TEMA 1: PROCESADORES SEGMENTADOS

1.3 Diferencias entre procesadores RISC y procesadores CISC.

PROCESADOR CISC: Complex Instruction Set Computer.

Los procesadores fueron dotados de conjuntos de instrucciones muy potentes que realizaban gran cantidad de operaciones internas.

Estas instrucciones se decodifican internamente en la unidad de control y se ejecutan mediante unas microinstrucciones almacenadas en una ROM interna, llamada memoria de control.

Para ello se requieren varios ciclos de reloj. Debido a la gran cantidad de operaciones internas que realizaban las instrucciones se empezaron a llamar CISC.

Simultáneamente se introducen los modos de direccionamiento para disminuir la cantidad de instrucciones a almacenar en memoria.

Está motivado porque el tiempo de acceso a memoria era mucho menor que el tiempo de procesamiento.

La arquitectura tipo CISC dificulta el paralelismo a nivel de instrucciones, para realizar la ejecución de una instrucción se debe de cargar desde memoria y traducirla a microoperaciones.

PROCESADOR RISC: Reduced Instruction Set Computer.

Para aumentar la velocidad de procesamiento se descubrió que con una determinada arquitectura de base la ejecución de programas compilados con instrucciones simples era más eficiente.

El avance en la fabricación de procesadores centró el objetivo de diseño en las arquitecturas RISC en la disminución de accesos a memoria mediante un aumento de la cantidad de registros internos en el procesador. El hardware de control es más sencillo debido al conjunto de instrucciones simplificado.

La decodificación de instrucciones puede ser directamente implementada por hardware dentro de la CPU, disminuyendo la lógica de control y obteniendo una mayor velocidad de ejecución.

La alternativa RISC permite la ejecución segmentada de instrucciones, pipeline, y el diseño de procesadores cableados.

Ventajas de un diseño RICS frente a un CISC.

Tienen una mayor velocidad de ejecución.

Reduce el tamaño de la CPU, por lo que se dispone de más espacio para recursos, mayor cantidad de ***registros*** o memoria caché. En arquitectura de ordenadores, un ***registro*** es una memoria de alta velocidad y poca capacidad, integrada en el microprocesador, que permite guardar transitoriamente y acceder a valores muy usados, generalmente en operaciones matemáticas.

Al ser la lógica de control más simple, se facilita el diseño.

Permite máquinas más compactas y con menor consumo al reducirse el tamaño de la CPU.

Posibilita la segmentación y el paralelismo en la ejecución de instrucciones.

Reduce los accesos a memoria debido a que los operandos se cargan en registros.

El rendimiento de la implementación pipeline en un procesador RISC depende directamente de los riesgos que se generan durante la ejecución de dos o más instrucciones que entran en conflicto.

Para minimizar los riesgos se plantea una relación muy estrecha entre los compiladores y la arquitectura. Las operaciones complejas que aparecen en el código fuente se descomponen en multitud de operaciones sencillas RISC.

El objetivo es mantener el hardware tan simple como podamos a base de hacer más complejo el compilador.

La ganancia de velocidad se debe a que predominan las instrucciones más frecuentemente utilizadas, mientras que las menos frecuentes se descomponen en operaciones simples.

La cantidad de instrucciones en un procesador RISC es mayor que en un CISC.

Un factor importante de los procesadores RISC es la compatibilidad con el software preexistente.

Tienen filosofías de diseño opuestas.

A igual tecnología y frecuencia de reloj, un procesador RISC tiene mayor capacidad de procesamiento, con una estructura de hardware más simple, teniendo como beneficio disminución de potencia y costes.

El aumento de rendimiento de un RISC se sustenta en el diseño de un compilador eficiente.

1.4 Clasificación y características generales de las arquitecturas paralelas.

El paralelismo se implementa siguiendo dos líneas:

Replicación de elementos: incluye unidades funcionales, procesadores, módulos de memoria entre los que se distribuye el trabajo.

