado algunas máquinas para probar el funcionamiento de la arquitectura de flujo de datos. Una de ellas fue desarrollada en el MIT y su estructura se muestra en la figura 4.2. El computador MIT está formado por cinco unidades:

Unidad de proceso, formada por un conjunto de elementos de proceso especializados.

Unidad de memoria, formada por células de instrucción para guardar la instrucción y sus operandos, es decir, cada célula guarda una copia de actividad.

Red de arbitraje, que envía instrucciones a la unidad de procesamiento para su ejecución.

Red de distribución, que transfiere los resultados de las operaciones a la memoria.

Unidad de control, que administra todas las unidades.

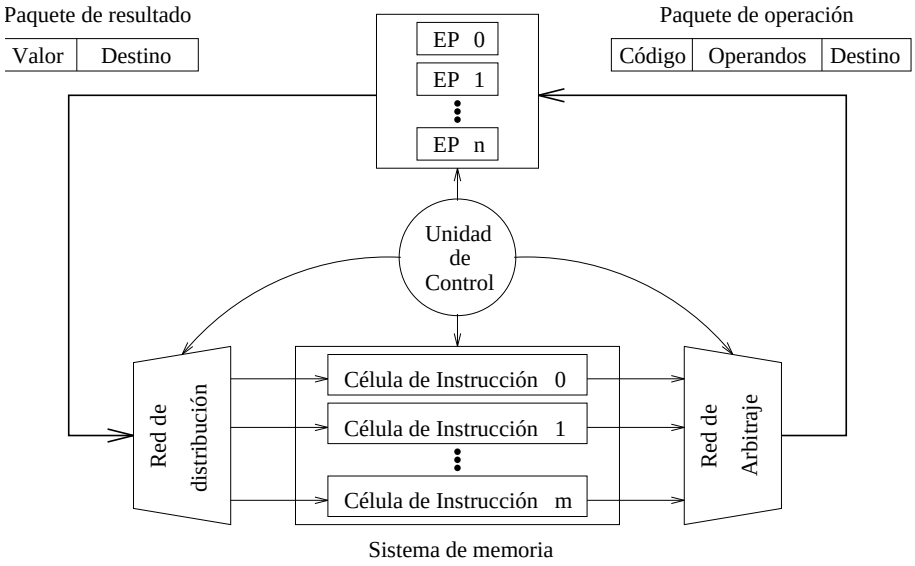

Figura 4.2: La máquina MIT de flujo de datos.

Cada célula de instrucción guarda una instrucción que consiste en un código de operación, unos operandos, y una dirección de destino. La instrucción se activa cuando se reciben todos los operandos y señales de control requeridas. La red de arbitraje manda la instrucción activa como un paquete de operación a unos de los elementos de proceso. Una vez que la instrucción es ejecutada, el resultado se devuelve a través de la red de distribución al destino en memoria. Cada resultado se envía como un paquete que consiste en un resultado mas una dirección de destino.

Las máquinas de flujo de datos se pueden clasificar en dos grupos:

Máquinas estáticas: En una máquina de flujo de datos estática, una instrucción se activa siempre que se reciban todos los operandos requeridos y haya alguna instrucción esperando recibir el resultado de esta instrucción; si no es así, la instrucción permanece inactiva. Es decir, cada arco en el grafo del flujo de datos puede tener una etiqueta como mucho en cualquier instante. Un ejemplo de este tipo de flujo de datos se muestra en la figura 4.3. La instrucción de multiplicación no debe ser activada hasta que su resultado previo ha sido utilizado por la suma.

A menudo, esta restricción se controla con el uso de señales de reconocimiento.

Máquinas dinámicas: En una máquina de flujo de datos dinámica, una instrucción se activa cuando se reciben todos los operandos requeridos. En este caso, varios conjuntos de operandos pueden estar listos para una instrucción al mismo tiempo. En otras palabras, un arco puede contener más de una etiqueta.

