TEMA 4:

ARQUITECTURAS CON PARALELISMO A NIVEL DE INSTRUCCIÓN (ILP)

Lección 2: CONSISTENCIA DEL PROCESADOR Y PROCEDIMIENTO DE SALTOS.

Consistencia. Reordenamiento.

Consistencia.

- El procesamiento de una instrucción hace referencia al paso de dicha instrucción por todas y cada una de las etapas del cauce superescalar que utiliza, desde que se capta hasta que la instrucción se retira del cauce. La ejecución de la instrucción se refiere al paso de la instrucción por las etapas del cauce en las que la unidad funcional correspondiente realiza la operación codificada en la instrucción.

- En el procesamiento de una instrucción se puede distinguir entre:
    - El Final de la Ejecución de la Operación codificada en las instrucciones.
        - Se dispone de los resultados generados por las unidades funcionales, pero no se han modificado los registros de la arquitectura.

    - El Final del Procesamiento de la Instrucción o momento en que se *retira* o *completa* la instrucción *(Complete o Commit)*.
        - Se escriben los resultados de la Operación en los Registros de la Arquitectura. Si se utiliza un Buffer de reorden, ROB, se utiliza el término *Retirar la Instrucción*, en lugar de *Completar*.

- La consistencia de un programa se refiere a:
    - El orden en que las instrucciones se completan.
    - El orden en que se accede a memoria para leer (LOAD) o escribir (STORE).

- Cuando se ejecutan instrucciones en paralelo, el orden en que termina (finish) esa ejecución puede variar según el orden que las correspondientes instrucciones tenían en el programa, pero debe existir consistencia entre el orden en que se completan las instrucciones y el orden secuencial que tienen en el código de programa.

- Consistencia de procesador: consistencia en el orden en que se completa el procesamiento de las instrucciones.
    - Si se utiliza renombramiento de registros para evitar los efectos de los riesgos WAW y WAR y aprovechar el máximo paralelismo entre instrucciones, al final de la ejecución de la operación, el resultado se encontrará en una de las líneas de un buffer.
    - Débil: Las instrucciones se pueden completar desordenadamente siempre que no se vean afectadas las dependencias. Deben detectarse y resolverse las dependencias.
    - Fuerte (es a lo que tiende a ser): Las instrucciones deben completarse estrictamente en el orden en que están en el programa. Se consigue mediante el uso de ROB.

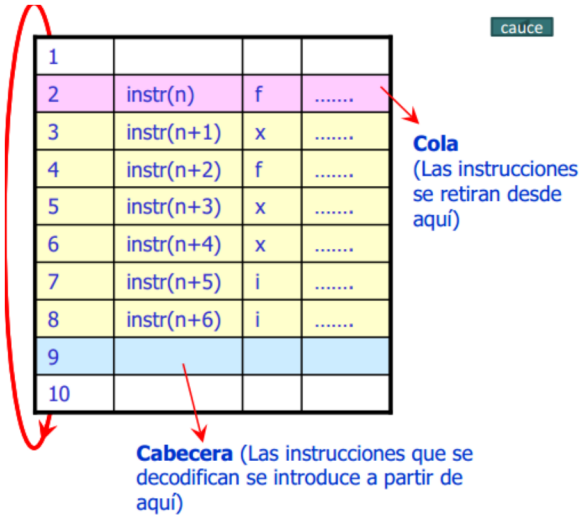
- Consistencia de memoria: consistencia del orden en que se realizan los accesos a memoria.
    - Los accesos a memoria se realizan a través de lecturas de memoria que hacen que los datos pasen de memoria a registros del procesador, o bien a través de escrituras en memoria que transfieren datos de registros del procesador a memoria. En los repertorios de tipo RISC, tales lecturas y escrituras se realizan únicamente a través de instrucciones de carga (LOADs) que transfieren datos de memoria a registros de la arquitectura, e instrucciones de almacenamiento (STOREs) que transfieren contenidos de registros de la arquitectura a memoria.
    - Hay que tener en cuenta que el tiempo de acceso a memoria puede ser considerable si la dirección a la que se accede no está en caché y hay que acceder a memoria principal. En ese caso, si la instrucción es un LOAD, se retrasarían considerablemente las instrucciones que la sigan, en el caso de que se pretenda mantener el orden en que las instrucciones aparecen en el programa.
    - Si una instrucción STORE no puede iniciarse porque no está disponible el operando a almacenar en memoria, se podría aprovechar que los buses de acceso a memoria no se están utilizando para iniciar alguna instrucción de acceso a memoria que está después de la instrucción STORE. De esta manera se mejora el aprovechamiento del sistema y se gana tiempo.
    - Consistencia de memoria Débil (es a lo que tiende a ser): Los accesos a memoria (Load/Stores) pueden realizarse desordenadamente siempre que no afecten a las dependencias. Deben detectarse y resolverse las dependencias de acceso a memoria.
    - Consistencia de memoria Fuerte: Los accesos a memoria deben realizarse estrictamente en el orden en que están en el programa. Se consigue mediante el uso de ROB.