A nivel de sistema están los multiprocesadores y los canales/procesadores de E/S.

A nivel de procesador se tiene el uso de varias unidades funcionales en los procesadores superescalares, los procesadores VLIW y los procesadores vectoriales.

Segmentación (*pipelining*): es una técnica en la que un elemento se divide en una serie de etapas que funcionan de forma independiente y por las que van pasando los operandos y las instrucciones que procesa cada elemento. De esta forma dicho elemento puede realizar simultáneamente operaciones distintas en las diferentes etapas en que se encuentra dividido.

Taxonomía de Flynn.

Computadores SISD (Single Intruction Single Data): Un único flujo de instrucciones procesa operandos y genera resultados definiendo un único flujo de datos. Computadores monoprocesador.

Computadores SIMD (Single Intruction Multiple Data): Un único flujo de instrucciones procesa operandos y genera resultados definiendo varios flujos de datos. Computadores matriciales y vectoriales.

Computadores MIMD (Multiple Intruction Multiple Data): El computador ejecuta varias secuencias o flujos distintos de instrucciones y cada uno de ellos procesa operandos y genera resultados de forma que existen también varios flujos, uno por cada flujo de instrucciones. Computadores multiprocesadores y multicomputadores.

Computadores MISD (Multiple Intruction Single Data): Se ejecutan varios flujos de instrucciones aunque todos actúan sobre el mismo flujo de datos.

Tipos de paralelismo según la Taxonomía de Flynn.

Paralelismo de datos.

Se utiliza cuando una misma función, instrucción, etc… se ejecuta repetidas veces en paralelo sobre datos diferentes.

Se utiliza en máquinas de la clase SIMD y MIMD.

Paralelismo funcional.

Se aprovecha cuando las funciones, bloques, instrucciones, etc… que intervienen se ejecutan en paralelo.

Existen distintos niveles de paralelismo funcional.

- **Nivel de instrucciones u operaciones**. (ILP), las instrucciones de un programa se ejecutan en paralelo, es de bajo nivel, se explota por el hardware de forma transparente o por el compilador. Es el nivel de granularidad más fina en la arquitectura de computadores la granularidad es la cantidad de trabajo asociado a cada tipo de tarea candidata a la paralelización.

- **Nivel de bucle**. Se ejecutan en paralelo distintas iteraciones de un bucle o secuencias de instrucciones de un programa, la granularidad es fina-media.

- **Nivel de funciones**. Los distintos procedimientos que constituyen un programa se ejecutan simultáneamente, la granularidad es media.

- **Nivel de programas**. Se ejecutan en paralelo programas diferentes que pueden ser o no de una misma aplicación, la granularidad es gruesa.

1.5 Medidas para evaluar el rendimiento de un computador.

Las medidas más utilizadas para el rendimiento de un computador son:

- **Tiempo de respuesta**: tiempo que tarda el computador en procesar una entrada.
- **Productividad**: número de instrucciones procesadas por unidad de tiempo.
- **Funcionalidad**: tipos de entradas diferentes que es capaz de procesar.
- **Expansibilidad**: posibilidad de ampliar la capacidad de procesamiento añadiendo bloques a la arquitectura existente.
- **Escalabilidad**: posibilidad de ampliar el sistema sin que esto suponga una devaluación de las prestaciones.
- **Eficiencia**: relación entre el rendimiento obtenido y el coste que ha supuesto conseguirlo. (eficiencia = rendimiento/coste).

$$T_{CPU} = NI \cdot CPI \cdot T_{ciclo} = NI \cdot \left(\frac{CPI}{f} \right)$$

TCPU : tiempo de CPU de un programa o tarea.
NI : número de instrucciones máquina del programa que se ejecutan.
CPI : número medio de ciclos por instrucción.
Tciclo : periodo del reloj del procesador.

$$CPI = \frac{Ciclos\ de\ reloj\ del\ \ programa}{NI} = \frac{\sum_{i=1}^{n} NI_i \cdot CPI_i}{NI}$$

