CAPITULO 4

PROCESAMIENTO EN PARALELO : PIPELINE.

4. - INTRODUCCIÓN A ARQUITECTURAS PARALELAS.

Muchos de los computadores antiguos y muchos de los minicomputadores contemporáneos son monoprocesadores, lo cual no debe sorprendernos ya que es la máquina más elemental para diseñar y construir.

Por otra parte, las computadoras digitales modernas de gran escala utilizan frecuentemente el procesamiento simultáneo (concurrencia) en distintos puntos del sistema, denominaremos a esta clase de computadoras Procesadores Paralelos.

Los computadores paralelos son sistemas de computadores consistentes de un conjunto centralizado de procesadores que pueden procesar simultáneamente los datos del programa.

El procesamiento paralelo se basa en la explotación de sucesos concurrentes en el proceso de cómputo.

Como apuntamos en el Capítulo 1 la concurrencia implica paralelismo, simultaneidad y pipelining.

Sucesos Paralelos ocurren en múltiples recursos durante el mismo intervalo de tiempo.

Sucesos Simultáneos ocurren en el mismo instante.

Sucesos Pipeline ocurren en lapsos superpuestos.

Se puede hablar de niveles de paralelismo, que caracterizamos de la siguiente manera:

- *Multiprogramación, Multiprocesamiento*: Estas acciones se toman a nivel de Programa o Trabajo.
- *Tarea o Procedimientos*: Acciones que se toman dentro de un mismo programa, ejecutándose procesos independientes en forma simultánea.
- *Interinstrucciones*: Acciones a nivel de instrucción, o sea, dentro de un mismo proceso o tarea se pueden ejecutar instrucciones independientes en forma simultánea.
- *Intrainstrucciones*: Acciones simultáneas que se pueden realizar para una misma instrucción, por ejemplo vectorización de operaciones escalares dentro de una instrucción compleja tipo DO, FOR, etc.

El paralelismo de un mayor nivel se obtiene por medio de algoritmos, los de menor nivel con importante actividad del hardware.

Últimamente ciertas técnicas del procesamiento distribuido son incorporadas a arquitecturas centralizadas para conseguir mayor grado de paralelismo.

El paralelismo puede obtenerse de distintas maneras, a saber:

- **Multicomputadoras**: Computadoras independientes, muy a menudo una de ellas actúa como supervisor, que realizan una tarea común en una sola ubicación (una configuración muy común, aunque ciertamente limitada, es la minicomputadora como preprocesador de un computador mainframe).
- **Multiprocesadores**: Un conjunto de unidades de cómputo, cada una de las cuales tiene sus propios conjuntos de instrucciones y datos, compartiendo una misma memoria. Los computadores multiprocesadores consisten en un número n mayor o igual a 2 de procesadores que operan simultáneamente sobre una misma memoria, y están interconectados mediante canales que transmiten comandos de control y datos. Están controlados por un único Sistema Operativo.
- **Redes de computadoras**: Computadoras independientes conectadas mediante un canal de manera tal que los recursos propios disponibles en un punto de la red pueden estar disponibles para todos los miembros de la red.
- **Procesador Pipeline:** Un solo computador el cual puede realizar simultáneamente operaciones de cálculos en determinadas secciones, con diferentes estadios de completitud. Los procesadores pipeline se basan en el principio de dividir los cálculos entre una cantidad de unidades funcionales que operan simultáneamente existiendo superposición.
- **Procesador Array:** Un grupo de unidades de cómputo cada una de las cuales realiza simultáneamente la misma operación sobre diferentes conjuntos de datos. Los procesadores array operan sobre vectores. Las instrucciones del computador vectorial son ejecutadas en serie (como en los computadores clásicos) pero trabajan en forma paralela sobre vectores de datos.

4.1. - CLASIFICACIÓN DE FLYNN.

A grandes rasgos, la ejecución de una instrucción puede verse como etapas distintas, que realizan:

- Búsqueda de la instrucción,
- Decodificación de la instrucción,
- Búsqueda de los operandos, y
- Ejecución de la instrucción.

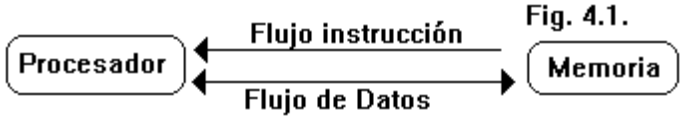


Fig. 4.1.

Las instrucciones pueden verse como un flujo de instrucciones que se desplazan de memoria al procesador y los operandos como un flujo de datos que se desplazan entre memoria y el procesador (Fig. 4.1).

Analizando el ciclo de una instrucción se puede diferenciar a un procesador en dos unidades funcionales:

La unidad de control (CU): decodifica las instrucciones y envía señales a una Unidad de Procesamiento.

La unidad de procesamiento (PU): ejecuta las instrucciones decodificadas y envía los resultados a la unidad funcional memoria.

M. J. Flynn en 1966 realizó una clasificación del paralelismo presente en un procesador basado en el número de Flujo de Instrucciones y de Datos simultáneos.

Combinando flujo de instrucciones y flujo de datos surgen 4 grupos de procesadores posibles.

En el caso de monoprocesamiento :

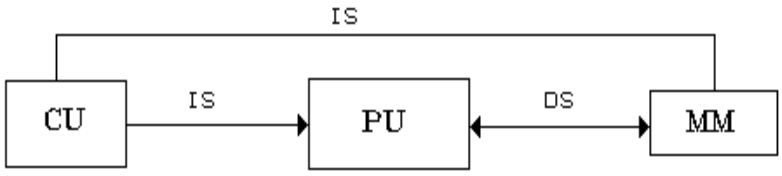


Fig.4.2. - Computador SISD.

CU: Unidad de Control
PU : Unidad de Procesamiento
MM : Módulo de memoria
SM : Memoria compartida
IS : Flujo de instrucción
DS : Flujo de dato

Tabla 4.3.

Este esquema se encuentra normalmente en la literatura como SISD (Single Instruction Single Data), o sea una sola instrucción, un solo dato a la vez.

Por ejemplo si se tuviera un programa de las siguientes características :

```
      DO 20  I=1,10
      A(I) = A(I) * 3
20    CONTINUE
```

Las diez veces que se pasa por la asignación, serán diez instrucciones, diez ejecuciones completamente independientes una de la otra.

Si se pudiera implementar el ejemplo anterior en forma paralela, en diez unidades de procesamiento simultáneas bajo el control de una única unidad de control ejecutaríamos las diez instrucciones en forma simultánea, con lo cual se ahorra mucho tiempo.

Esta idea se traduce esquemáticamente en la arquitectura que se visualiza en la Fig. 4.4.

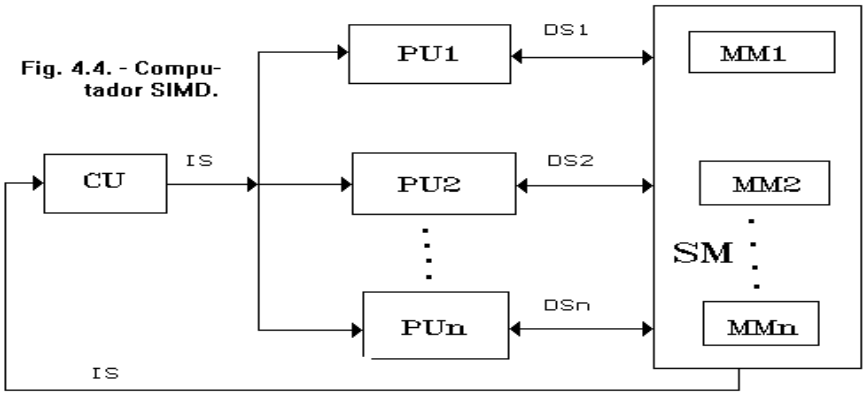


Fig. 4.4. - Computador SIMD.

O sea, tener **una** memoria, **una** unidad de control, y de dicha unidad de control tener **varias** unidades de ejecución que depositarían sus datos en memoria. A estas arquitecturas se las encuentra en la literatura como SIMD (Single Instruction Multiple Data), o sea una sola instrucción que ataca a muchos datos. Este es el esquema más básico, de lo que comúnmente se conoce como un procesador vectorial. Que es capaz de tomar un vector o matriz, y simultáneamente, hacer muchas operaciones sobre él.

Luego existe un caso distinto :

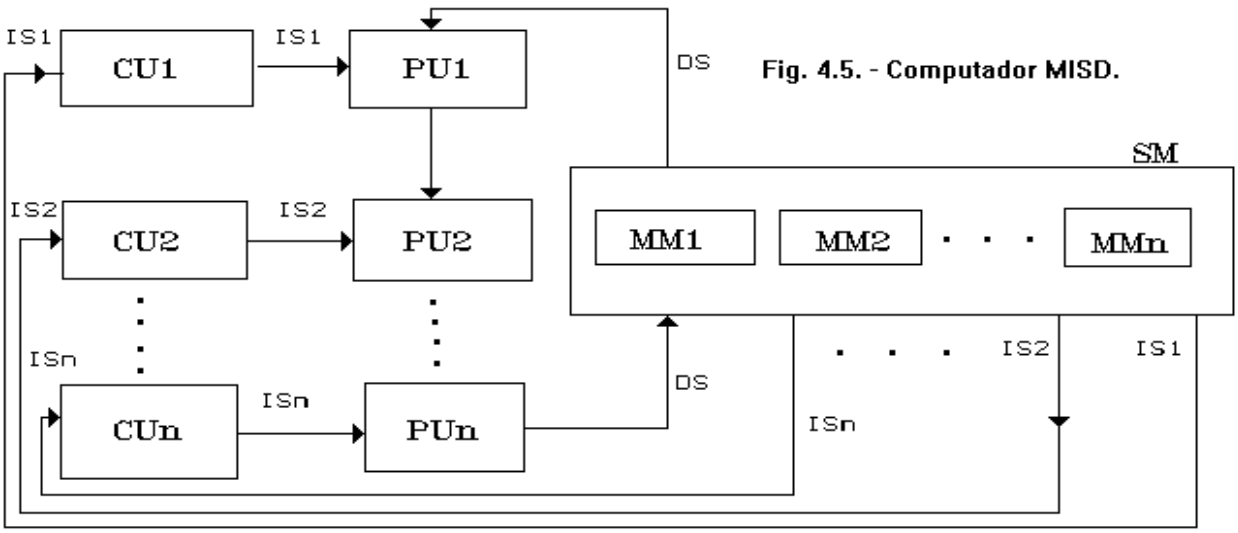


Fig. 4.5. - Computador MISD.

Aquí se tienen varias unidades de control, con varias unidades de ejecución, existen distintos flujos de instrucciones y un **único flujo de datos**. O sea se tienen múltiples instrucciones con un solo dato, MISD (Multiple Instruction Single Data).

Es un esquema teórico, en realidad no existe ningún procesador que esté implementado.
Y por último se tiene :

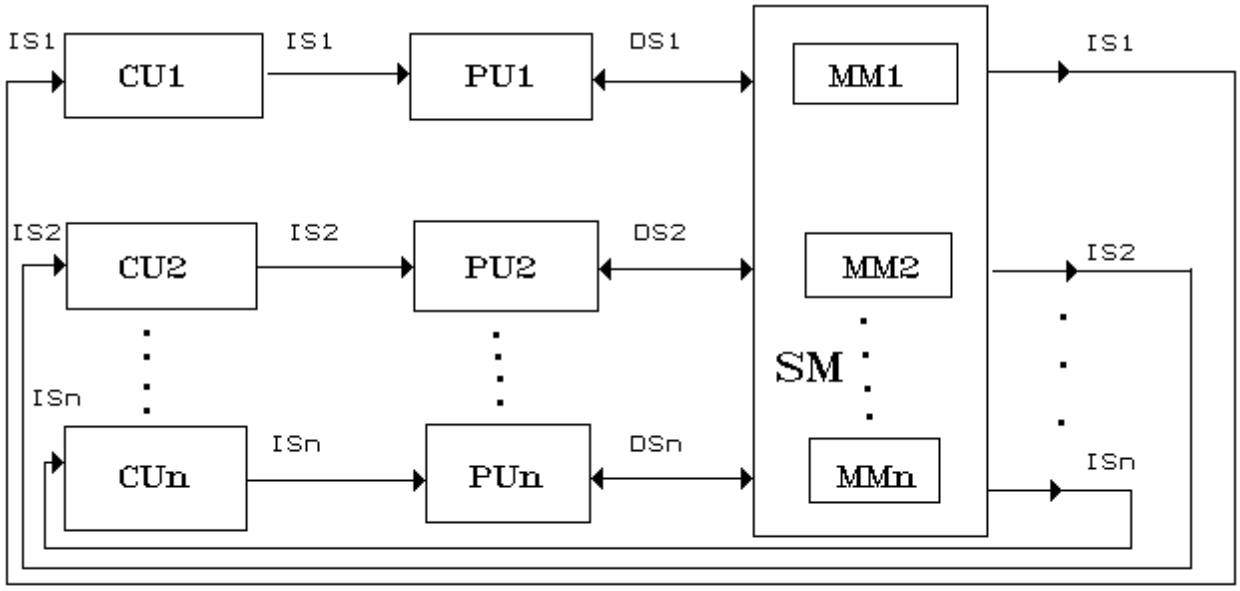


Fig. 4.6. - Computador MIMD.

Varias unidades de control, con varias unidades de ejecución. Es decir, se ejecutan muchas instrucciones con mucho datos, MIMD (Multiple Instruction Multiple Data), siendo éste el caso típico de un sistema de multiprocesamiento.

4.2. - PROCESAMIENTO EN SERIE versus PROCESAMIENTO EN PARALELO.

Tsé-Yun Feng (1972) sugirió la utilización del grado de paralelismo para clasificar las distintas arquitecturas de computadoras. La cantidad máxima de bits que pueden ser procesados dentro de una unidad de tiempo por un computador, es lo que se denomina máximo grado de paralelismo.

