TEMA 4:

ARQUITECTURAS CON PARALELISMO A NIVEL DE INSTRUCCIÓN (ILP)

Lección 3: PROCESAMIENTO VLIW.

Características generales y motivación (ILP hardware vs. ILP software).

Un VLIW es un procesador segmentado que puede terminar más de una operación por ciclo en el que el compilador es el principal responsable de agrupar operaciones que puedan procesarse en paralelo para definir instrucciones que, de esta forma, se codifican a través de las denominadas palabras de instrucción larga (LIW: Long Instruction Word) o muy larga (VLIW: Very Long Instruction Word).

Mientras que en un procesador superescalar, cada una de las operaciones de una instrucción VLIW se codificaría mediante una única instrucción y las dependencias entre ellas se pueden comprobar en el propio cauce, en un procesador VLIW gracias al trabajo del compilador, las operaciones empaquetadas en una instrucción VLIW son independientes y se emiten a mismo tiempo a las unidades funcionales sin más comprobación. Así mientras que en un procesador superescalar, la planificación de instrucciones es dinámica, en un procesador VLIW es estática.

Características generales de los procesadores VLIW.

- En un procesador VLIW es el compilador el que se encarga de planificar la distribución de las operaciones del programa.

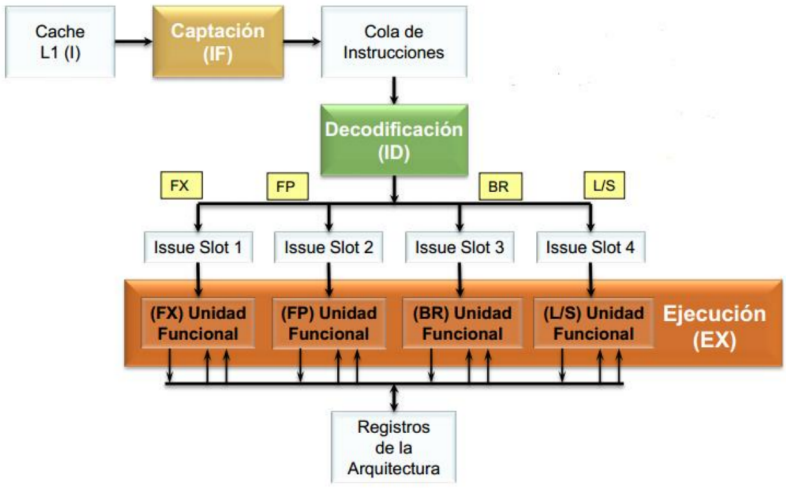
- En los procesadores VLIW el paralelismo se indica explícitamente en cada una de las instrucciones que se captan (a diferencia del paralelismo en los procesadores superescalares, que es el hardware quien tiene que descubrir el paralelismo que puede aprovecharse en las instrucciones que se van captando). Es decir, en cada instrucción VLIW codifica operaciones que se ejecutan simultáneamente.

- En un procesador VLIW cada palabra de instrucción está constituida por un conjunto de subpalabras o '*slots*', y cada una de tales palabras puede codificar una operación. Las operaciones que se codifican en cada una de las subpalabras de una instrucción VLIW se deben poder ejecutar en paralelo. Es decir, no existen dependencias de datos ni control entre ellas, y el computador dispone de las correspondientes unidades funcionales que se necesitan para la ejecución simultánea de operaciones.

- Por tanto, el compilador debe encargarse de ubicar las distintas operaciones que deben realizarse en un programa en las distintas subpalabras de las instrucciones VLIW teniendo en cuenta las dependencias y los recursos de procesador. En cada ciclo, el procesador VLIW emitirá una instrucción VLIW, es decir, cada una de las operaciones codificadas en las subpalabras de la instrucción se emite a su correspondiente unidad funcional.