CPIi :número medio de ciclos de las instrucciones de tipo i.
NIi : número de instrucciones de ese tipo.

$$T_{CPU} = NI \cdot \left(\frac{CPE}{IPE} \right) \cdot T_{ciclo}$$

CPE : número medio de ciclos entre inicios de ejecución de instrucciones.
IPE : número medio de instrucciones que se emiten.

$$T_{CPU} = \left(\frac{N_{operaciones}}{Op_{instrución}} \right) \cdot CPI \cdot T_{ciclo}$$

$$MIPS = \frac{NI}{T_{CPU} \cdot 10^6} = \frac{f}{CPI \cdot 10^6}$$

MIPS : Millones de instrucciones por segundo.

$$MFLOPS = \frac{Operaciones\ en\ coma\ \ flotante}{T_{CPU}}$$

$$S_p = \frac{rendimiento(p)}{rendimiento\_original} = \frac{T_{CPU\_original}}{T_{CPU\_mejorada}}$$

$$S_p \leq \frac{p}{1 + f(p-1)}$$ ley de Amdahl

$$T_p \geq f \cdot T_1 + (1-f)\frac{T_1}{p}$$

1.6 Características de los procesadores segmentados.

La segmentación (***pipelining***) es una técnica empleada en el diseño de procesadores que trata de explotar el paralelismo intrínseco que existe entre las instrucciones de un flujo secuencial.

Mediante la segmentación se puede solapar la ejecución de múltiples instrucciones. Cada etapa de la segmentación completa una parte de la tarea total. La salida de un segmento es la entrada del siguiente.

La segmentación puede procesar las subtareas de forma simultánea, aunque sea sobre diferentes tareas, ***lo más importante*** es la posibilidad de comenzar una nueva tarea sin que la anterior haya terminado.

La ***medida de eficacia de un procesador*** es el tiempo máximo que pasa entre la finalización de dos tareas consecutivas.

La velocidad de emisión de tareas es el ritmo al que salen las tareas del procesador.

La profundidad de segmentación es el número de n etapas en las que puede dividirse el procesamiento de una instrucción.

Para que el tiempo de latencia del procesador segmentado sea el mínimo posible es necesario que el procesador esté equilibrado.

El procesador está equilibrado si todas las subtareas en que se haya dividido la tarea total tarden en procesarse el mismo tiempo. Las tareas no pueden avanzar a la etapa siguiente hasta que no se haya terminado la subtarea más lenta, en el caso que el procesador no esté equilibrado las etapas más rápidas estarán un tiempo sin hacer trabajo, disminuyendo el rendimiento total del procesador.

La relación de precedencia de un conjunto de subtareas $T_1,\ldots,T_n$ que componen cierta área T, especifica para cada subtarea T_j que no puede comenzarse hasta que hayan terminado ciertas subtareas T_i.

Las relaciones de precedencia para todas las subtareas de T forman su grafo de precedencia.

1.7 Arquitectura segmentada genérica.

La arquitectura segmentada genérica ASG, es de tipo de registros de propósito general donde los operandos se referencian explícitamente.

La ventaja de estas máquinas surgen del uso efectivo de los registros por parte del compilador al calcular expresiones aritmético lógicas y al almacenar datos.

Los registros permiten una gestión más flexible de los datos, además se reduce el tráfico de memoria, se acelera la ejecución del programa y se mejora la densidad de código.

Dos características importantes del repertorio de instrucciones que clasifican las arquitecturas de propósito general:

- El número de operandos que puede tener las instrucciones aritmético-lógicas.
- El número de operandos que se pueden direccionar en memoria en las instrucciones aritmético-lógicas.

Las instrucciones aritmético-lógicas de la ASG utilizan en total tres operandos y ninguno de ellos se referencia a memoria.

Las máquinas en la que los operandos no se referencian en memoria se les denominan máquinas registro-registro o máquinas de carga/almacenamiento.