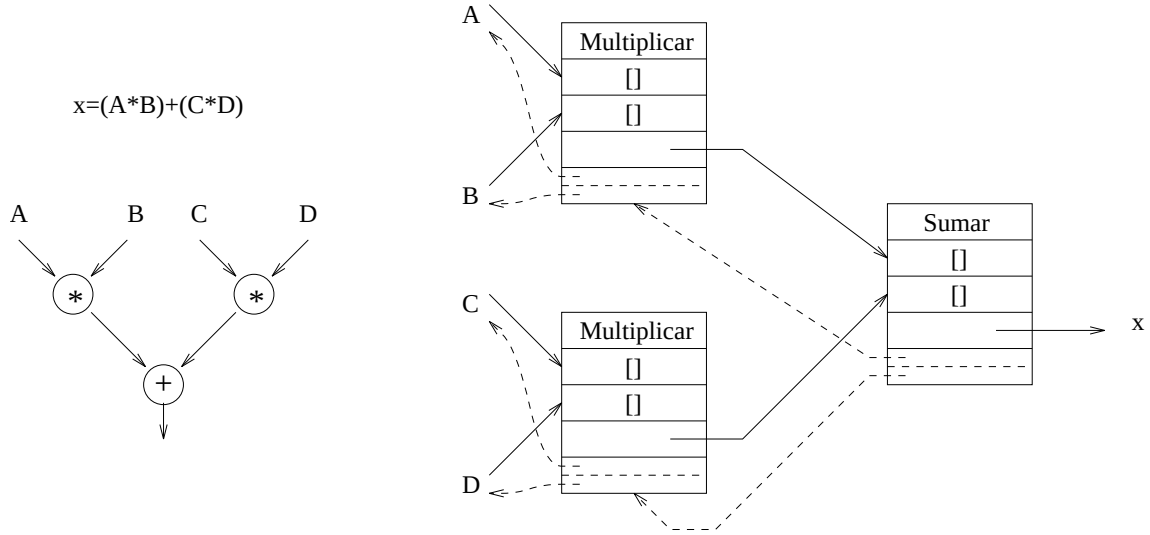

Figura 4.3: Ejemplo de máquina de flujo de datos estática (las flechas discontinuas son las señales de reconocimiento).

Comparado con el flujo de datos estático, el dinámico permite mayor paralelismo ya que una instrucción no necesita esperar a que se produzca un espacio libre en otra instrucción antes de guardar su resultado. Sin embargo, en el tipo dinámico es necesario establecer un mecanismo que permita distinguir los valores de diferentes conjuntos de operandos para una instrucción.

Uno de estos mecanismos consiste en formar una cola con los valores de cada operando en orden de llegada. Sin embargo, el mantenimiento de muchas largas colas resulta muy costoso. Para evitar las colas, el formato del paquete de resultado a menudo incluye un campo de etiqueta; con esto, los operandos que coinciden se pueden localizar comparando sus etiquetas. Una memoria asociativa, llamada *memoria de coincidencias*, puede ser usada para buscar las coincidencias entre operandos. Cada vez que un paquete de resultados llega a la unidad de memoria, sus campos de dirección y de etiqueta son guardados en la memoria de coincidencias. Tanto la dirección como la etiqueta serán utilizadas como clave para determinar qué instrucción está activa.

A pesar de que las máquinas convencionales, basadas en el modelo Von Neumann, tienen muchas desventajas, la industria todavía sigue fabricando de forma mayoritaria este tipo de computadores basados en flujo de control. Esta elección está basada en la efectividad/coste, lanzamiento al mercado, etc. Aunque las arquitecturas de flujo de datos tienen un mayor potencial de paralelización, todavía se encuentran en su etapa de investigación. Las máquinas de flujo de control todavía dominan el mercado.

4.2 Otras arquitecturas

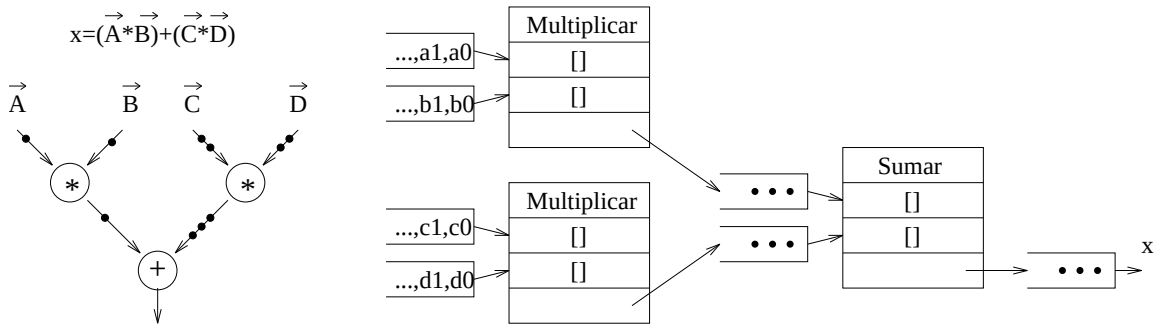

Figura 4.4: Ejemplo de máquina de flujo de datos dinámica. Un arco en grafo de flujo de datos puede llevar más de una etiqueta a un tiempo.