Consideremos un conjunto de m palabras de n bits cada una, llamamos un bit-slice a una columna vertical de bits de un conjunto de palabras en la misma posición (ver figura 4.7).

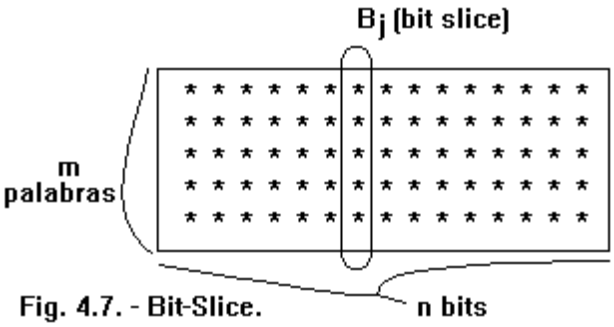


Fig. 4.7. - Bit-Slice.

La figura 4.8 muestra la clasificación de las computadoras por su máximo grado de paralelismo.

El eje horizontal muestra el tamaño de la palabra n. El eje vertical corresponde al tamaño de la franja de bits m (bit-slice). Ambas medidas están dadas en la cantidad de bits que contiene tanto una palabra o una franja de bits.

El máximo grado de paralelismo P(C) está dado por el producto de m y n:

$$P(C) = m * n$$

De este diagrama se desprende que existen cuatro tipos de procesamiento:

- Palabra en serie y bit en serie (WSBS - Word Serial/Bit Serial)
- Palabra en paralelo y bit en serie (WPBS - Word Parallel/Bit Serial).
- Palabra en serie y bit en paralelo (WSBP - Word Serial/Bit Parallel).
- Palabra en paralelo y bit en paralelo (WPBP - Word Parallel/Bit Parallel).

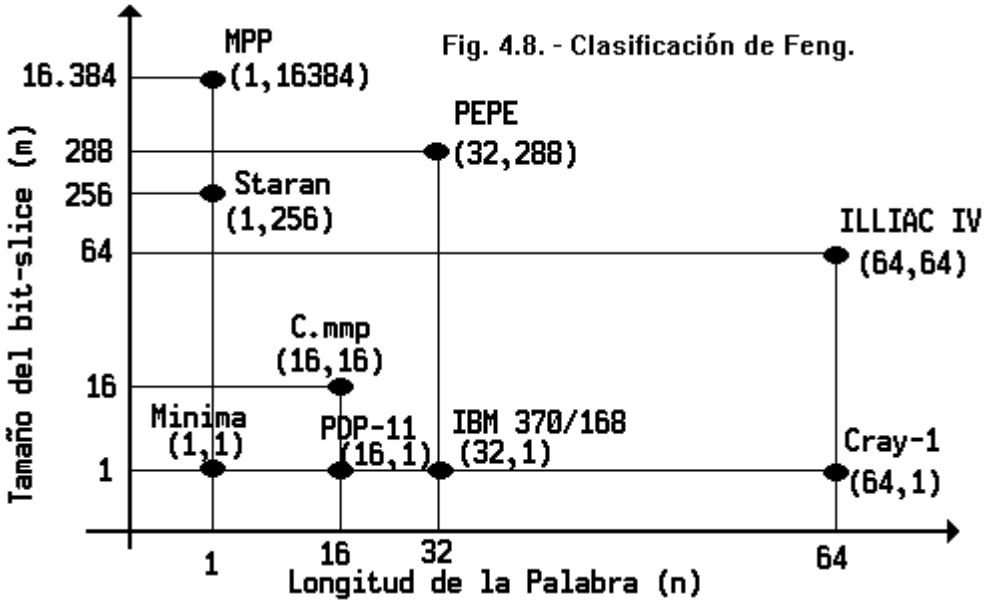


Fig. 4.8. - Clasificación de Feng.

WSBS ha sido llamado el procesamiento en serie de bits, ya que se procesa un bit por vez (n = m = 1). Es el procesamiento más lento, y solo existió en la primera generación de computadoras.
WPBS (n = 1, m >> 1) ha sido denominado el procesamiento **bis** (de bit-slice) ya que se procesa una porción de m bits a la vez. (Ej.: STARAN).
WSBP (n > 1, m = 1), tal como se encuentra en la mayoría de las computadoras existentes, ha sido llamado **procesamiento de porciones de palabra**, ya que se procesa una palabra de n bits a la vez. (Ej.: arquitectura de Von Neumann).
Finalmente, el WPBP (n > 1, m > 1) es conocido como **procesamiento paralelo total** (o más simplemente como procesamiento en paralelo) en el cual se procesa por vez un arreglo de n x m bits, siendo el más veloz de los cuatro modos vistos.

4.3. - BALANCE DEL ANCHO DE BANDA DEL SUBSISTEMA (BANDWIDTH).

La estructura de bus común tiene el inconveniente que ofrece una comunicación en cuello de botella.

Cada acceso a memoria de un ordenador desaprovecha accesos a millones de bits cuando escoge unos pocos para enviar a través del bus desde la memoria a la unidad central de proceso.

Este despilfarro se tolera por dos razones:

- Primero, porque simplifica nuestro concepto de la máquina y se adapta a nuestra inclinación natural de hacer las cosas una a una.
- Segundo, nos suministra una sola y simple interconexión entre las distintas partes de la máquina.

En general, la CPU es la unidad más veloz dentro de una computadora. Mediremos su ciclo en un tiempo Tp dado en decenas de nanosegundos; el de la memoria, en Tm dado en cientos de nanosegundos; y el del subsistema de E/S, que es el más lento, en Td, dado en unos pocos milisegundos. Entonces:

$$Td > Tm > Tp$$

Ej.: En la IBM 370/168, Td = 5 ms (discos); Tm = 320 ns y Tp = 80 ns.

Se define el **ancho de banda o bandwidth** de un subsistema como la cantidad de operaciones realizadas por unidad de tiempo. Para el caso de memoria es la cantidad de palabras de memoria que pueden accederse por unidad de tiempo.

Es importante al hacer mediciones comparativas entre los anchos de banda de los diferentes subsistemas tener bien en claro ciertos conceptos.

Cuando se mide el ancho de banda de la memoria (o equivalentemente el del subsistema de E/S) su visualización no es complicada debido a que es bastante sencillo pensar que, ya que la memoria en un dispositivo pasivo, lo que se mide es la cantidad de información (bytes, palabras, instrucciones, etc.) que la memoria puede transmitir en una cierta unidad de tiempo.

En cambio hablar del ancho de banda de la CPU, que es un dispositivo activo, significa en cierta forma la capacidad de proceso de la misma, es decir más burdamente hablando, que cantidad de instrucciones puede ejecutar por unidad de tiempo.

Si lo que se desea es comparar los anchos de banda de la memoria y la CPU debe ponerse especial cuidado en qué patrón de comparación se está utilizando. Por ejemplo, utilizando los guarismos anteriores, si la IBM 370/168 ejecuta 50 instrucciones en 80 ns su memoria demora 320 ns en transferir esas 50 instrucciones a la CPU.

Sea W la cantidad de palabras que se obtienen por cada ciclo de memoria Tm; luego, el ancho de banda máximo de la memoria es:

$$Bm = W / Tm \quad \text{(Palabras o bytes)}$$

Ej.: La IBM 3033 tiene un ciclo de procesador de Tp = 57 ns. Por cada ciclo de memoria de Tm = 456 ns pueden obtenerse 8 palabras dobles de 8 bytes c/u. a partir de un sistema de memoria con 8 elementos lógicos de almacenamiento en forma intercalada (interleaved). Luego:

$$Bm = 8 * 8 \text{ bytes} / 456 \text{ ns} = 134 \text{ Mb/s}$$

Pero debido a esta partición lógica de la memoria pueden producirse determinados conflictos al querer acceder a una posición, y por lo tanto, el ancho de banda útil (Bmu) será menor:

$$Bmu \leqslant Bm$$

Se sugiere una medida del tipo:

$$Bmu = Bm / M^{1/2}$$

Donde M es la cantidad de módulos de memoria del sistema de memoria. Luego, en el ejemplo anterior, se tiene:

$$Bmu = 134 / 8^{1/2} = 47.3 \text{ Mb/s}$$

En cuanto a dispositivos externos, el concepto de bandwidth se complica un poco. Piense que, por ejemplo, según los tiempos de latencia y rotación, la tasa de transferencia de una unidad de disco puede variar. En general nos referimos a la tasa de transferencia promedio de una unidad de disco como el bandwidth Bd del disco; un valor típico es del orden de 3 Mb/s. En una unidad de cinta el valor típico ronda los 1.5 Mb/s. Las impresoras, lectoras y terminales son mucho más lentas aún. Usando múltiples drivers estas tasas aumentan.

El bandwidth de un procesador se mide como el máximo porcentaje de cómputo de la CPU; por ejemplo, 160 megaflops (millones de instrucciones de punto flotante por segundo) en los computadores Cray-1, y de 12.5 MIPS (millones de instrucciones por segundo) en los computadores IBM 370/168. Estos son valores pico obtenidos de la división de 1/Tp = 1/12.5 ns y 1/80 ns respectivamente.

En la realidad, el porcentaje útil de CPU es Bpu menor o igual a Bp. Este porcentaje de utilización se calcula sobre la cantidad de resultados (o palabras) por segundo:

Bpu = Rw / Tp (palabras/s)

donde Rw es la cantidad de resultados medidos en palabras, y Tp es el tiempo necesario para generar esos resultados Rw.

Por ejemplo, el CDC Cyber-205 tiene un porcentaje pico de 200 MFLOPs para resultados de 32 bits, y solamente de 100 MFLOPs para resultados de 64 bits.

En las computadoras actuales se observa la siguiente relación entre los bandwidths de los distintos subsistemas:

Bm $\geqslant$ **Bmu**,
Bmu $\geqslant$ **Bp**,
Bp $\geqslant$ **Bpu**, y
Bpu $\geqslant$ **Bd**

Esto significa que la memoria principal tiene el mayor bandwidth, ya que debe ser accedida tanto por la CPU como por los dispositivos de E/S, es decir que del porcentaje efectivo de información transferida por unidad de tiempo por la memoria una parte se dirige a la CPU y otra corresponde al intercambio de información con los medios externos. Por lo tanto necesitamos igualar la potencia de procesamiento de los tres subsistemas. A continuación describimos los dos mecanismos más usados:

4.3.1. - **Balance entre CPU y Memoria.**

La diferencia de velocidad entre la CPU y la memoria puede achicarse mediante el uso de memoria cache de alta velocidad. La cache tiene un tiempo de acceso Tc = Tp. Un bloque de palabras de memoria es movido a la cache (Por ej. bloques de 16 palabras en la IBM 3033) de tal manera que los datos e instrucciones están disponibles inmediatamente para su uso; también puede servir como un buffer para instrucciones.

4.3.2. - **Balance entre dispositivos de E/S y Memoria.**

Pueden utilizarse canales de diferentes velocidades entre los dispositivos lentos y la memoria. Estos canales de E/S realizan funciones de bufferización y multiplexamiento para la transferencia de datos desde muchos discos y la memoria principal mediante el robo de ciclos a la CPU. Incluso pueden utilizarse controladores de disco inteligentes para filtrar la información irrelevante de las pistas del disco, lo cual aliviaría la saturación de los canales de E/S.

En el caso ideal desearemos alcanzar un balance total del sistema, en el cual el bandwidth de la memoria coincida con la suma del bandwidth del procesador y de los Dispositivos de E/S, es decir:

Bpu + Bd = Bmu

donde Bpu = Bp y Bmu = Bm han sido maximizados ambos.

4.4. - **Paralelismo con una sola CPU**

Procesamiento paralelo es un término utilizado para denotar operaciones simultáneas en la CPU con el fin de aumentar su velocidad de cómputo. En lugar de procesar cada instrucción secuencialmente como en las arquitecturas convencionales, un procesador paralelo realiza tareas de procesamiento de datos e instrucciones concurrentemente.

Para lograr concurrencia en un sistema con un solo procesador se utilizan técnicas de paralelismo que se consiguen multiplicando los componentes de hardware, o técnicas de pipelining.

Introduciremos las características básicas de los **computadores paralelos**, que son aquellos sistemas que enfatizan el procesamiento en paralelo. Estos computadores se pueden dividir en tres configuraciones según la arquitectura:

- Computadores Pipeline (tratados en el presente capítulo).
- Procesadores Matriciales - Array Processors (ver capítulo 5).
- Sistemas Multiprocesadores (ver capítulo 6).

Un computador pipeline hace operaciones superpuestas para explotar el **paralelismo temporal**. Un Array Processor usa ALUs múltiples sincronizadas, para lograr **paralelismo espacial**. Un sistema multiprocesador logra **paralelismo asincrónico** a través de un conjunto de procesadores que interactúan y comparten recursos (periféricos, memorias, bases de datos, etc.).

El pipelining es una forma económica de hacer paralelismo temporal en computadoras. La idea es la misma que la de las líneas de montaje de las plantas industriales. Se divide la tarea en una secuencia de subtareas, cada una de las cuales se ejecuta en una etapa de hardware especializada que trabaja concurrentemente con otra de las etapas del pipeline. Esto permite aumentar el throughput del sistema de forma considerable.

4.5. - COMPUTADORES PIPELINE

Veamos el ejemplo de un pipeline de cuatro etapas: el proceso de ejecución de una instrucción en un computador digital envuelve cuatro pasos principales: levantar la instrucción de memoria (Instruction Fetch - IF); identificar la operación que debe efectuarse (Instruction Decoding - ID); levantar los operandos si son necesarios en la ejecución (Operand Fetch - OF); y ejecutar la operación aritmético lógica que ha sido decodificada. Antes de comenzar a ejecutar una nueva instrucción deben completarse estos cuatro pasos.