En la ASG el modo de direccionamiento es con desplazamiento y la Dirección Efectiva se calcula sumando al contenido de un registro el Operando declarado en la instrucción, que es un desplazamiento respecto al contenido del registro.

1.7.1 Repertorio de instrucciones de la ASG.

La ASG tiene un total de 32 registros de 32 bits identificados desde R0 a R31.
Cada registro puede contener un valor entero a excepción de R0 que siempre contiene el valor 0.
Los registros de coma flotante de 64 bits se identifican como F0,… F30
La longitud de las instrucciones en procesadores segmentados de la ASG es de 32 bits.

Los cuatro tipos básicos de operaciones son:

- Aritméticas y lógicas
- Transferencia de datos.
- Bifurcaciones o saltos incondicionales.
- Saltos condicionales.

Las instrucciones de la ALU son operaciones aritméticas sencillas (suma, resta, multiplicación, división y comparación)y lógicas (AND, OR, XOR).

En las instrucciones de comparación, si la condición es cierta se almacena un 1 en el bit menos significativo del registro de destino y los restantes se fijan a 0, en caso contrario se coloca un 0 en todos los bits.

Las operaciones de transferencia de datos entre los registros y la memoria se realizan exclusivamente a través de instrucciones de carga (LD) y almacenamiento (SD).

En una instrucción de carga se almacena el contenido de una dirección de memoria en un registro y al contrario en una instrucción de almacenamiento.

LD Rd, desplazamiento (Rf).

Rd: registro destino donde se almacena el dato leído de memoria.
desplazamiento(Rf): dirección de memoria a la que acceder para recuperar el dato, Rf es el registro base.

SD desplazamiento (Rd), Rf.

Rf: registro cuyo contenido se va a escribir en memoria.
desplazamiento(Rd): dirección de memoria donde se va a escribir, Rd es el registro base.

El valor del desplazamiento y el valor del registro base serán siempre valores enteros positivos o negativos.

Si utilizamos registros en coma flotante las expresiones son:

LD Fd, desplazamiento (Rf).
SD desplazamiento (Rd), Ff.

Una carga o un almacenamiento sencillo se lee o escribe una palabra de 4 bytes.
Una carga o un almacenamiento en coma flotante se lee o escribe una palabra de 8 bytes.

Instrucción Ejemplo	Nombre	Significado
LD R1, 4(R2)	Carga	R1 ← M[R2 + 4]
SD 0(R3), R5	Almacenamiento	M[R3 + 0] ← R5
LD F6, 5(R3)	Carga flotante	F6 ← M[R3 + 5]
SD 6(R1), F4	Almacenamiento flotante	M[R1 + 6] ← F4

La modificación del flujo de control se realiza mediante las instrucciones de bifurcación y salto.

La bifurcación ocurre cuando el cambio en el flujo de control sea incondicional y de salto cuando sea condicional, es decir cuando establece una condición (R5>0).

En los saltos la condición que determina el resultado del salto se especifica en el código de operación de la instrucción que examina el registro fuente para compararlo con cero, distinto de cero, mayor de cero o menor de cero.

La forma más común de especificar el destino del salto es suministrar un desplazamiento que se sume al contador del programa, este tipo de saltos son los saltos relativos.

Un salto es efectivo cuando la condición probada por la instrucción de salto es verdadera y la siguiente instrucción que se vaya a ejecutar es el destino del salto.

Las bifurcaciones son siempre efectivas.

Instrucción Ejemplo	Nombre	Significado
JUMP etiqueta	Bifurcación	PC ← etiqueta
JUMP R4	Bifurcación a registro	PC ← R4
BEQZ R3, etiqueta	Salto igual a cero	If (R3 = 0) PC ← PC + etiqueta
BNEZ R2, etiqueta	Salto distinto de cero	If (R2 ≠ 0) PC ← PC + etiqueta
BGT R5, etiqueta	Salto mayor que cero	If (R5 > 0) PC ← PC + etiqueta
BLT R7, etiqueta	Salto menor que cero	If (R7 < 0) PC ← PC + etiqueta

1.7.2 Implementación de la segmentación de instrucciones en la ASG.

En un procesador segmentado el cálculo a segmentar es el trabajo que es necesario realizar en cada ciclo de instrucción, que es el número de ciclos de reloj que consume su procesamiento.