- Dado que es el compilador el que ha ordenado las operaciones de dentro de las subpalabras de las instrucciones VLIW de forma que las operaciones que se incluyen en cada instrucción VLIW sean independientes entre sí, los VLIWs no necesitan disponer de estaciones de reserva, buffers etc., puesto que se trata de recursos que usan los procesadores superescalares para determinar las dependencias entre instrucciones y garantizar la consistencia de programas con ejecución desordenada de instrucciones.

- Las microarquitecturas VLIW, por tanto, son más sencillas.



- Se trata de una microarquitectura segmentada en la que las instrucciones que se captan pasan a una cola desde la que se emite una instrucción por ciclo a las unidades de ejecución. Cada instrucción VLIW codifica varias operaciones (cuatro en la imagen de arriba) que se emitirán en el mismo ciclo a las correspondientes unidades funcionales.

- A medida que sea posible emitir más operaciones independientes en un ciclo mayor ventaja de una microarquitectura VLIW frente a otra superescalar que tenga el mismo grado de complejidad, puesto que será posible incluir unidades funcionales adicionales en la microarquitectura VLIW. Para que el procesador VLIW sea capaz de aprovechar el paralelismo, el compilador debe ser capaz de explicitarlo ordenado adecuadamente las operaciones en las distintas subpalabras de las instrucciones VLIW.

- Tras decodificarse la instrucción VLIW, cada una de las operaciones que codifica pasa a la ventana de emisión correspondiente para empezar a ejecutarse en el ciclo siguiente. En la imagen, hay cuatro ventanas de emisión (issue slot), una para cada una de las operaciones que se codifican en las subpalabras de la instrucción VLIW.

- Los tipos de operaciones que pueden codificarse en cada una de las subpalabras de la instrucción VLIW dependen de las unidades funcionales a las que se pueda acceder desde cada ventana de emisión.

- El trabajo del compilador se facilita si en cada una de las subpalabras de la instrucción VLIW se puede codificar cualquier tipo de operación. Para ello es necesario que cada una de las ventanas de emisión tenga acceso a las unidades funcionales necesarias para ejecutar todas las operaciones. El trabajo del compilador es esencial para aprovechar las posibilidades del procesador, y en esta labor, el compilador debe tener un conocimiento muy preciso de las características de la microarquitectura.

- Los operandos se encuentran en un banco de registros y los resultados también se escriben en registros de dicho banco. La conexión entre el banco de registros y unidades funcionales se realiza a través de la correspondiente estructura de interconexión. La complejidad de esta estructura crece con el número de ventanas de emisión de la microarquitectura VLIW y lo mismo ocurre con la complejidad hardware asociada al banco de registros, que precisará de un conjunto elevado de puertos de lectura/escritura para proporcionar los operandos y recibir los resultados de las operaciones que se ejecutan en paralelo.

Planificación estática.

- Cada una de las instrucciones VLIW contiene las operaciones que van a ejecutarse en paralelo, pasando todas ellas al mismo tiempo a cada una de las unidades funcionales que se utilizarán.

- La capacidad del compilador para descubrir el paralelismo es crucial para aprovechar al máximo los recursos que ofrece un procesador VLIW. En un procesador superescalar el papel del compilador es menos importante, si bien puede ayudar a minimizar los efectos derivados de las limitaciones del hardware que se añade para realizar la planificación dinámica en aplicaciones con determinadas características.

- El problema con el que se enfrenta el compilador a la hora de generar código eficiente para un procesador VLIW es encontrar un número suficiente de operaciones entre las que no existan dependencias, de forma que se puedan ir ocupando las subpalabras o '*slots*' de cada una de las instrucciones VLIW y procurar que el orden entre las instrucciones VLIW que se generan permita un buen aprovechamiento del cauce, sin que se produzcan los atascos que reducen el rendimiento del procesador.

- Existen dependencias RAW entre cada dos instrucciones consecutivas, y además existe una instrucción de salto que controla el final del bucle:
    - Parece que no se puede aprovechar mucho ILP.