La idea de un computador pipeline la vemos en la figura 4.9 : hay cuatro etapas IF, ID, OF y EX ordenadas de forma de una "cascada lineal". Las instrucciones sucesivas se ejecutan de forma superpuesta. La diferencia entre la ejecución superpuesta de instrucciones y la ejecución no superpuesta secuencial se muestra en los diagramas de espacio/tiempo de las Fig. 4.10 y 4.11.

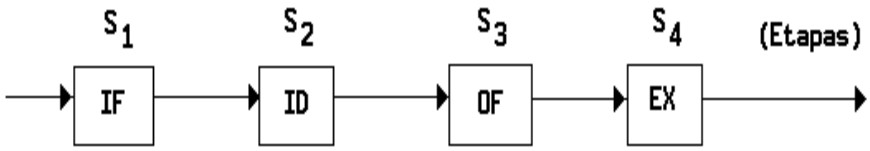


Fig. 4.9. - Un procesador pipeline.

Cada "columna" del gráfico representa un ciclo de pipeline, que es aproximadamente igual al tiempo que tarda la etapa más lenta. Al computador sin pipeline le toma cuatro ciclos de pipeline completar una instrucción, en cambio, un pipeline produce un resultado de salida por cada ciclo luego de cargado el pipe. El ciclo de instrucción ha sido reducido efectivamente a un cuarto del tiempo de ejecución original, por medio de las ejecuciones superpuestas.

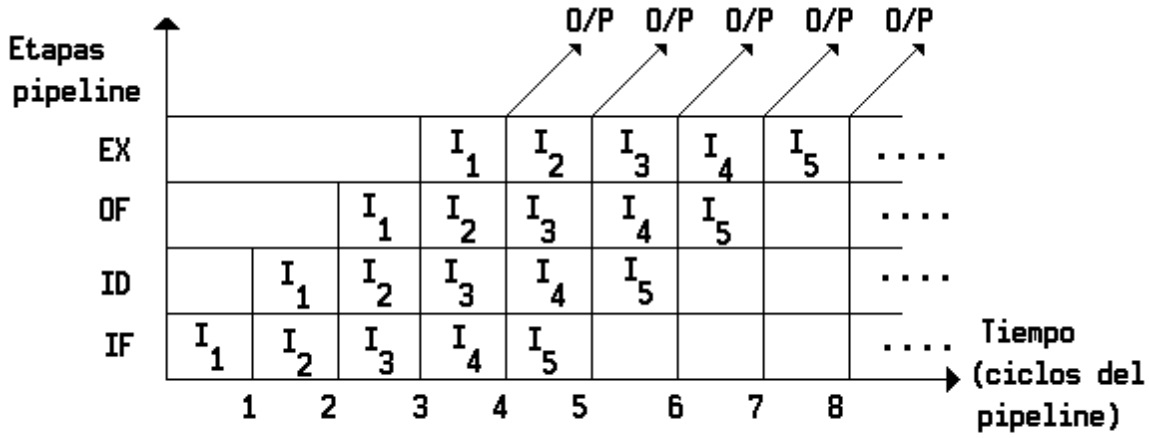


Fig. 4.10. - Diagrama espacio-temporal para un procesador pipeline.

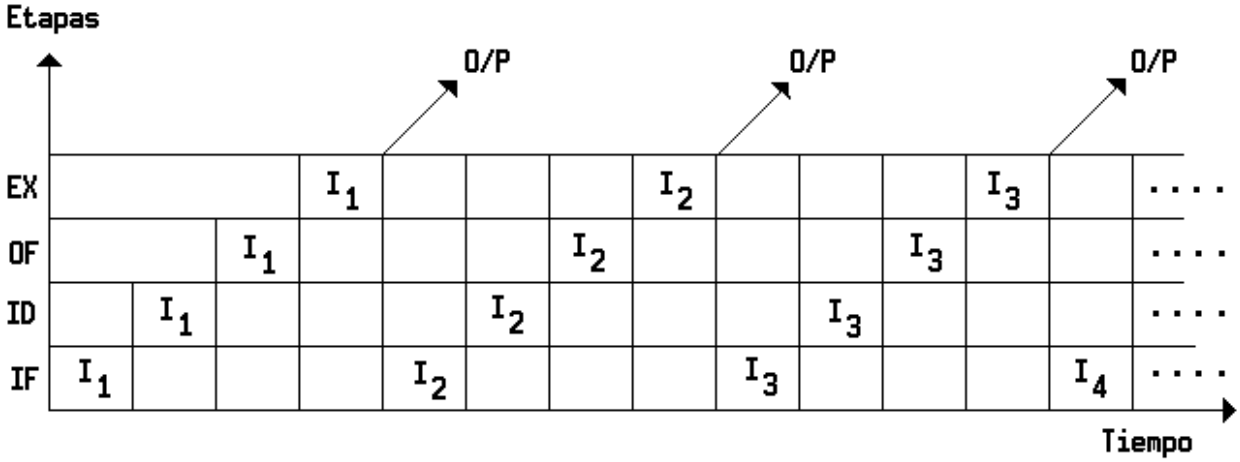


Fig. 4.11. - Diagrama espacio-temporal para un procesador no-pipeline.

Lo que hemos descripto anteriormente, es un **pipeline de instrucción**.

La **máxima velocidad** a la cual pueden ingresar las instrucciones al pipeline depende exclusivamente **del tiempo máximo** requerido para **atravesar una etapa** y **del número de ellas**.

Hay ciertas dificultades que impedirán al pipeline operar con su máxima velocidad. Los segmentos pueden tomar tiempos diferentes para cumplir su función sobre los datos que llegan. Algunos segmentos son salteados por ciertas instrucciones. Por ejemplo, una instrucción modo registro no necesita un cálculo de dirección de operando; por otra parte, dos o más segmentos pueden requerir acceso a memoria al mismo tiempo, haciendo que un segmento tenga que esperar hasta que el otro termine. (Los conflictos de acceso a memoria se resuelven a menudo utilizando técnicas de interleaving).

4.5.1. - Principios de pipelining lineal

Las líneas de montaje de las plantas han sido utilizadas para aumentar productividad. Su forma original es una línea de flujo (pipeline) de estaciones de montaje en donde los ítems se ensamblan continuamente.

Idealmente todas las etapas tienen que tener igual velocidad de procesamiento, porque sino la más lenta se transforma en un cuello de botella de todo el pipe. La subdivisión de la tarea de entrada en una secuencia apropiada de subtareas es un factor crucial para determinar la performance del pipeline.

En un pipeline de tiempo uniforme, todas las tareas tienen igual tiempo de procesamiento en todas las estaciones. Sin embargo, en la realidad las estaciones sucesivas tienen tardanza de tiempo distinta. La partición óptima del pipeline depende de un número de factores, incluyendo la calidad de las unidades de trabajo (eficiencia y capacidad), la velocidad deseada y la efectividad en costo de toda la línea.

El tiempo que se tarda en obtener resultados continuos es lo que se conoce como **"tiempo de carga del pipe"**.

Esto se traduce en una mejora de la performance, ya que si una función (no sólo instrucciones) se lleva a cabo en T segundos, en la manera convencional, al utilizar un procesador pipeline de N etapas, esa misma función podrá realizarse en T/N segundos.

Obviamente este es un resultado teórico, pues dependerá de la cantidad de operaciones que pueden estructurarse en pipeline y la calidad (pueden ser aritmética de punto flotante, ciclo de instrucciones, etc.).

Dada una tarea T, la podemos subdividir en un conjunto de subtareas {T1,...,Tk}. La relación de precedencia de este conjunto implica que una tarea Tj no puede comenzar hasta que otra tarea anterior Ti (i < j) no haya terminado. Las interdependencias de todas las subtareas forman el **grafo de precedencia**. Con una relación de precedencia **lineal**, la tarea Tj no puede comenzar hasta que todas las subtareas anteriores {Ti para todo i no mayor a j} terminen. Un **pipeline lineal** puede procesar una sucesión de subtareas que tengan un grafo de precedencia lineal.

En la figura 4.12 se muestra la estructura básica de un pipeline lineal. El mismo consiste de una cascada de etapas de procesamiento. Las etapas son circuitos que efectúan operaciones aritméticas o lógicas sobre el conjunto de datos que fluyen a través del pipe. Están separadas por registros de muy
alta velocidad que almacenan los resultados intermedios entre etapas, llamados **latches**. La información que fluye entre etapas adyacentes está bajo el control de un reloj común, que se aplica a todos los latches simultáneamente.

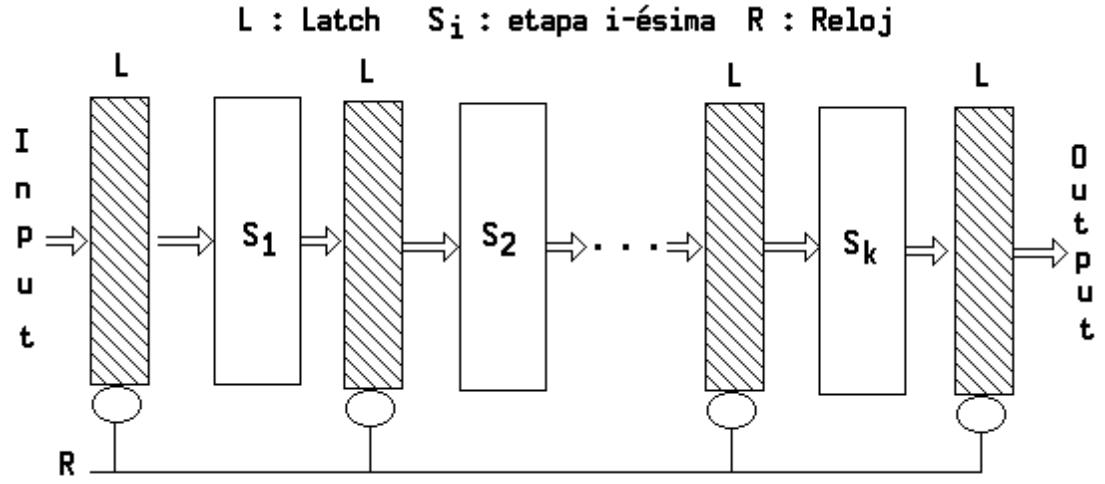


Fig. 4.12. - Estructura de un procesador pipeline lineal.

Para poder aplicar pipeline, o sea la partición en subfunciones de una función se deben dar las siguientes condiciones :

1.- La evaluación de la función es equivalente a la evaluación secuencial de las subfunciones.

2.- La entrada a una subfunción proviene exclusivamente de subfunciones previas en la secuencia de evaluación.

3.- Excepto el intercambio de E/S, no existe otra vinculación entre las subfunciones.

4.- Disponer del hardware suficiente para evaluar las subfunciones.

5.- El tiempo de evaluación de cada subfunción es aproximadamente el mismo.

El flujo de datos dentro del pipeline es **discreto** y se desplazan de etapa en etapa sincronizados por un reloj.

Para evitar que los datos de una etapa ingresen a la siguiente, antes que haya finalizado el proceso anterior se utilizan elementos de memoria en la Entrada y Salida de cada etapa controlados por un reloj asociado (memoria ó **latch**, como se mostró en la Fig. 4.12).

4.5.1.1. - El período de reloj

La circuitería lógica en cada etapa Si tiene una tardanza de tiempo Ti.

Sea T_l la tardanza en llenar cada latch; llamamos entonces período de reloj del pipeline T a:

$$T = \text{máximo } \{T_i\}_{1k} + T_l = T_m + T_l$$

La **frecuencia** del período del reloj se define como f = 1 / T.

Para ilustrar las operaciones superpuestas en un procesador pipeline lineal, podemos dibujar un diagrama de espacio-tiempo. En la figura 4.13. se muestra el diagrama de espacio-tiempo de un procesador pipeline de cuatro etapas.

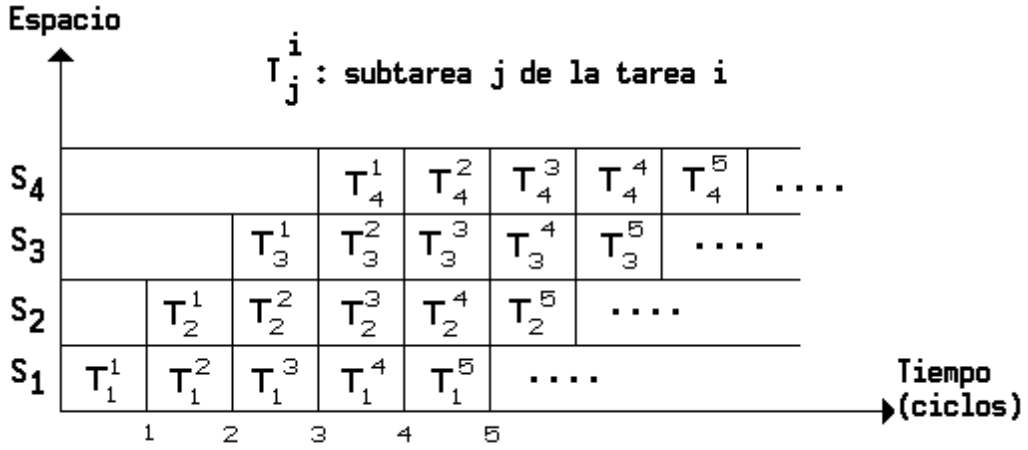


Fig. 4.13. - Diagrama espacio-temporal mostrando el solapamiento de etapas.

Como hemos visto, una vez que el pipe se llena sale un resultado por período de reloj, independientemente del número de etapas en el pipe. Idealmente, un pipeline lineal con k etapas procesa n tareas en $T_k = k + (n-1)$ períodos de reloj. Se usan k ciclos para llenar el pipeline (con la primer instrucción), y hacen falta n-1 ciclos para completar las restantes n-1 tareas. El mismo número de tareas puede ejecutarse en un procesador sin pipeline en un tiempo $T_1 = n * k$.