En la ASG ***un ciclo de una instrucción se descompone en cinco etapas*** básicas:

1. IF (Instruction Fetch): lectura de la instrucción de la caché de instrucciones. La caché de instrucciones puede admitir la lectura de una instrucción en cada ciclo de máquina y un fallo en la caché detiene la segmentación
2. ID (Intruction Decoding): decodificación de la instrucción y lectura de sus operandos del fichero de registros.
3. EX (Execution): ejecución de las operaciones si se trata de una instrucción aritmético-lógica y del cálculo de la condición y de la dirección de salto si se trata de una bifurcación o salto condicional.
4. MEM (Memory Access): Acceso a la caché de datos para lecturas (cargas) o escrituras (almacenamientos).
5. WB (Write-Back result): Escritura del resultado en el fichero de registros. Las instrucciones de almacenamiento y salto no realizan ninguna acción en esta etapa ya que no necesitan escribir en el fichero de registros y se encuentran liberadas.

En cada ciclo de máquina el fichero de registros debe admitir dos lecturas en la etapa ID y una escritura en la etapa WB.

La ventaja de la segmentación es la posibilidad de empezar a ejecutar una nueva instrucción en cada ciclo de reloj.

El tiempo total de la instrucción segmentada es ligeramente superior al de su equivalente no segmentada debido al tiempo que se consume en el control de la segmentación.

Este tiempo viene determinado por varios factores:

- Los cerrojos o buffers de contención con el objeto de aislar la información entre etapas.
- La duración de todas las etapas viene determinada por la duración de la etapa más lenta.
- Los riesgos que se producen en la segmentación y producen detenciones.

1.8 Riesgos en la segmentación.

Los riesgos de la segmentación son consecuencia tanto de la organización como de las dependencias entre las instrucciones.

Un ***riesgo*** es la situación que impide a una instrucción acceder a la ejecución de sus etapas al depender de otra anterior.

Los riesgos provocan una parada en el flujo de las instrucciones dentro de las etapas de la segmentación hasta que la dependencia se resuelva.

1.8.1 Riesgos estructurales.

Surgen por conflictos en los recursos, debido a que el hardware que necesita una instrucción está siendo utilizado por otra.

Otras situaciones en las que pueden aparecer riesgos estructurales son:

- No todas las etapas de la segmentación tienen la misma duración.
- Hay instrucciones más complejas que otras.

Para evitar las detenciones en las etapas de segmentación podemos plantear dos soluciones:

- Duplicar la unidad funcional aritmética para poder solapar dos etapas de dos instrucciones en el mismo ciclo de reloj.
- Separar las instrucciones si se puede, se conoce como ***planificación de código***.

1.8.2 Riesgos por dependencia de datos.

Se produce cuando una instrucción necesita los resultados de otra anterior por no haberse terminado de ejecutar.

Los riesgos por dependencia de datos se clasifican en función del orden de los accesos de escritura y lectura de las instrucciones en tres tipos, consideraremos las instrucciones *i* ocurre primero y *j* después.

- Riesgo tipo **RAW** (Read After Write): también conocido como ***dependencia verdadera***, se produce cuando la instrucción j intenta leer un dato antes de que la instrucción *i* lo escriba.
- Riesgo tipo **WAR** (Write After Read): también conocido como ***antidependencia o dependencia falsa***, se produce cuando la instrucción *j* intenta escribir en su destino antes de que sea leído por la instrucción *i*.
- Riesgo tipo **WAW** (Write After Write): también conocido como ***dependencia de salida o dependencia falsa***, se produce cuando la instrucción *j* intenta escribir un operando antes de que sea escrito por la instrucción *i*.