Reordenamiento Load/Store:

- Las instrucciones LOAD y STORE implican cambios en el Procesador y en Memoria:
    - LOAD:
        - Cálculo de Dirección en ALU o Unidad de Direcciones.
        - Acceso a Caché.
        - Escritura del Dato en Registro.
    - STORE:
        - Cálculo de Dirección en ALU o Unidad de Direcciones.
        - Esperar que esté disponible el dato a almacenar (en ese momento acaba).

- La consistencia de memoria débil.
    - "Bypass" de Loads/Stores: Los Loads pueden adelantarse a los Stores pendientes y viceversa (siempre que no se violen las dependencias).
    - Permite los Loads y Stores especulativos: Cuando un Load se adelanta a un Store que le precede antes de que se haya determinado la dirección se habla de Load especulativo. Igual para un Store que se adelanta a un Load o a un Store.
    - Permite ocultar las Faltas de Caché: Si se adelanta un acceso a memoria a otro que dio lugar a una falta de caché y accede a Memoria Principal.

Buffer de Reordenamiento (ROB).



- El ROB es una estructura circular en el que, en cada una de sus elementos o líneas, se va introduciendo información de las instrucciones con el mismo orden con el que dichas instrucciones han sido decodificadas. Cada línea del ROB contiene información de una instrucción según los campos.
- El ROB permite gestionar correctamente el procesamiento especulativo de las instrucciones de salto y las interrupciones. Por otra parte, el ROB también puede utilizarse para el renombramiento.
- El puntero de cabecera apunta la siguiente posición libre y el puntero de cola a la siguiente instrucción a retirar.
- Las instrucciones se introducen en el ROB en orden de programa estricto y pueden estar marcadas como emitidas (issued, i), en ejecución (x) o finalizada su ejecución (f).
- Una instrucción solo se puede retirar (y modificar los registros de la arquitectura) si ha terminado su ejecución, y todas las que les preceden también.
- La consistencia se mantiene porque sólo las instrucciones que se retiran del ROB se completan (escribe en los registros de la arquitectura) y se retiran en el orden estricto de programa.
- La gestión de interrupciones y la ejecución especulativa se pueden implementar fácilmente mediante el ROB.

Procesamiento especulativo de saltos.

Un procesador superescalar, al ser un procesador segmentado, también sufre una reducción de prestaciones como consecuencia de los cambios en la secuencia de instrucciones que debe introducirse en el cauce cuando se produce un salto. El efecto de los saltos en los superescalares es todavía más grave, dado que al captarse, decodificarse y emitirse varias instrucciones por ciclo puede ocurrir que, en determinados códigos, prácticamente en cada ciclo haya una instrucción de salto. Además, aumenta el número de instrucciones que podrían introducirse incorrectamente en el cauce y dar lugar a un uso ineficiente en recursos.

Teniendo en cuenta la forma en que las instrucciones se procesan en cada una de las etapas de un superescalar, el salto retardado no resultará útil aquí. Esto se debe a que es la unidad de emisión en la que decide que instrucciones pasan a ejecutarse, independientemente del orden.

Los elementos hardware de que dispone un procesador superescalar relevan al compilador de parte del trabajo de optimización a realizar en un procesador segmentado, en donde se tienen que ordenar las instrucciones para conseguir el mejor rendimiento posible.

Clasificación de Saltos.

- Saltos incondicionales:
  - Salto incondicional.
  - Llamada a subrutina.
  - Retorno de subrutina.
- Saltos condicionales:
  - Condición de Bucle.
  - Otro Salto Condicional.