4.5.1.2. - Aceleración

Definimos la aceleración de un pipeline lineal de k etapas sobre un procesador sin pipeline equivalente como:

$$S_k = T_1 / T_k = (n * k) / (k + (n - 1))$$

Nótese que la máxima aceleración de Sk tiende a k para n tendiendo a infinito.

En otras palabras, la máxima aceleración de un pipeline lineal es k, donde k es el número de etapas en el pipe, y, en consecuencia, la aceleración es mayor cuantas más instrucciones se puedan procesar.

Esta máxima aceleración nunca se alcanza debido a dependencias de datos entre instrucciones, interrupciones, bifurcaciones del programa y otros factores.

4.5.1.3. - Eficiencia

Si volvemos a la figura 4.13., el producto de un intervalo de tiempo y el espacio de la etapa forman un área llamada el **lapso de espacio-tiempo**. Un lapso de espacio-tiempo determinado puede estar en un estado de Ocupado o Libre, pero no en ambos (Por ej. el lapso de espacio-tiempo 4-S_3 está ocupado por T^3_3).

Utilizamos este concepto para medir la performance de un pipeline: llamaremos la **eficiencia** de un pipeline al porcentaje de lapsos de espacio-tiempo ocupados sobre el total de lapsos de espacio-tiempo (libres más ocupados). Sea n el número de tareas, k el número de etapas del pipeline, y T el período de reloj de un pipeline lineal. Entonces la eficiencia del pipeline se define como:

$$\text{Eficiencia o EF} = (n * k * T) / (k (kT + (n-1) T) = n / (k + (n - 1))$$

Notar que si n tiende a infinito entonces la eficiencia tiende a 1. Esto implica que cuanto mayor es el número de tareas que fluyen por el pipeline, mayor es su eficiencia. Además Eficiencia = Sk / k. Esto nos da otra forma de ver la eficiencia: la relación entre la aceleración real y la aceleración ideal k.

4.5.1.4. - Throughput

Es el número de tareas que puede ser completado por un pipeline por unidad de tiempo. Esta tasa refleja el poder de computación de un pipeline. En términos de la Eficiencia y el período de reloj T de un pipeline lineal, definimos el throughput como sigue:

$$\text{Throughput o TH} = n / (k * T + (n-1) * T) = EF / T$$

donde n es igual al número total de tareas procesadas durante un período de observación k * T + (n-1) * T.

En el caso ideal, TH = 1/T = f, cuando EF tiende a 1. Esto significa que el máximo throughput de un pipeline lineal es igual a su frecuencia, la que corresponde a **un** resultado de salida por período de reloj.

4.5.1.5. - Pipeline versus Solapamiento.

Si bien tienen significados parecidos y en muchos casos los usaremos como sinónimos no son exactamente equivalentes.
Las condiciones del **Pipeline** son :

1.- Cada evaluación de funciones básicas es independiente de la anterior.
2.- Cada evaluación requiere aproximadamente la misma secuencia de subfunciones.
3.- Cada subfunción se encadena perfectamente con la anterior.
4.- Los tiempos de las subfunciones son aproximadamente iguales.

Las condiciones de **Solapamiento** son :

1.- Existe dependencia entre las distintas evaluaciones (funciones).
2.- Cada evaluación puede requerir una secuencia diferente de subfunciones.
3.- Cada subfunción tiene un propósito distinto.
4.- El tiempo de cada evaluación no es necesariamente constante, sino que depende de la función y de los datos que la atraviesan.

El primer caso se lo suele denominar **Pipeline Sincrónico o Unifunción** y el segundo **Pipeline Asincrónico o Multifunción.**

Un ejemplo de Pipeline Unifunción está dado por una unidad destinada a realizar sumas en punto flotante. El resultado de una suma **no** depende de la anterior y la secuencia de etapas de las sumas es siempre la misma.

Un ejemplo de Pipeline Multifunción es la ejecución de instrucciones en un procesador, donde cada instrucción puede desarrollar caminos diferentes a través del Pipeline, por ejemplo al decodificarse una instrucción de salto la próxima etapa que debe continuar para dicha instrucción no consiste en obtener la o las direcciones de los operandos sino en alterar la dirección de la próxima instrucción que debe ingresar en el pipe, vaciando (flush) previamente aquellas instrucciones que ingresaron hasta el momento en que se detectó la instrucción de salto.

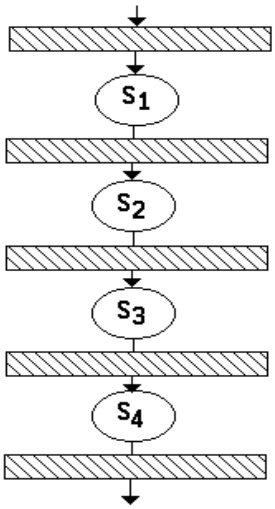


Fig. 4.14. - Pipeline Aritmético.

4.6. - CLASIFICACIONES DE PROCESADORES PIPELINE.

De acuerdo a los niveles de procesamiento, Händler (1977) ha propuesto el siguiente esquema de clasificación para los procesadores pipeline:

4.6.1. - **Pipelines aritméticos**

La ALU de un computador puede dividirse para hacer operaciones de pipeline en varios formatos. Hay ejemplos bien claros en los pipelines usados en la Star-100, TI-ASC, Cray-1 y Cyber 205 (Fig. 4.14).

4.6.2. - **Pipelines de instrucción**

La ejecución de un flujo de instrucciones puede hacerse en forma de pipeline, como vimos en la primer parte del capítulo, superponiendo la ejecución de la instrucción actual con las acciones de levantar, decodificar instrucciones y levantar operandos.

Esta técnica también es conocida con el nombre de **lookahead de instrucciones**. Casi todos los computadores de alta performance actuales tienen pipelines de instrucción (Ver Fig. 4.15).

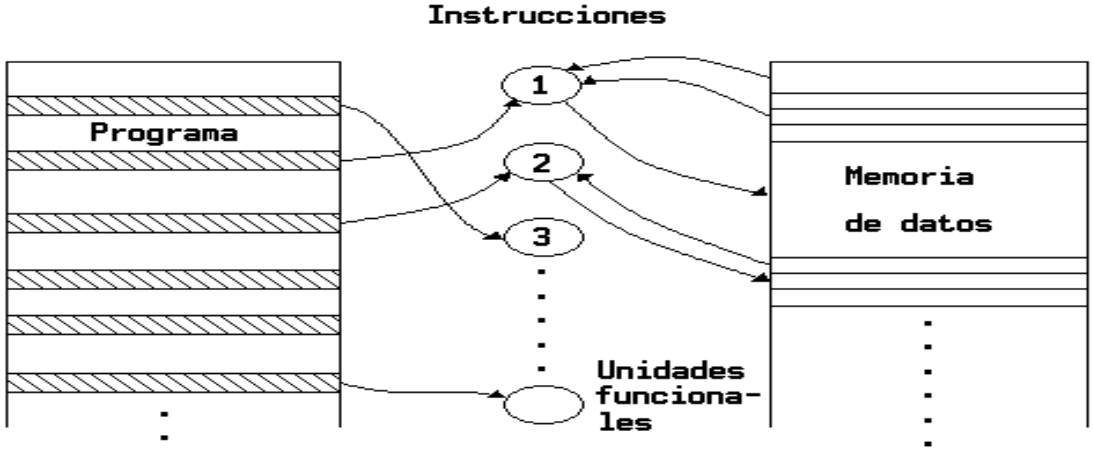


Fig. 4.15. - Pipeline de instrucciones.

4.6.3. - **Pipelines de procesador**

Se refiere al procesamiento del mismo flujo de datos por una cascada de procesadores, cada uno de los cuales procesa una tarea específica. El flujo de datos pasa el primer procesador, cuyo resultado se almacena en un bloque de memoria intermedia, que también es accesible por el segundo procesador. Este pasa el resultado refinado al tercero, y así siguiendo. En el momento en que Händler escribió su clasificación de los pipelines aún no existían implementaciones de pipelines de procesador (Fig. 4.16).

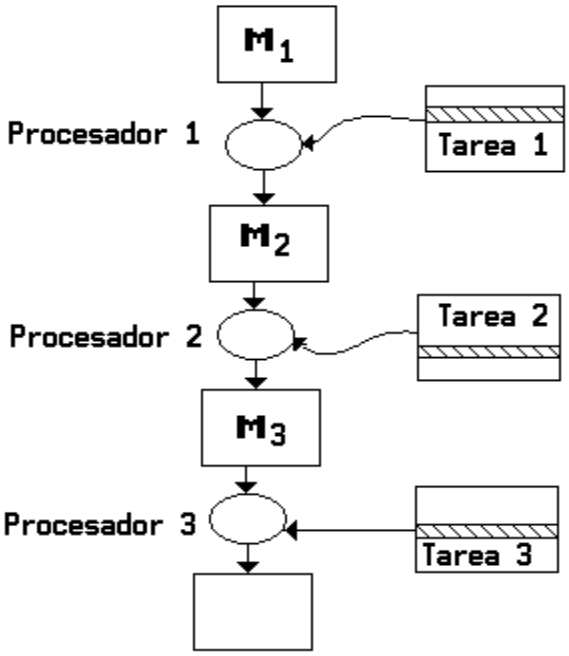


Fig. 4.16. - Pipeline de procesadores.

De acuerdo a las configuraciones del pipeline y estrategias de control, Ramamoorthy y Li (1977) han propuesto los siguientes tres esquemas de clasificación:

4.6.4. - **Pipelines unifuncionales versus multifuncionales**

Un pipeline unifuncional es aquel que tiene una sola función y dedicada, como el que mostraremos en el Ejemplo 1 (sumador de punto flotante).

Un pipeline multifuncional puede efectuar distintas funciones, por medio de interconexiones de varios subconjuntos de etapas en el pipeline, ya sea en diferentes momentos o al mismo tiempo, aquí además de datos existen señales de control que indican al pipeline las funciones a cumplir. El camino a seguir está determinado por las mismas instrucciones (IBM 360/91).

Esto es equivalente a un cambio de **configuración**.

4.6.5. - **Pipelines estáticos vs. dinámicos**

Un pipeline estático puede tener una sola configuración funcional por vez. Los pipelines estáticos pueden ser unifuncionales o multifuncionales.

El pipelining es posible en los pipelines estáticos sólo si existen instrucciones del mismo tipo que se ejecutan una seguida de la otra. La función de un pipeline no debería cambiar frecuentemente, sino su performance sería muy baja.

Un procesador pipeline dinámico permite distintas configuraciones funcionales existiendo simultáneamente. Por lo tanto, un pipeline dinámico tiene que ser multifuncional. Y por otro lado, un pipeline unifuncional tiene que ser estático. La configuración dinámica requiere un mayor control que los estáticos y además mecanismos de secuenciamiento.

4.6.6. - **Pipelines escalares vs. vectoriales**

Dependiendo de los tipos de instrucción o datos, los procesadores pipeline pueden clasificarse en escalares o vectoriales. Un pipeline escalar procesa una secuencia de operandos escalares bajo el control de un ciclo (por ejemplo un DO de Fortran). Generalmente se hace prefetching de las instrucciones de un loop pequeño y se almacena en el buffer de instrucción.

Los operandos necesarios para instrucciones escalares repetidas se almacenan en una cache de datos para siempre tener el pipeline cargado con operandos. El IBM 360/91 es una máquina equipada con pipelines escalares (aunque no tiene una cache).

Los pipelines vectoriales están diseñados especialmente para manejar instrucciones vectoriales sobre operandos vectoriales. Los computadores con instrucciones vectoriales también suelen llamarse **procesadores vectoriales**. El manejo de operandos vectoriales está hecho por medio de firmware o hardware, en lugar del control de software que tienen los pipelines escalares. La ventaja fundamental es que se levanta y se decodifica la instrucción una sola vez por cada par de vectores, lo que ahorra gran cantidad de tiempo. Existen diversos pipelines vectoriales: el TI-ASC, STAR-100, Cyber-205, Cray-1, etc. Para mayor detalle remitirse al punto 4.11 del presente capítulo.

4.7. - PIPELINES GENERALES Y TABLAS DE RESERVACIÓN

El uso eficiente de un pipeline requiere :
- Flujo de datos permanente a través del mismo
- Controlar y organizar el flujo de datos evitando conflictos internos en las etapas del pipeline.

Si fuesen todos lineales **no** habría problemas y el ruteo sería trivial. Pero las etapas no son todas iguales en duración y además pueden ser reconfigurados los Pipes, luego es necesario establecer procedimientos para ruteo y control.

A pesar de estas dificultades pueden hallarse algoritmos de ruteo que requieren del siguiente entorno :

1.- El tiempo de ejecución de todas las etapas es un múltiplo de un reloj.

2.- Una vez que el proceso comenzó en el pipeline el diagrama temporal de utilización de las etapas por los distintos datos de entrada queda fijo.

Lo que hemos estudiado hasta ahora son pipelines lineales, sin conexiones feedback. Las entradas y salidas de estos pipelines son totalmente independientes. En algunos tipos de cálculos, las salidas de un pipeline se utilizan como futuras entradas. En otras palabras, las entradas pueden depender de salidas anteriores. En estos casos la historia de utilización del pipeline determina el estado actual del mismo. Estos pipelines pueden tener un flujo de datos no lineal.

Como ejemplo, en la figura 4.17 mostramos un pipeline que tiene conexiones feedback y feedforward.

Llamaremos S1, S2 y S3 a las etapas. Las conexiones entre etapas adyacentes forman la cascada original del pipeline. Una conexión feedforward conecta dos etapas Si y Sj con j mayor o igual a i+2; y una conexión feedback es aquella que conecta una etapa Si con Sj si j es menor o igual a i.