- Un bloque básico es un trozo de código al que internamente no se accede desde ninguna instrucción de salto (sólo la primera instrucción puede ser el destino de una instrucción de salto externa al bloque), y en el que no existen instrucciones de salto, la última instrucción del bloque. En un bloque básico las dependencias que hay que tener en cuenta son las dependencias de datos y las posibles colisiones debidas a la disponibilidad de unidades funcionales donde se puedan ejecutar las operaciones.

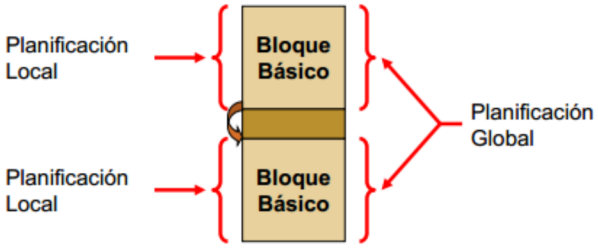
- Planificación local: Actúa sobre un bloque básico (mediante desenrollado de bucles y planificación de las instrucciones del cuerpo aumentado del bucle).

- Desenrollado de bucles:
    - Se transforma el bucle de partida en un nuevo bloque cuyas iteraciones incluyen las operaciones de varias de las iteraciones del bucle de partida no desenrollado. Se tienen así bloques básicos con más instrucciones y, al mismo tiempo, disminuye el número de instrucciones de salto a ejecutar para controlar el bucle dado que el número de iteraciones es menor. Al tener bloques básicos con más instrucciones, el compilador dispone de más operaciones sin dependencias entre ellas para ubicarlas en los '*slots*' de las instrucciones VLIW. Al disponerse de un bloque básico más grande hay más operaciones independientes para distribuir entre los '*slots*' de las instrucciones VLIW.
    - Al desenrollar un bucle se crean bloques básicos más largos, lo que facilita la planificación local de sus sentencias.
    - Además de disponer de más sentencias, éstas suelen ser independientes, ya que operan sobre diferentes datos.
    - PROBLEMA: Tamaño de memoria ocupado por el programa es mayor a pesar de que se pueda mejorar el uso de la misma. Existe una técnica que permite transformar los bucles aumentando el número de operaciones independientes que contienen los bloques básicos sin que se produzca un incremento sustancial en el tamaño del cuerpo del bucle. Se trata de la segmentación software.

- Segmentación software (software pipelining):
    - Esta técnica pretende reorganizar el cuerpo del bucle sobre el que se aplica, de forma que cada iteración del bucle transformado contenga instrucciones tomadas de distintas iteraciones del bucle original con el objetivo de situar las instrucciones dependientes lo más alejadas posible.
    - Se basa en que en el cuerpo de un bucle suelen existir instrucciones que cargan los datos, otras que hacen operaciones sobre los mismos, y por último están las instrucciones que almacenan los resultados. Entre las instrucciones de una misma iteración suelen darse dependencias de tipo RAW porque las instrucciones que operan con los datos necesitan que las que los cargan se hayan ejecutado y las que almacenan los resultados necesitan que las que hacen los cálculos terminen de ejecutarse.
    - Sin embargo, si se construyen bucles en cuyas iteraciones se utilicen instrucciones que cargan los datos que se utilizan en la iteración siguiente, se hacen los cálculos con los datos que se cargaron en la iteración anterior, y se almacenan los resultados de los cálculos de la iteración anterior, las instrucciones del cuerpo del bucle no tienen dependencias de datos entre ellas.

- Planificación global: Actúa considerando bloques de código entre instrucciones de salto.
    - Extrae paralelismo entre bloques básicos diferentes, es decir, entre operaciones que están separadas por instrucciones de salto condicional. En principio, no sería posible, situar en una misma instrucción VLIW operaciones entre las que existe algún salto condicional que determina si, al final, se tienen que ejecutar ambas operaciones o sólo una de ellas. De todas formas, aunque no es posible determinar en el momento de compilación cuál será el flujo de control resultado de la ejecución de la operación de salto condicional, existen técnicas denominadas de *planificación global*, que consideran grupos de bloques básicos como si fueran uno solo.
    - La planificación global mueve código a través de los saltos condicionales (que no correspondan al control del bucle).