Aspectos del Procesamiento de Saltos en un procesador Superescalar:

- Detección de la instrucción de salto: cuanto antes se detecte que una instrucción es de salto, menor será la posible penalización y antes comenzará a procesarse como tal, por ejemplo, iniciando el cálculo de la dirección de destino del salto lo más pronto posible. Si se identifica que una instrucción es de salto en la etapa de decodificación y se determina la dirección de destino de salto en ese mismo ciclo de captación, en el siguiente ciclo se pueden empezar a captar instrucciones a partir de la dirección adecuada sin que exista prácticamente penalización. La etapa de predecodificación es muy útil para esta detección rápida de la instrucción de salto.

  Existen varias posibilidades para la detección temprana de las instrucciones de salto:
  1. Detección paralela: Hay una etapa específica para detectar instrucciones de salto que opera en paralelo con una etapa común de decodificación.
  2. Detección anticipada: Además de utilizarse la decodificación paralela, se analizan las últimas líneas de la cola de instrucciones captadas antes de que pasen a la unidad de decodificación.
  3. Detección integrada en la captación: En el momento en que se captan las instrucciones se detecta si la instrucción es de salto o no (uso de los bits de la predecodificación).

- Gestión de los saltos condicionales no resueltos: Si en el momento en que la instrucción evalúa la condición de salto, ésta no se halla disponible, se dice que el salto o la condición no se ha resuelto. Para resolver este problema se suele utilizar el procesamiento especulativo del salto.
  En cuanto al establecimiento de la condición de salto, existen alternativas que suelen implementarse:
  - Bits de estado: Opción de utilizar un registro de estado a cuyos bits se asignan determinados significados (bit de signo, de cero, de desbordamiento u overflow, etc.) y son modificados al ejecutar una instrucción de acuerdo con las características del resultado. La instrucción de salto condicional comprueba el valor de los bits de estado adecuados según sea la condición que determine el salto. Esta forma de realizar los saltos condicionales establece una dependencia entre las instrucciones que viene determinada por la ubicación de dichas instrucciones, en lugar de aparecer explícitamente a través de los operandos de cada instrucción. Este esquema plantea importantes dificultades para su implementación en una arquitectura segmentada superescalar en la que se pretende que las instrucciones puedan ejecutarse desordenadamente para aprovechar el máximo paralelismo, y es la propia máquina la que detecta la dependencia entre las instrucciones comprobando los operandos que necesita cada una.

  - Comprobación directa: Los resultados de las operaciones se comprueban directamente respecto a las condiciones específicas mediante instrucciones específicas. Para esta alternativa hay dos posibilidades: utilizar una instrucción o dos instrucciones.
    - Dos instrucciones: una de ellas evalúa la condición y actualiza convenientemente un registro de la arquitectura, y la instrucción de salto propiamente dicha tiene en cuenta el valor de ese registro para dar lugar a la bifurcación o no.
    - Una instrucción: se encargaría tanto de evaluar la condición como de dar lugar al salto o no, según el caso.

- Acceso a las Instrucciones destino del Salto: Hay que implementar procedimientos que permitan el acceso más rápido posible a la secuencia de instrucciones a la que se produce el salto, una vez resuelta la condición de salto.

- El efecto de los saltos en los procesadores superescalares es más pernicioso ya que, al emitirse varias instrucciones por ciclo, prácticamente en cada ciclo puede haber una instrucción de salto.

Gestión de Saltos Condicionales no resueltos:

- Una vez detectada la instrucción de salto y la dirección de destino, interesa conocer cuanto antes si se verifica la condición de salto para empezar a captar cuanto antes las instrucciones desde la dirección del destino, y si finalmente se hace el salto, y evitar que se introduzcan en el cauce instrucciones que no deben procesarse. Reducir el número de ciclos que pasan desde que se capta una instrucción y se puede empezar a captar instrucciones es esencial para reducir la penalización que ocasionan los saltos condicionales.

- En el caso del uso del salto retardado se utilizan ciclos que siguen a la captación de una instrucción de salto para insertar instrucciones que deben ejecutarse independientemente del resultado del salto. Esta técnica no es efectiva en el caso de procesadores superescalares dada la planificación dinámica de instrucciones que realiza el procesador.