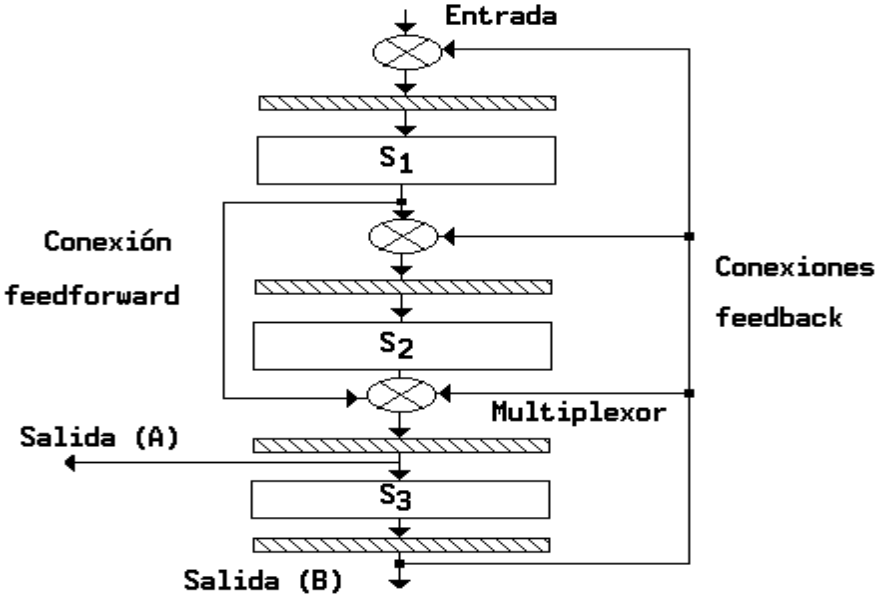


Fig. 4.17. - Un pipeline de ejemplo.

En este sentido, un pipeline lineal "puro" es un pipeline lineal sin conexiones feedback o feedforward.

La coordinación de las entradas feedback y forward es crucial ya que el uso indebido de tales conexiones puede destruir las ventajas inherentes del pipeline. En la práctica, muchos pipelines aritméticos permiten conexiones no lineales como un mecanismo para implementar recursividad y funciones múltiples.

Llamaremos a estos pipelines con conexiones feedforward o feedback, **pipelines generales**. Utilizaremos un gráfico bidimensional para mostrar cómo se utilizan las etapas sucesivas del pipeline en ciclos sucesivos. Llamaremos a este gráfico una **tabla de reserva o tabla de reservación**.

Las dos tablas de reserva que se muestran en la figura 4.18 corresponden a las dos funciones del pipeline del ejemplo 4.19.

Las filas corresponden a las etapas del pipeline, y las columnas a las unidades de tiempo de reloj. El total de unidades de reloj en la tabla se llama el **tiempo de evaluación** para la función dada. La tabla de reserva representa el flujo de datos a través del pipeline para la evaluación de una función dada.

Una entrada marcada en el cuadro (i,j) indica que la etapa Si se utilizará j unidades de tiempo luego de la iniciación de la evaluación de la función.

Tiempo

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
S_1	A			A			A	
S_2		A						A
S_3			A		A	A		

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
S_1	B				B		
S_2			B			B	
S_3		B		B			B

Fig. 4.18. - Tablas de reservación para dos funciones.

El flujo de datos en un pipeline estático y unifuncional puede ser descripto de forma completa con una tabla de reservación. Un pipeline multifuncional puede usar distintas tablas de reserva para funciones distintas. Por otro lado, una tabla de reserva dada no corresponde unívocamente a un pipeline particular. Puede haber distintos pipelines con distintas estructuras de interconexión utilizando la misma tabla de reserva.

Para visualizar el flujo de datos a través de las vías de datos en un pipeline para su evaluación completa, en la figura 4.19. se muestran los ocho pasos necesarios para evaluar la función A en el pipeline del ejemplo.

Esta función está determinada por la tabla de reservación para la función A dibujada precedentemente (Fig. 4.18).

Como explicamos anteriormente las tablas de reservación son válidas para la evaluación de una función particular. Además una tabla de reservación **no** corresponde a un pipeline determinado, sino que a más de uno.

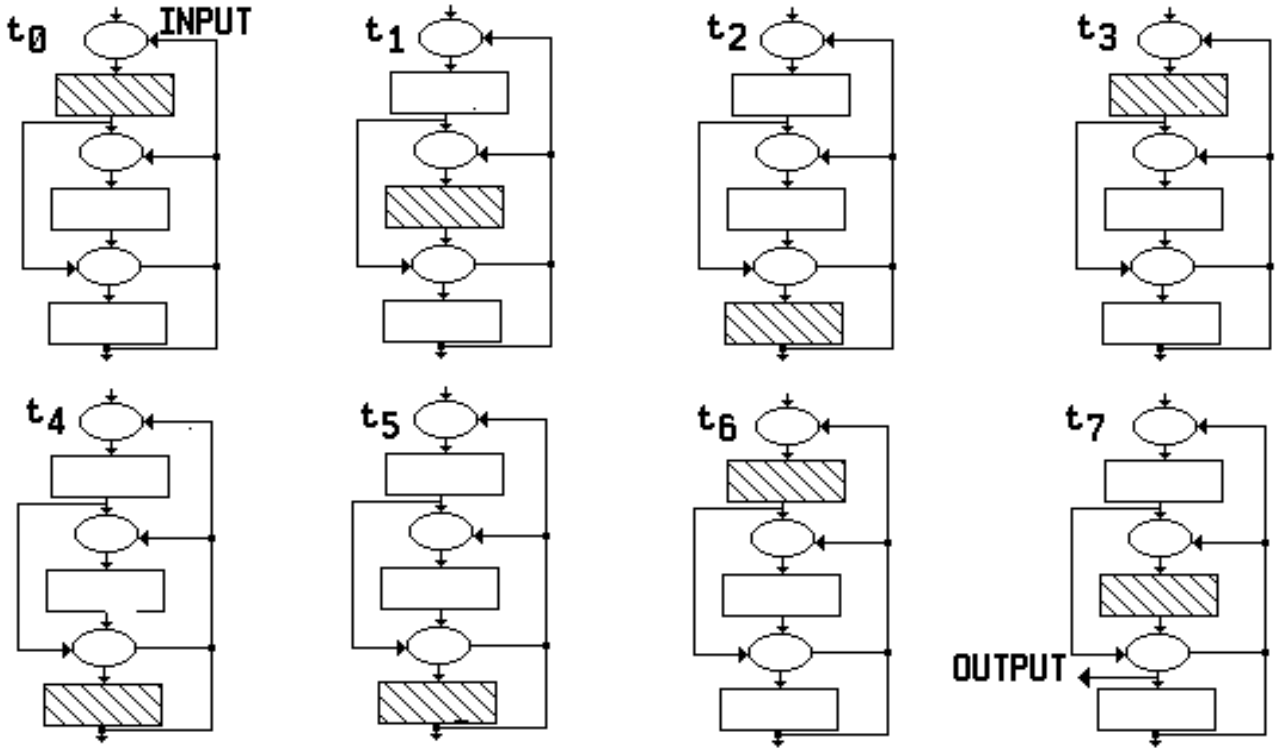


Fig. 4.19. - Ocho pasos utilizando el pipeline de ejemplo para evaluar la función A.

La idea de las Tablas de Reservación es evitar que dos o más datos intenten ingresar en forma simultánea a una misma etapa, lo que provocaría una **colisión**.

Supongamos que tenemos la tabla de reservación de la Fig. 4.20.

Para esta tabla de reservación se podrían tener los diseños de los pipelines de la Fig. 4.21.

En nuestro ejemplo, si se inician procesos en los instantes 0, 2 ó 6 se producirá una colisión.

	Tiempo	0	1	2	3	4	5	6	7
Etapa	1			A		A			
	2		A		A		A		
	3	A		A				A	

Fig. 4.20.

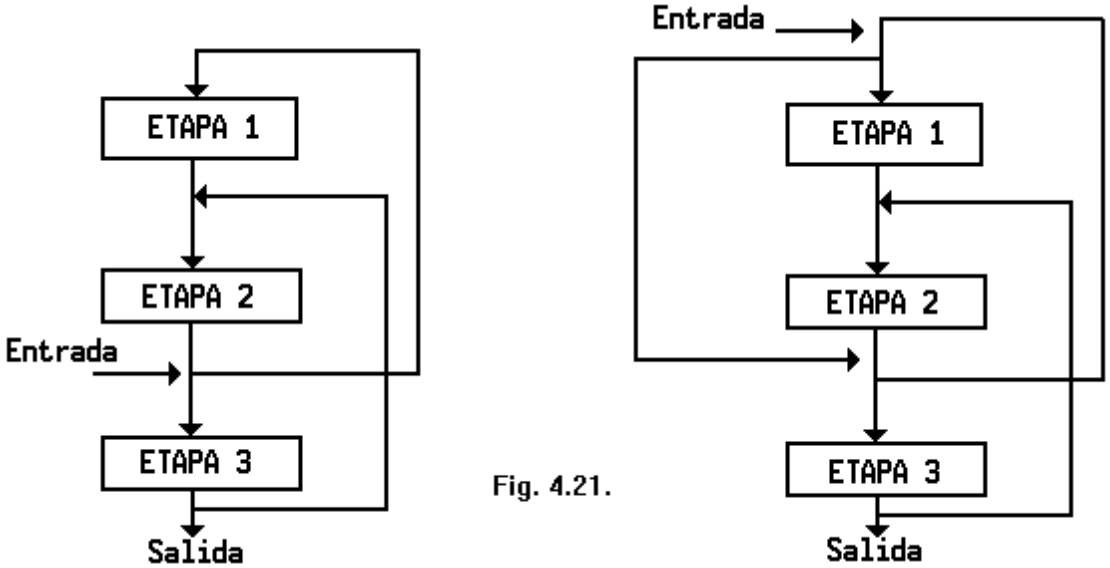


Fig. 4.21.

4.8. **Performance de pipelines.**

Un parámetro clave para la evaluación de la performance de un Pipeline es la "Latencia", que se define como el número de unidades de tiempo que separa dos inicios de la misma o distintas tablas de reservación.

En el caso de los Pipeline unifuncionales la latencia nunca podrá ser cero, ya que todos comienzan por la misma etapa (el menor posible es 1).

En el caso de los Pipeline dinámicos la latencia puede llegar a ser cero, ya que dos tablas de reservación distintas pueden superponerse.

Para lograr una mejor performance en un Pipeline es deseable obtener una "Latencia" promedio mínima.

Hay que tener en cuenta que elegir la mínima latencia entre dos inicios (estrategia **"greedy"**) no siempre llevará a la mejor performance.

Tomemos el ejemplo de la Fig. 4.22.

Tiempo		0	1	2	3	4	5	6	7
Etapa	1	B	B					B	B
	2			B		B			
	3				B		B		

Fig. 4.22.

Nótese que eligiendo la mínima latencia:

- B2 puede comenzar en el intervalo 3 (latencia 3)
- y B3 puede iniciarse en el intervalo 11 (con una latencia 8 respecto del inicio de B2).

Tomando la estrategia "greedy" veremos que nos da una Latencia alternada de 3 y 8 esto implica que :
(3 + 8) / 2 = 5,5 (latencia media 5,5) (Fig.4.23).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	B1	B1		B2	B2		B1	B1		B2	B2	B3	B3		B4	B4		B3	B3		B4	B4
2			B1		B1	B2		B2						B3		B3	B4		B4			
3				B1		B1	B2		B2						B3		B3	B4		B4		

Fig. 4.23.

Repetición del ciclo

Ahora observemos una estrategia que comience con "Latencia " 4, esto nos permite un nuevo inicio cada cuatro intervalos :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	B1	B1			B2	B2	B1	B1	B3	B3	B2	B2	B4	B4	B3	B3	B5	B5	
2			B1		B1		B2		B2		B3		B3		B4		B4		
3				B1		B1		B2		B2		B3		B3		B4		B4	

Fig. 4.24.

Repetición del ciclo

con lo cual llegamos a una mejor performance.

Visto el problema anterior nos queda cómo establecer una "latencia" óptima.

Para ello se construye un *Vector de Colisiones* (Colisión = 1) que se obtiene estudiando la Tabla de reserva de una determinada tarea.

Se intenta empezar la función en cada una de las columnas del intervalo. Si dentro del intervalo existe colisión se asigna 1 en el vector, caso contrario vale 0. Para nuestro ejemplo (Fig. 4.22) ese vector de colisiones resulta : < 1 1 1 0 0 1 1 1 >.

Ahora se debe simular el paso del tiempo sobre este vector de colisiones (haciendo shifts a izquierda) y comprobar que una configuración similar puede ser incluida en el pipeline sin que ocurran colisiones entre ellas. Para ello se construye un grafo, donde cada nodo es un tick de reloj (shift a izquierda) y se trata de comprobar cuando es posible incluir una tabla de reserva similar por medio de una operación OR con la configuración inicial del vector.

Con los OR se encuentran todos los vectores de colisiones resultantes, el algoritmo, en nuestro ejemplo, no hace más de 7 shifts.

Obviamente estas operaciones OR se darán solamente cuando el vector que se está corriendo comience con un cero (a izquierda); pues de otra manera, independientemente de lo que diga el resultado de la operación OR, habrá colisión.

Los vectores de colisiones derivados se utilizan para prevenir las colisiones en el cálculo de las funciones futuras respecto de las iniciadas previamente, en tanto que el vector de colisiones construido al principio sirve para prevenir colisiones en el cálculo de la función actual.

Contando los niveles de este grafo se obtiene cual es la latencia "mínima óptima promedio" (MAL - Minimum Average Latency).