Hay que tener en cuenta que el caso **RAR** (Read After Read) no es un riesgo, ya que cuando dos instrucciones necesitan leer desde el mismo registro, lo hacen sin problemas en la etapa de decodificación de la instrucción.

En la ASG sólo se puede presentar el riesgo RAW.

El riesgo WAR no se puede presentar ya que todas las lecturas de registro se realizan en la etapa ID y todas las escrituras de registro tienen lugar en la etapa WB, siendo la ID anterior a la WB.

El riesgo WAW se presenta en segmentaciones que escriben en más de una etapa. La ASG evita este riesgo ya que solo escribe un registro en WB.

Riesgos por dependencia de datos en registros en el caso de instrucciones aritmético-lógicas.
Riesgos por dependencia de datos en memoria con instrucciones de carga y almacenamiento.

Las alternativas más importantes ***para evitar los problemas de los riesgos RAW*** son:

1.8.2.1 La reorganización de código.

Consiste en disponer las instrucciones en el programa de forma que entre las instrucciones con dependencias tipo RAW existan instrucciones que permitan retrasar la segunda instrucción con respecto a la primera, de esta forma la primera tendrá tiempo a escribir su resultado antes que la segunda la lea.

Debemos de mantener la semántica original del programa, el orden original de las lecturas y escrituras de los registros y la memoria.

En el caso de que el compilador no pueda reorganizar el código se deben insertar instrucciones NOP, entre las instrucciones que tengan dependencia de datos. Una ventaja es que no hace falta un hardware adicional, pero se necesita un compilador más complejo y pérdida de tiempo.

1.8.2.2 El interbloqueo entre etapas.

Se introducen elementos hardware en el cauce para detectar la existencia de dependencias, en el caso que detecte una la instrucción se detiene el número de ciclos necesarios. El programa finaliza correctamente pero seguimos perdiendo ciclos lo que nos lleva a una disminución del rendimiento.

1.8.2.3 El adelantamiento (Caminos de bypass o forwarding).

Con esta técnica aprovechamos los elementos de la técnica de interbloqueo y la utilizamos para habilitar una serie de buses para permitir que los resultados de una etapa pasen como entradas a la etapa donde son necesarios en caso de dependencias RAW, a la vez que prosigue su camino para almacenarse en el fichero de registros.

La función que tiene la lógica de bypass es comprobar si hay coincidencia entre el identificador del registro de destino de la instrucción que acaba su etapa EX y los identificadores de los registros fuente de los operandos de la instrucción siguiente que va a iniciar su etapa EX.

1.8.3 Riesgos de control.

Se producen a partir de las instrucciones de control de flujo, saltos y bifurcaciones, ya que no podemos leer la instrucción siguiente hasta que no se conozca su dirección.

Cuando se ejecuta un salto condicional el valor del contador del programa puede incrementarse automáticamente o cambiar su valor en función de que el salto sea efectivo o no.

En la ASG si la instrucción *i* es un salto efectivo entonces ***el PC no se actualiza hasta el final de la etapa MEM***, hasta haya verificado la condición y calculado la nueva dirección del PC en la etapa EX.

El PC se incrementa en cada ciclo automáticamente para apuntar a la siguiente instrucción a buscar.

Se incrementa 4 bytes si la memoria se direcciona por bytes.

Se incrementa 1 byte si la memoria se direcciona por palabras de 4 bytes.

En las instrucciones de salto condicional el PC debe cargarse con la dirección de destino del salto, si el salto es efectivo controlando el valor de carga un multiplexor:

- El valor actual incrementado si el salto no es efectivo.
- El correspondiente a la dirección de destino si el salto es efectivo, calculado en la etapa EX.

El esquema más fácil de implementar en ASG es detener la segmentación hasta que se conozca el resultado del salto, introduciendo instrucciones NOP después de cada instrucción de salto condicional, el problema es que se reduce el rendimiento de la segmentación.