- Una condición de salto no se puede comprobar si no se ha terminado de evaluar:
  - Bloqueo del procesamiento de salto: Se bloquea la instrucción de salto hasta que la condición esté disponible. Esta situación es la más desfavorable desde el punto de vista de las prestaciones y, en ese sentido se puede tomar como punto de partida para posibles mejoras.
  - Procesamiento especulativo de los saltos: Opción más frecuente. El procedimiento se basa en estimar de alguna forma el camino que, con más probabilidad, va a seguir la ejecución cuando se haya evaluado la condición de salto. Las instrucciones se captan teniendo en cuenta esa predicción, y una vez que se conoce si se debe producir el salto o no, continúa la ejecución si se acertó en la predicción o se recupera el camino correcto si hubo error en la predicción.
  - Múltiples Caminos: Cada vez que llega una instrucción de salto, el procesador empezaría a captar y procesar instrucciones de los dos caminos alternativos. En ese momento se cancela el camino incorrecto. Esta opción es la más costosa en cuanto a demanda de recursos en el procesador.

Esquemas de predicción de salto:

- La predicción de salto se basa en la idea de que el comportamiento de una instrucción de salto condicional presenta una cierta regularidad, y por tanto puede predecirse con una tasa de aciertos suficientemente elevada. En el momento en el que se detecta una instrucción de salto condicional, se sigue una estrategia para predecir si se producirá o no el salto antes de que se complete la evaluación de la condición de la que depende la instrucción de salto. Lógicamente, una vez terminado el cálculo de la condición de salto habrá que comparar lo que se debe hacer según esa condición con la decisión que se tomó a partir de la predicción. Si no hay coincidencia habrá que deshacer el efecto de las instrucciones que se han captado incorrectamente, y empezar a captar instrucciones desde la dirección adecuada según la condición de salto.

- En caso de error en la predicción, habrá una penalización. Así, como en cualquier otra técnica de procesamiento especulativo, interesa disponer de un procedimiento con una tasa de aciertos lo más elevada posible y una implementación que permita la recuperación rápida de errores.

- Predicción fija: Se toma siempre la misma decisión ante cualquier instrucción de salto condicional: el salto siempre se realiza, '*taken*' (tomado), o no, '*not taken*' (no tomado).
    - A la hora de diseñar el procesador se decide implementar un esquema u otro en función de lo más probable, de la penalización asociada a la recuperación en caso de predicción errónea y de la complejidad del hardware que se requiere para implementar el esquema de predicción.

    - En el caso de '*salto siempre no tomado*': Para toda instrucción de salto cuya condición no está resuelta se considera que lo más probable es que no se produzca el salto. Tras evaluar la condición de salto se comprueba si la predicción ha sido correcta, y si es así se continua la ejecución.

    - En el caso de '*salto siempre tomado*': Se considera que toda instrucción de salto condicional no resuelta, da lugar a un salto, por lo que se empiezan a captar instrucciones a partir de la dirección de destino del salto. En previsión de que se pueda producir un error en la predicción, se guarda valor del contador de programa (PC), que apunta la instrucción siguiente a la instrucción de salto condicional. Si la predicción era incorrecta, se recupera el valor del contador almacenado y prosigue la captación de instrucciones a partir de esa dirección. La secuencia de instrucciones que se estaba ejecutando se abandona, deshaciendo el efecto que hayan podido tener en los registros o estado del procesador.

    - En el caso de '*no tomado*' se necesita una implementación más sencilla ya que si es '*tomado*' es necesario disponer de una dirección de destino de salto lo más rápido posible para no ocasionar penalizaciones, aun en caso de que se acierte en la predicción. Sin embargo, la predicción de salto '*tomado*' proporciona mejores prestaciones debido a que es más probable que se produzca salto.

- Predicción Verdadera:
    - La decisión depende de las características propias de la instrucción de salto condicional concreta que se esté ejecutando, por eso se dice que son de predicción verdadera.

    - La decisión de si se realiza o no se realiza el salto se toma mediante la predicción estática o dinámica.