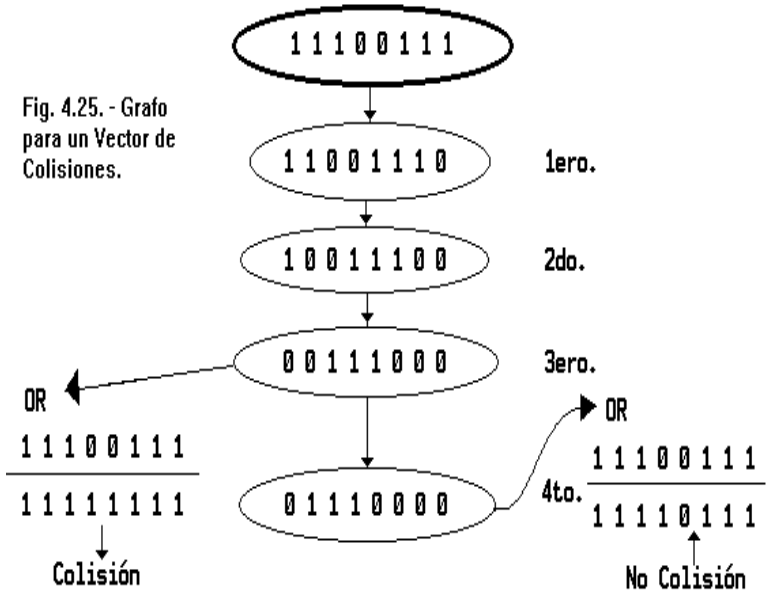


Fig. 4.25. - Grafo para un Vector de Colisiones.

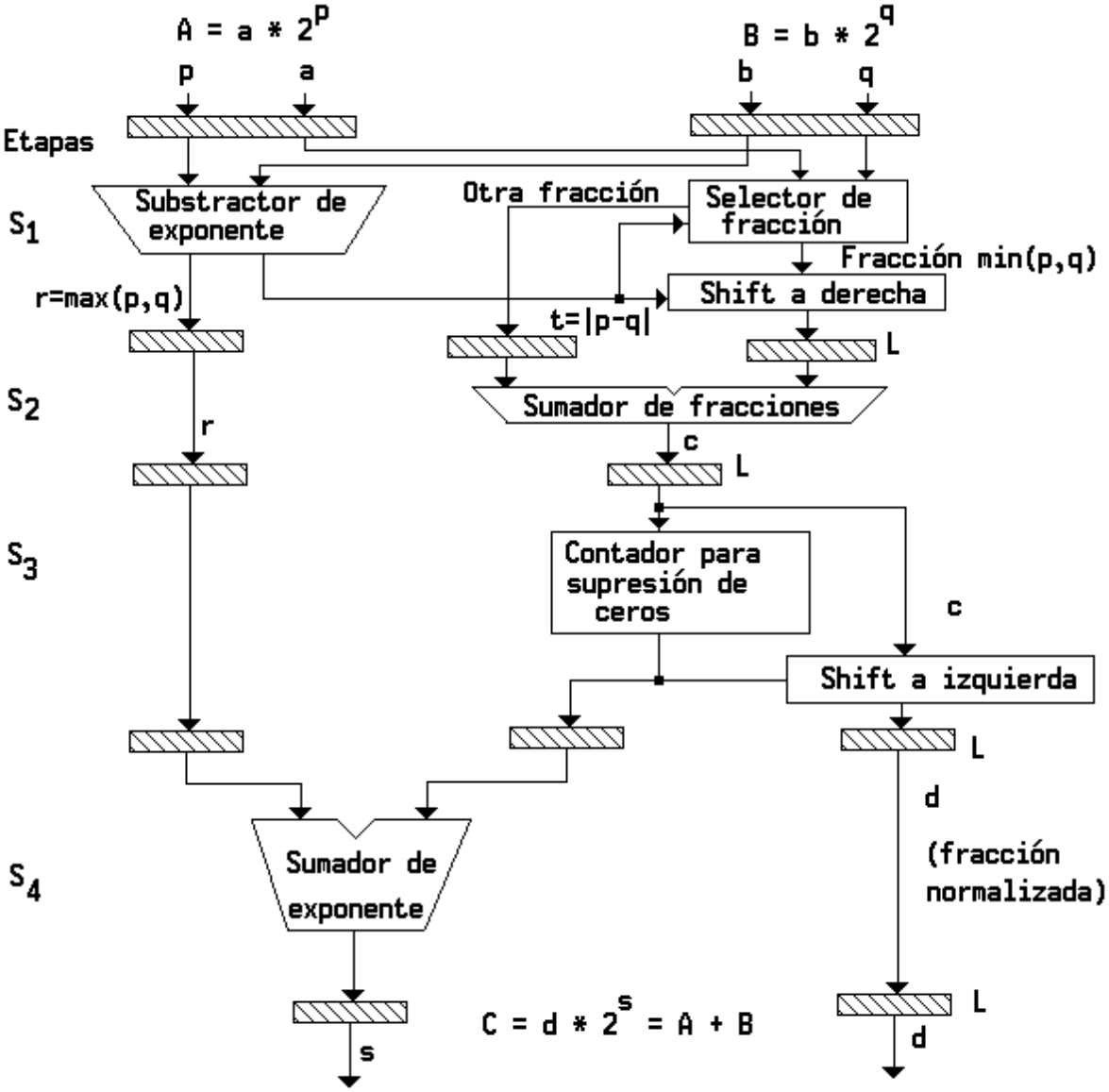


Fig. 4.26. - Un sumador pipeline de punto flotante de cuatro etapas de procesamiento.

Veamos una aplicación del método en nuestro ejemplo.
Como estamos en el 4to. nivel del grafo esto implica que la latencia media mínima la obtendremos introduciendo una nueva instrucción con latencia 4.

4.9. - Ejemplificación De Los Principios Operacionales De Los Pipelines**.**

4.9.1. - **Ejemplo 1:** Ilustraremos el diseño de un pipeline sumador de punto flotante, pipeline unifuncional. (Figura 4.26.)

Este pipeline puede construirse linealmente con cuatro etapas funcionales. Las entradas al pipeline son dos números normalizados en punto flotante:

$A = a * 2^{p}$ $B = b * 2^{q}$

Queremos calcular la suma:

$C = A + B = c * 2^{r} = d * 2^{s}$;

donde r = máximo(p,q), y d es c normalizado.

Las operaciones que se hacen en las cuatro etapas de pipeline son las siguientes:

1. Comparar los dos exponentes p y q para hallar el mayor exponente r = máximo (p,q) y determinar su diferencia t = p - q en módulo.
2. Hacer shift a la derecha de la fracción del menor exponente t bits para igualar los dos exponentes antes de la suma de fracciones.
3. Sumar ambas fracciones para producir la fracción intermedia c.
4. Contar el número de ceros a izquierda, digamos u, en la fracción c, y hacer shift u bits a la izquierda para producir la fracción normalizada d. Actualizar el exponente mayor s, haciendo s = r - u para producir el exponente de salida.

4.9.2. - **Ejemplo 2:**

La CPU de una computadora digital moderna puede dividirse generalmente en tres secciones, como vemos en la figura 4.27:

- la Unidad de Instrucciones,
- la Cola de Instrucciones y
- la Unidad de Ejecución.

Desde el punto de vista operacional, las tres unidades forman un pipeline. Los programas y datos residen en la memoria principal, que generalmente consiste de módulos de memoria interleaved. La memoria cache guarda copias de datos y programa listos para ejecución, y así acortamos la diferencia de velocidades entre Memoria y CPU.

La Unidad de Instrucciones consiste de etapas de pipeline que hacen

IF, levantar instrucciones de la memoria,
ID, decodificar los códigos de operandos de las instrucciones,
Cálculo de direcciones de operandos, y
OF , levantar los operandos desde la memoria, (si es necesario).

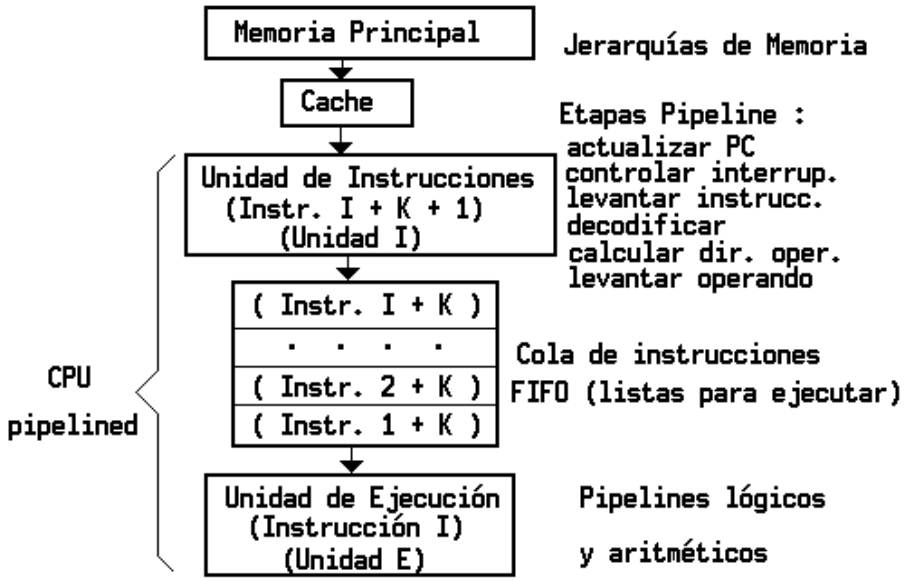


Fig. 4.27. - Estructura típica de una CPU pipelined.

La Cola de Instrucciones almacena instrucciones decodificadas y operandos que han sido levantados de memoria. La Unidad de Ejecución puede contener pipelines con varias funciones que hagan operaciones aritmético-lógicas. Mientras la Unidad de Instrucciones está levantando la instrucción I+K+1, la cola de instrucciones contiene las instrucciones I+1,...,I+K, y la unidad de ejecución ejecuta la instrucción I.

En este sentido, la CPU es un buen ejemplo de un pipeline lineal.

4.10. - **ALGUNOS PROBLEMAS DE LOS PIPELINES.**

En una máquina secuencial tradicional una instrucción **no** comienza hasta que se haya completado la anterior.

Si aplicamos Pipeline a un procesador de este tipo pueden aparecernos 3 clases de riesgos :
Read-after-Write (RAW)

Write-after-Read (WAR)
Write-after-Write (WAW)
Veamos ejemplos en el siguiente código de programa :

```
.....
.....
1) ALMACENAR  X
2) SUMAR   X
.....
3) ALMACENAR  X
.....
.....
4) ALMACENAR  X
.....
```

Riesgo RAW - Si 2) extrae después que 3) escribe
Riesgo WAR - Si 1) escribe después que 2) extraiga
Riesgo WAW - Si 3) escribe después que 4)

Luego si dadas las instrucciones i y j (j posterior a i), y definimos L(i) como el conjunto de todos los objetos leídos por i, y M(i) como el conjunto de todos los objetos modificados por i, resulta que existen condiciones de riesgo cuando se de alguna de las siguientes 3 condiciones :

$M(i) \cap L(j) \neq \phi$ (WAR)

$L(i) \cap M(j) \neq \phi$ (RAW)

$M(i) \cap M(j) \neq \phi$ (WAW)

Para detectar riesgos generalmente, se utiliza la etapa de búsqueda, y se comparan los conjuntos de objetos leídos y modificados por las instrucciones dentro del Pipe.

Para resolver situaciones de riesgo, generalmente, se detiene el Pipe hasta que se ejecute completamente la instrucción que provocaría el riesgo y luego continuar normalmente con las siguientes instrucciones.

Como veremos posteriormente estos riesgos también existen al dividir un programa en varias tareas concurrentes.

4.10.1. - **Prebúsqueda de Instrucciones**

En los ejemplos que hemos visto, hemos considerado que la carga de instrucciones se realiza mientras se está ejecutando otra.

De todas maneras esto es un motivo de riesgo, especialmente en las instrucciones de salto que debería tratarse de la manera que se puede visualizar en la Fig. 4.28.

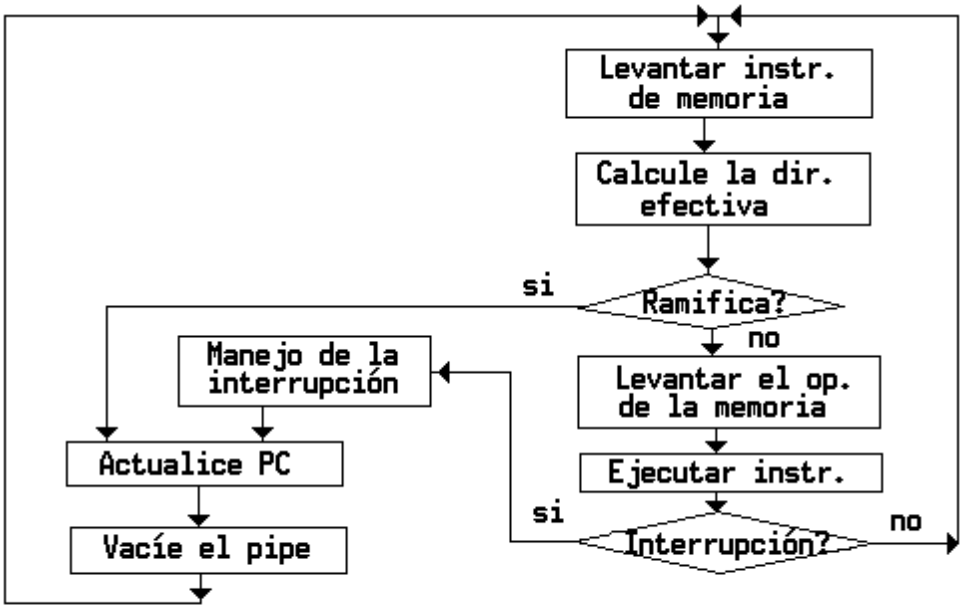


Fig. 4.28. - Efecto de las bifurcaciones en un pipeline de cuatro etapas.