La alternativa es dejar que las instrucciones sigan ejecutándose. El compilador introduce en esos huecos instrucciones que se tienen que ejecutar antes de la instrucción de destino de salto siendo el efecto independiente si el salto es efectivo o no.

De esta forma el cauce puede terminar una instrucción por ciclo, mejorando el rendimiento, a esto lo llamamos **s*alto retardado***.

Un esquema mejor y un poco más complejo es predecir el salto como no efectivo, permitiendo que el hardware continúe procesando la siguiente instrucción en la secuencia del programa como si el salto fuese efectivo, se conoce como ***ejecución especulativa***.

Hay que tener cuidado de no cambiar el estado de la máquina hasta que no se conozca definitivamente el resultado del salto.

El estado de la máquina viene definido por el contador de programa, el fichero de registros y el espacio de memoria asignado al programa.

Si el salto es efectivo se debe detener la segmentación y recomenzar la búsqueda de la instrucción destino del salto. Se eliminan las instrucciones que se estaban ejecutando detrás de la instrucción de salto, perdiendo ciclos de reloj.

Si el salto no es efectivo no se penaliza con pérdidas de ciclo de reloj.

Se mejora considerablemente el rendimiento del procesador frente a los saltos si se adelanta el cálculo de la dirección y de la condición de la instrucción de salto a las primeras etapas del cauce

1.9 El algoritmo de Tomasulo como técnica de planificación dinámica en segmentación.

Hasta el momento utilizamos la planificación estática como única técnica en un procesador segmentado. Una de las principales limitaciones de la segmentación estática es que emiten las instrucciones en orden, si el procesador se detiene con una instrucción las posteriores no pueden proceder aunque no mantengan ninguna dependencia con las instrucciones que van por delante en el cauce. Una dependencia entre dos instrucciones puede dar lugar a un riesgo y a una detención.

En la planificación dinámica el hardware reorganiza la ejecución de la instrucción para reducir las detenciones mientras mantiene el flujo de datos y la consistencia del estado del procesador y de la memoria.

Entre las ventajas de la planificación dinámica está el aprovechamiento más óptimo en tiempo real de una etapa EX con múltiples unidades funcionales con lo que simplificamos el trabajo del compilador, aunque esto supone un aumento de la complejidad del hardware.

La ***ejecución fuera de orden*** introduce la posibilidad de tener que gestionar más riesgos que no existirían en un cauce de cinco etapas ni en un procesador segmentado con operaciones en coma flotante.

Para permitir la ejecución fuera de orden hay que desdoblar la etapa ID del procesador en:

- ***Decodificación*** (ID, Instruction Decoding): decodificación de instrucciones y comprobación de los riesgos estructurales.
- ***Emisión*** (II, Instruction Issue): la instrucción espera hasta que no hay riesgos de tipo RAW y cuando estén listos todos los operandos fuente, se leen y se emiten la instrucción hacia la unidad funcional.

El algoritmo de Tomasulo es una de las primeras técnicas basada en la planificación dinámica, de este algoritmo se derivan las técnicas de planificación dinámica que utilizan todos los procesadores superescalares actuales. El algoritmo de Tomasulo se utilizó en la unidad de coma flotante del IBM 360/91, cuyo objetivo era conseguir un alto rendimiento y evitar los grandes retrasos que se tenían en los accesos a memoria y en las operaciones de coma flotante.

La Unidad de Coma Flotante FPU contiene:

- Dos unidades funcionales, una de ***suma flotante*** y otra de ***multiplicación/división flotante***.
- Tres ficheros de registro:
  - Los registros de coma flotante ***FR***.
  - Los buffers de coma flotante ***FB***.
  - Los buffers de almacenamiento de datos ***SDB***.

La generación de direcciones y el acceso a memoria se realizan fuera de la FPU.

ÍNDICE

Índice Analítico

R

S