    - Predicción estática: Según los atributos de la instrucción de salto (el código de operación, el desplazamiento, la decisión del compilador). La predicción se hace sobre la base de atributos de la instrucción de salto que están relacionados con la probabilidad de que dicha instrucción ocasione o no un salto. Se habla de predicción estática porque se trata de características que no cambian al ejecutar el código. Ejemplos:
        - Predicción basada en el código de operación: Según el tipo de instrucción de salto condicional se considera más probable que se produce el salto o que no.
        - Predicción basada en el Desplazamiento del Salto: Si el desplazamiento es positivo (salto hacia delante) se predice que no se toma el salto y si el desplazamiento es negativo (salto hacia atrás) se predice que se toma.
        - Predicción dirigida por el compilador: A través de una serie de bits que existen en las instrucciones de salto (bits de predicción), el compilador puede fijar la predicción que se va a realizar para la instrucción. Para fijar esos bits, el compilador puede tener en cuenta los resultados obtenidos a partir del tipo de comportamiento es el más probable para cada instrucción de salto.

    - Predicción dinámica: Según el resultado de ejecuciones pasadas de la instrucción (historia de la instrucción de salto).
        - En cada ejecución de un programa se pretende utilizar el comportamiento observado de salto/no salto de cada instrucción de salto condicional para establecer la predicción a realizar cuando esa instrucción vuelva a captarse.
        - El presupuesto básico de la predicción dinámica es que es más probable que el resultado de una instrucción de salto sea similar al que se tuvo en la última (o en las n últimas ejecuciones).
        - Para cada instrucción de salto condicional debe guardarse algún tipo de información que condense el comportamiento pasado y permita realizar la predicción.
        - Presenta mejores prestaciones de predicción, aunque su implementación es más costosa.
        - Predicción dinámica Implícita: no se guarda ninguna información explícita que represente el comportamiento pasado de la instrucción. Se almacena únicamente la dirección de la instrucción que se ejecutó tras la instrucción de salto la última vez que se captó ésta. Este esquema es bastante limitado ya que predice hacer lo mismo que ocurrió la última vez que se ejecutó esa instrucción.
        - Predicción dinámica Explícita: Para cada instrucción de salto existen un conjunto bits que codifican la información relativa al comportamiento pasado de la instrucción en cuestión. Estos bits se denominan 'bits de historia', y el número de estos bits que se guardan dependen del tipo de esquema de predicción dinámica explícita que se haga.

Extensión del Procesamiento Especulativo:

- Tras la predicción, el procesador continúa ejecutando instrucciones especulativamente hasta que se resuelve la condición.

- El intervalo de tiempo entre el comienzo de la ejecución especulativa y la resolución de la condición puede variar considerablemente y ser bastante largo. Durante la ejecución especulativa de un salto condicional puede suceder que se capten nuevas instrucciones de salto condicional que tampoco estén resueltas. En general, cuanto mayor sea el número de instrucciones que puedan ejecutarse especulativamente, mejores prestaciones proporcionará el sistema de gestión de saltos condicionales.

- También hay que tener en cuenta que, si el número de instrucciones que se ejecutan especulativamente es muy elevado y la predicción es incorrecta, la penalización asociada al procesamiento de la instrucción de salto es mayor.

- En los procesadores superescalares, que pueden emitir varias instrucciones por ciclo, pueden aparecer más instrucciones de salto condicional no resueltas durante la ejecución especulativa.

- Así, cuanto mejor es el esquema de predicción mayor puede ser el número de instrucciones ejecutadas especulativamente.
    - Nivel de especulación: Número de instrucciones de salto condicional sucesivas que pueden ejecutarse especulativamente (si se permiten varias, hay que guardar varios estados de ejecución).
    - Grado de especulación: Indica hasta qué etapa se ejecutan las instrucciones que siguen en un camino especulativo después de un salto condicional.

Instrucciones de Ejecución Condicional:

- Una alternativa para resolver los problemas que acarrean las instrucciones de salto condicional, en lo que respecta a la reducción de prestaciones que ocasionan en los procesadores superescalares, consiste en eliminar las instrucciones de salto en los códigos.

- Esto se puede conseguir introduciendo en el repertorio de instrucciones máquina una serie de instrucciones con predicado, o instrucciones de ejecución condicional o vigilada.

- Estas instrucciones tienen dos partes: la condición (denominada *guardia*) y la parte de operación.

- La operación tiene efecto si la condición se verifica.

- Es posible diseñar un repertorio en el que todas las instrucciones puedan ejecutarse en función de una condición que evalúa la propia instrucción. En tal caso, se podrían eliminar unos porcentajes elevados de saltos en los programas.