Una forma de graficar esta situación sería considerar al procesador de la forma que se ve en la Fig. 4.29.

Donde la Unidad I realiza la decodificación y cálculo de direcciones y la Unidad E realiza la ejecución de la instrucción.

Si así lo dejamos prácticamente todas las instrucciones serían ejecutadas en la Unidad I y muy pocas en la Unidad E, por ejemplo las sumas.

Veamos de que manera es posible dividir (particionar) la Unidad I, en especial en el manejo de saltos, inclusive adjudicándole más funciones.

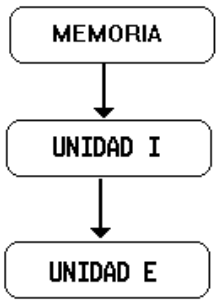


Fig. 4.29.

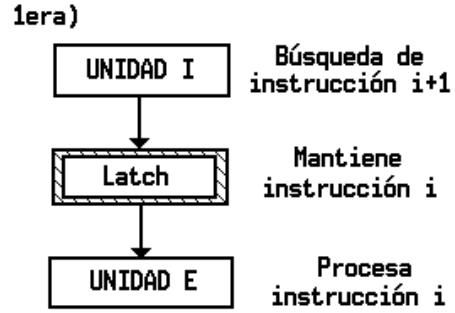


Fig. 4.30.

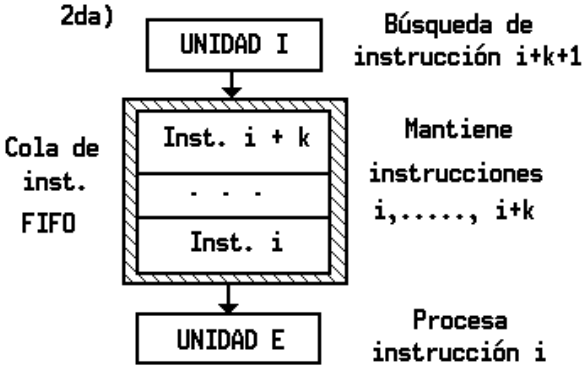


Fig. 4.31.

4.10.2. - **Manejo de saltos**

Las unidades I y E pueden estar básicamente conectadas de 2 formas. La primera se denomina **conexión rígida** (Fig. 4.30). Nótese que la segunda forma de conexión (Fig. 4.31) es mucho más flexible atendiendo que permite un mejor manejo por almacenar varias instrucciones simultáneamente.

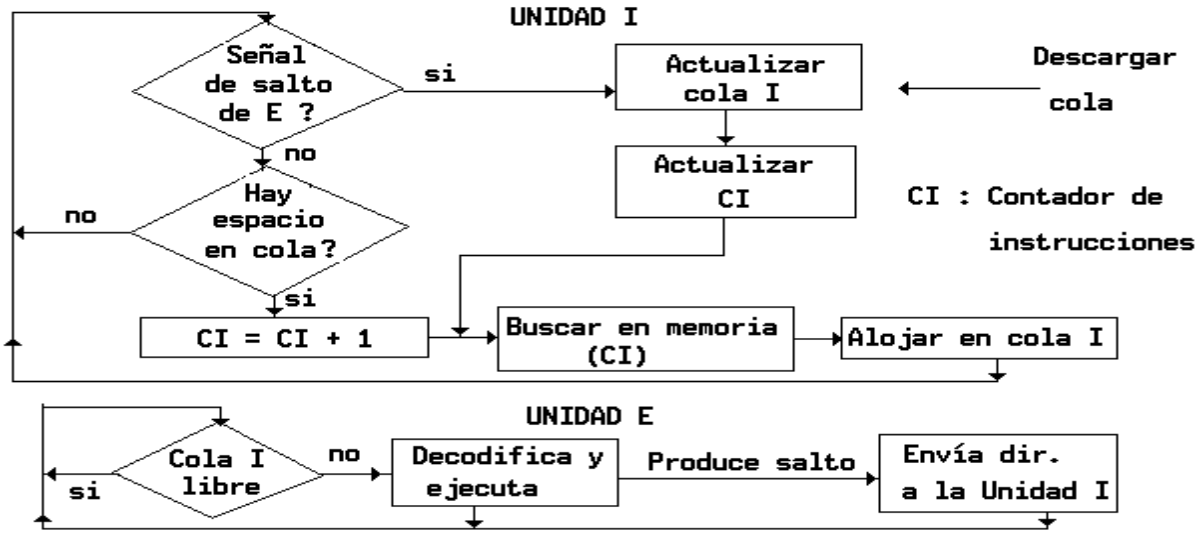


Fig. 4.32.

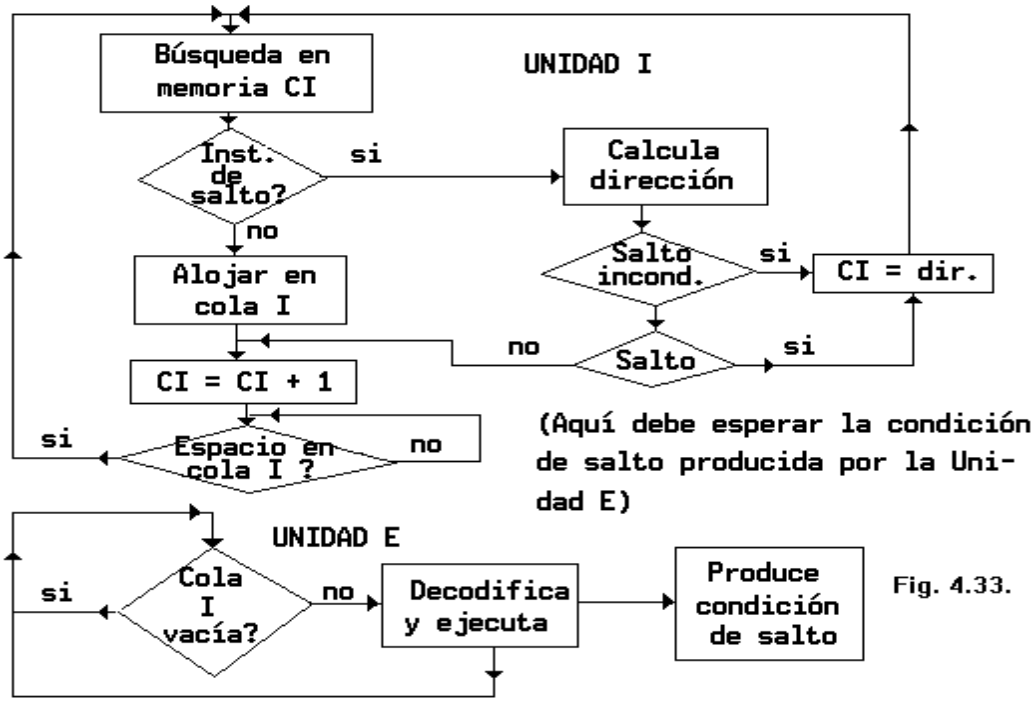


Fig. 4.33.

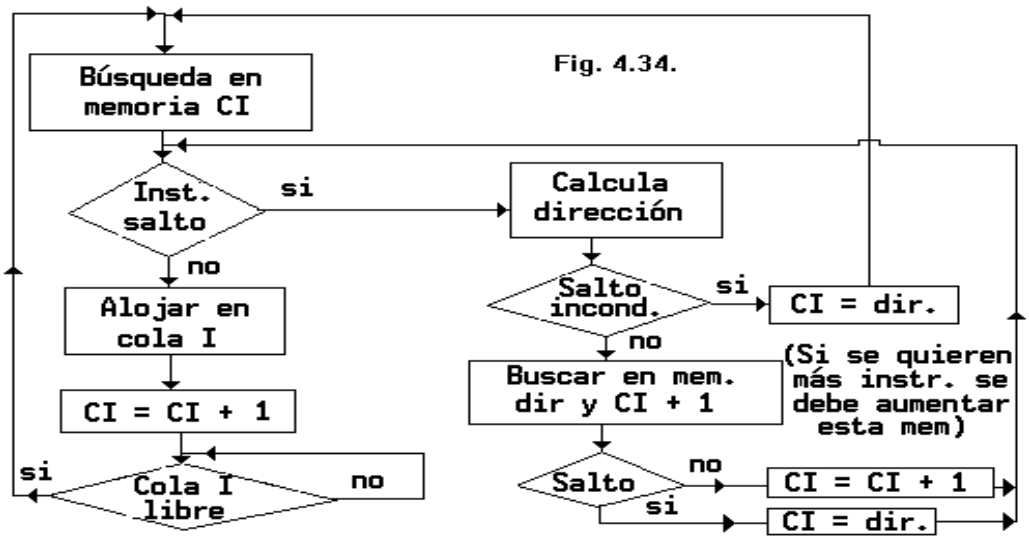
Veamos un diagrama lógico (Fig. 4.32) de cómo actúan ambas unidades, suponiendo que la decodificación la realiza la unidad E, utilizando cola FIFO de instrucciones.

Veamos ahora otro diagrama lógico (Fig. 4.33) de estas unidades permitiendo que la Unidad I decodifique en parte las instrucciones, para detectar saltos y lógicamente debe generar direcciones.

En el caso anterior el "descargar cola" no es necesario, luego queda resolver el problema de la espera de la Unidad I mientras se espera la condición producida por la Unidad E.

Una solución es realizar la prebúsqueda en ambas ramas posibles y descartar luego la que no cumple la condición.

Un diagrama posible de la **Unidad I** para este caso sería el que puede visualizarse en la Fig. 4.34.



Un ejemplo : el caso del Intel 8086 es un pipeline de 2 etapas, una de prebúsqueda de instrucciones y otra de ejecución, interconectadas por una cola de instrucciones.

Para determinar el tamaño de la cola de instrucciones se realizaron simulaciones que determinaron que el tamaño mayor de 6 bytes **no** producía mejoras, luego se eligió esta cifra que da un valor promedio de 2 instrucciones.

Hay que tener en cuenta que para que un procesador pipeline sea eficiente es necesario que pueda acceder a memoria toda vez que sea necesario, o sea que se necesita un apropiado ancho de banda de memoria.

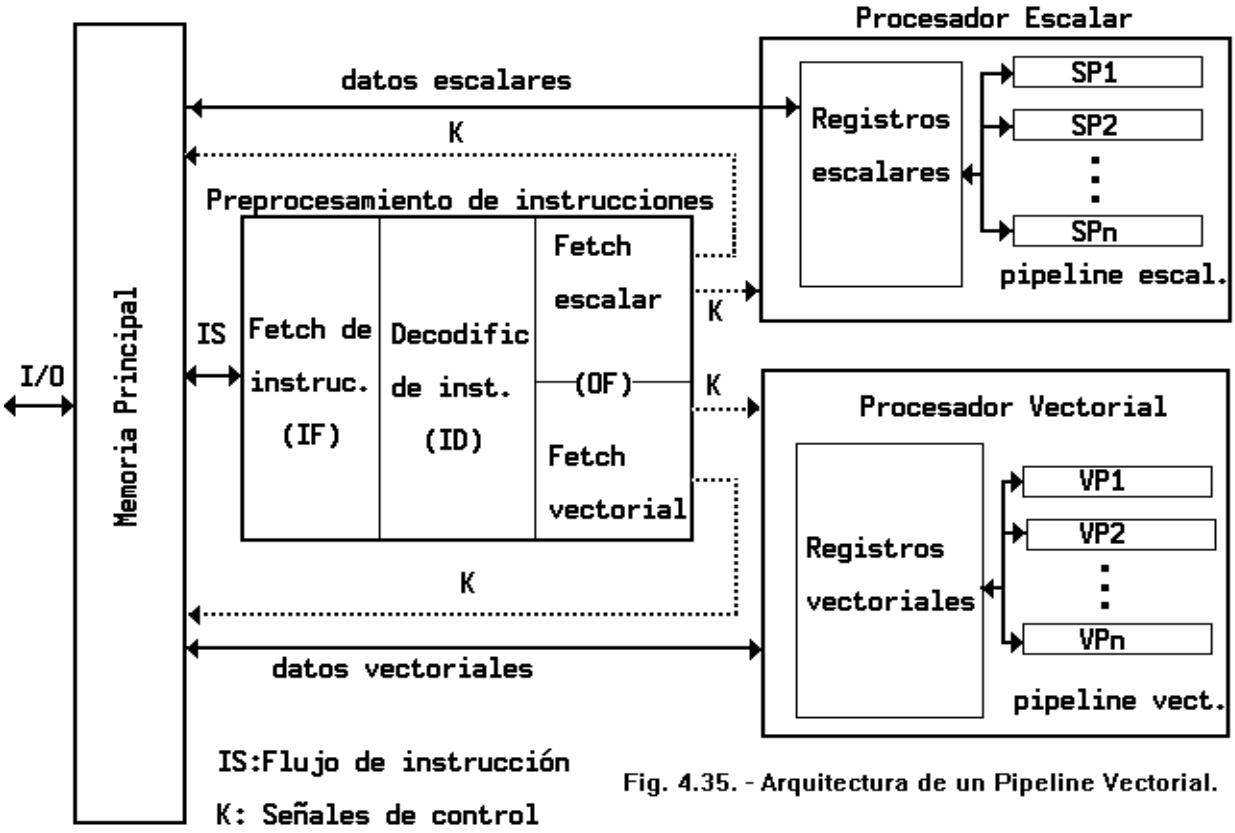


Fig. 4.35. - Arquitectura de un Pipeline Vectorial.

4.11. - **PIPELINES VECTORIALES.**

Existen aplicaciones que por su gran volumen de información a tratar, no es posible resolverlas por medio de procesadores secuenciales (escalares).

Una manera de resolver esto es dotar al procesador de instrucciones que puedan actuar sobre un gran conjunto de datos, esto se logra por el desarrollo de subrutinas adecuadas o incluyendo en el set de instrucciones del procesador instrucciones capaces de manejar ese volumen de datos. (instrucciones vectoriales que inducen obviamente a una ampliación del microcódigo).

Como ya fue explicado un procesador que maneja instrucciones vectoriales se denomina Procesador Vectorial, los pipelines vectoriales son procesadores vectoriales que tienen una organización interna que combina unidades funcionales múltiples operando en paralelo, junto con procesamiento pipeline dentro de las unidades funcionales.

La arquitectura básica de un procesador Vectorial sería la de la Fig. 4.35.

La funcionalidad de esta máquina puede esquematizarse como se ve en la Fig. 4.36 y opera de la siguiente forma:

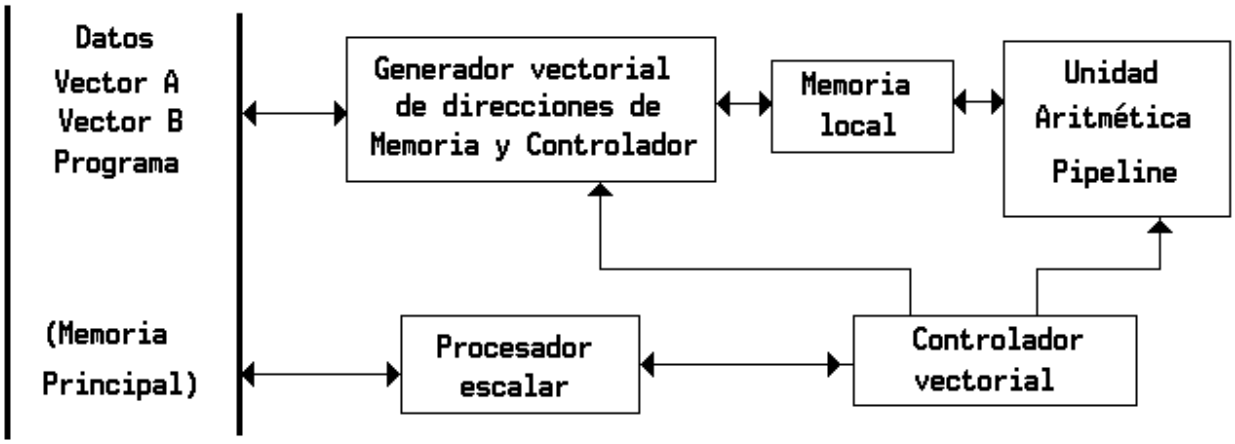


Fig. 4.36. - Funcionalidad de un pipeline vectorial.

El Procesador escalar va ejecutando las instrucciones escalares, hasta que se encuentra con una de tipo vectorial (de manejo de vectores), a partir de ese momento transfiere esa instrucción al controlador vectorial, el cual envía la información de descripción de los vectores (dirección, dimensiones, etc.) al Generador Vectorial y el código de operación a la Unidad Aritmética.

El Generador Vectorial toma la información de Memoria Principal y la coloca en la Memoria local para que esté a disposición de la Unidad Aritmética, la que ejecuta la instrucción deseada según el código de operación enviado por el Controlador Vectorial y es controlada por el mismo.

Nótese, en particular, que en la arquitectura del computador de la Fig. 4.35 la unidad Aritmética del procesador escalar también se encuentra pipelinizada y la unidad de preprocesamiento de las instrucciones también.

Como un ejemplo de una ejecución de instrucción vectorial consideraremos el siguiente ciclo DO:

```
        DO 10  I = 1,50
10      A(I) + B(I)
```

Esto define un circuito de programa que suma conjuntos de números. Para la mayoría de los computadores el lenguaje equivalente de máquina del ciclo DO anterior es una secuencia de instrucciones que leen un par de operandos de los vectores A y B y realizan una suma de punto flotante. Las variables de control del circuito son entonces actualizadas y los pasos repetidos 50 veces.

Un procesador vectorial elimina la cantidad de administración asociada al tiempo que lleva levantar y ejecutar las instrucciones en el circuito de programa. Esto se debe a que una instrucción vectorial incluye la dirección inicial de los operandos, la longitud de los vectores, la operación a realizar y los tipos de los operandos; todos en una instrucción compuesta. Dos operandos A(I) y B(I) son leídos simultáneamente de una memoria interleaved y se envían a un procesador pipeline que realiza la suma de punto flotante. Cuando se obtiene C(I) como salida del pipe se almacena en el tercer módulo interleaved mientras que las operaciones sucesivas se están realizando dentro del pipe.

4.11.1. - **Formato de instrucciones vectoriales**

El código de operación de una instrucción vectorial equivale a la **selección** de una **tabla de reservación** a utilizar por la unidad aritmética. Las tablas de reservación están implementadas por hardware o microcódigo.

Las instrucciones son suma vectorial, producto interno, producto vectorial, etc. Estas instrucciones deberán tener por lo menos :

- Código de operación
- Dirección de vectores
- Dimensiones de vectores
- Número de elementos de cada dimensión
- Tipo de dato de cada elemento
- Disposición de los elementos en memoria

4.11.2. - **Historia y un ejemplo**

La primer arquitectura de procesador vectorial se desarrolló en los años 60 y más tarde en los 70 para poder realizar directamente cálculos masivos sobre matrices o vectores.
Los procesadores vectoriales se caracterizan por múltiples unidades funcionales pipelinizadas que operan concurrentemente e implementan operaciones aritméticas y booleanas tanto escalares como matriciales.

Tales arquitecturas proveen procesamiento vectorial paralelo haciendo fluir secuencialmente los elementos del vector a través de una unidad funcional pipelinizada y ruteando los resultados de salida de una unidad hacia el input de otra unidad pipeline (proceso conocido como "encadenamiento" -chaining-).

Una arquitectura representativa puede tener una unidad de suma vectorial de seis etapas (Ver Fig. 4.37).

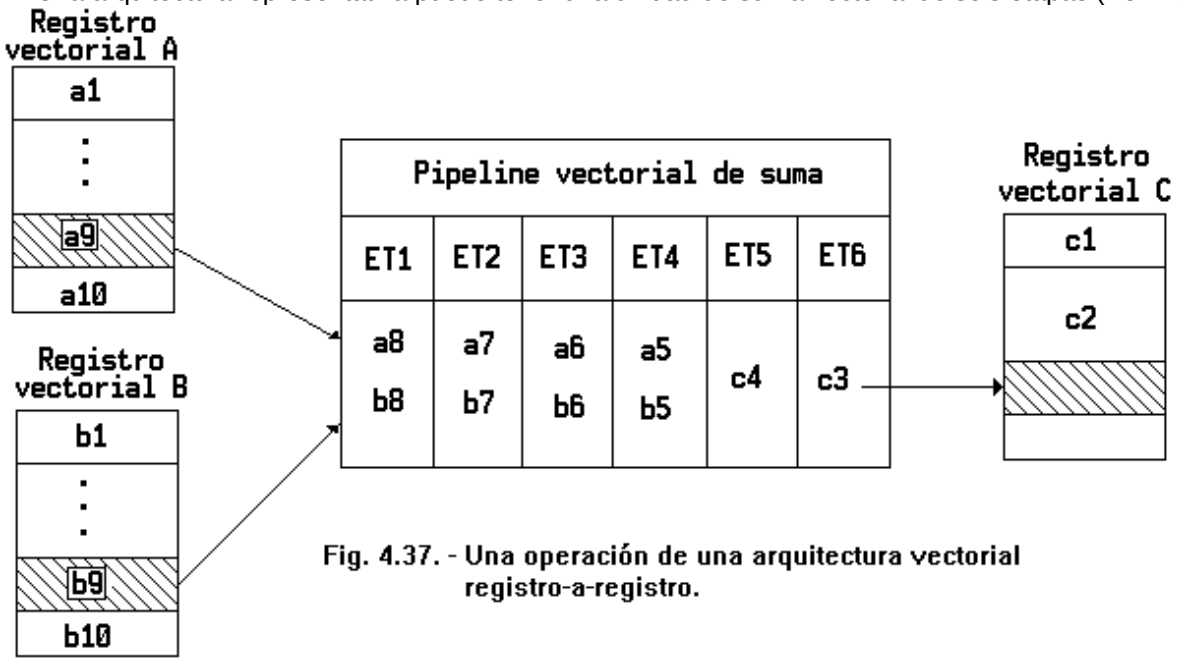


Fig. 4.37. - Una operación de una arquitectura vectorial registro-a-registro.

Si cada etapa del pipe de esta hipotética arquitectura tiene un ciclo temporal de 20 nanosegundos, entonces resultan 120 nanosegundos desde que los operandos **a1** y **b1** ingresan en la etapa 1 hasta que el resultado **c1** se halla disponible.

Cuando el pipe se llena tenemos sin embargo un resultado cada 20 nanosegundos. Luego, el tiempo de carga de las unidades vectoriales pipelinizadas tiene significativas implicancias en cuanto a performance.

En el caso de las arquitecturas registro-a-registro que se muestra en la figura, existen registros vectoriales de alta velocidad para los operandos y resultados.

Se obtiene una performance eficiente en tales arquitecturas (por ej. la Cray-1 y la Fujitsu VP-200) cuando la longitud de los elementos del vector es un múltiplo del tamaño de los registros vectoriales.

Las arquitecturas memoria-a-memoria como las de la Control Data Cyber 205 y la Advanced Scientific Computer de Texas Instruments (ASC) utilizan buffers especiales de memoria en lugar de registros vectoriales.

Las recientes supercomputadoras con procesamiento vectorial (tales como la Cray X-MP/4 y la ETA-10) reúnen entre 4 a 10 procesadores vectoriales mediante una gran memoria compartida.

EJERCICIOS

1) Cómo se logra el paralelismo en los sistemas dentro de la CPU ?
2) En un sistema con una única CPU y sin pipeline puede existir algún tipo de paralelismo ? En caso afirmativo describa un caso concreto.
3) Qué tipos de paralelismo (asincrónico, espacial, temporal) explotan las siguientes arquitecturas : Pipeline, Procesador Array y MIMD ? Justifique.
4) En cuántas etapas puede dividirse la ejecución de una instrucción ? Descríbalas **todas**.
5) Cómo es un pipeline de instrucción ? Grafíquelo.

6) Cuál es la diferencia entre un pipeline aritmético y un Pipeline de Instrucción ?
7) De qué depende la velocidad de un pipe ?
8) Qué es el tiempo de carga de un pipe ? Cómo influye en su rendimiento ?
9) Cuál es la utilidad de los latches ?
10) Defina específicamente :
- Frecuencia de un pipe
- Aceleración
- Eficiencia
- Troughput

11) Cuáles son las diferencias entre Pipeline y Solapamiento ?
12) Enumere la clasificación de Händler de los procesadores pipeline.
13) Diga si es verdadera o falsa la siguiente sentencia :
"Un procesador que cuenta con un coprocesador matemático para las operaciones de punto flotante es una forma de implementación de un pipeline aritmético". Justifique.
14) Cuál es la diferencia entre :
- Pipelines unifuncionales y Multifuncionales.
- Pipelines estáticos y Dinámicos.
- Pipelines escalares y Vectoriales.

15) Diga si son verdaderas o falsas las siguientes sentencias y justifique :
"Un pipe no puro tiene alguna conexión feedback o feedforward".
"Las conexiones feedback no permiten recursividad".
16) Qué es una tabla de reservación y para qué se utiliza ?
17) Construya por lo menos dos tablas de reservación para diferentes funciones que podría realizar el diseño de pipeline del ejemplo (Fig. 4.38).
a)- Construya un diagrama espacio-temporal que muestre la ejecución de ambas funciones, siguiendo la estrategia Greedy, de la secuencia : A A B A A B A A .....
Suponer que el pipeline es :
i) estático
ii) dinámico

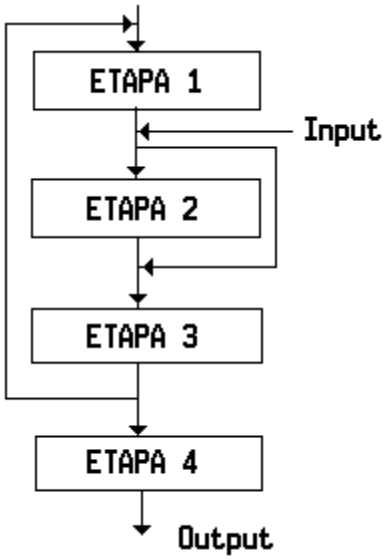


Fig. 4.38.

18) Construya una única tabla de reservación que muestre la ejecución de ambas funciones del ej.) 17) cuya latencia sea mínima. La latencia elegida resultó ser la latencia greedy ? Porqué ?
19)- Sobre la ejecución del siguiente conjunto de instrucciones :
(1) B = A + 1
(2) C = D + 1
(3) Z = B + H
explique cuáles riesgos se pueden suceder (RAW, WAR o WAW) y justifíquelo en base a las intersecciones de los conjuntos R(i) y D(i).
20)- Cuál es el problema que plantean las instrucciones de salto en estructuras pipeline ?
21)- Qué problema se puede suscitar si el ancho de banda de memoria es diferente del necesario para poner a disposición del pipe las instrucciones requeridas ? Discuta el caso en que sea mayor y el caso en que sea menor.
22) Cómo funciona un procesador con pipeline aritmético (vector processing) ?
23) Cuál es la información que se agrega a una instrucción escalar para transformarla en una instrucción vectorial ?
24) Cuáles son las condiciones para poder aplicar un pipe ?
25) El flujo de datos dentro de un pipe es discreto o continuo ?
26) Quién sincroniza el flujo de datos dentro de un pipe ?
27) Cómo funciona un procesador vectorial pipelinizado ?