- Se parte de una estimación de las frecuencias de ejecución de las posibles alternativas tras una instrucción de salto condicional.
- Apoyo para facilitar la planificación global:
  - Instrucciones con predicado.
  - Especulación.



- Instrucciones/operaciones con predicado: Permiten reducir el número de saltos condicionales que hay en un código para conseguir bloques básicos más grandes para disponer de más operaciones paralelas que distribuir entre los '*slots*'.
  - Ocasiona que disminuyan las dependencias de control en los programas, lo cual es bastante importante, sobre todo si entre las opciones de un salto condicional no hay una que sea mucho más frecuente que otra y facilite realizar una predicción estática de saltos eficiente.
  - En los repertorios de instrucciones escalares utilizados por distintos procesadores superescalares se han ido incorporando instrucciones de ejecución condicional.
  - Dentro de este tipo de instrucciones, las más frecuentemente utilizadas son las de transferencia condicional de datos. No obstante, si se dispone únicamente de instrucciones de trasferencia condicional de datos, la transformación de trozos de código largos que dependen de saltos puede ser bastante ineficiente. Por esta razón se han propuesto repertorios en los que la ejecución de cualquier instrucción puede controlarse mediante el uso de predicados.
  - Una operación con predicado es aquella cuyo resultado modifica o no el destino de dicha operación en función del valor de un operando, denominado predicado, que, por tanto, establece la condición de la que depende que la operación tenga efecto. El predicado solo puede tomar dos valores: verdadero o falso.
  - Por supuesto, son necesarias operaciones que permitan asignar valores a los predicados según se verifiquen o no determinadas condiciones. En algunos procesadores incluso existen registros especiales para almacenar predicados.
  - El uso de predicados permite reducir el número de operaciones de salto condicional que hay en los programas permitiendo definir bloques básicos mayores de forma que al compilador le resulte más sencillo ocupar las subpalabras de las instrucciones VLIW.
  - Existen una serie de cuestiones que limitan la utilidad del uso de predicados. Entre ellas están:
    - El uso de instrucciones con predicado para facilitar el desplazamiento de instrucciones sigue siendo una forma de especulación, y supone un costo cuando la instrucción no debería haberse ejecutado.
    - Si la condición que debe evaluarse en una instrucción con predicado no está disponible con la suficiente antelación que producirían atascos en el cauce (consecuencia de las dependencias RAW que hay que tener en cuenta a la hora de planificar el código).

- Cuando el flujo contiene más de una alternativa puede resultar complicado el uso de predicados.

Procesamiento especulativo.

El procesamiento especulativo se basa en que el cumplimiento de una condición será muy probable, y por ello se puede adelantar el procesamiento de operaciones que habrá que ejecutar tras evaluar la condición y verificar el resultado. Al adelantar el procesamiento se pueden utilizar recursos que hubieran estado ociosos, por lo que mejorará el rendimiento del procesador.

En otras palabras, las instrucciones con predicado o de ejecución condicional permiten que el procesador pueda aplicar técnicas de procesamiento especulativo para aprovechar información de que disponga acerca de la mayor o menor probabilidad de que se verifique una condición y, por lo tanto, haya que ejecutar una u otra alternativa tras una instrucción de salto. En el caso de los procesadores VLIW, la especulación debe hacerse en el momento de la compilación. Por esta razón, para mejorar el rendimiento del compilador en la especulación que lleva a cabo, los procesadores VLIW de propósito general suelen incorporar ciertos recursos hardware.

El procesamiento especulativo tiene un coste si la predicción que se ha hecho no es correcta. Este coste va desde el correspondiente a haber ejecutado una instrucción que no tendría que haberse ejecutado, hasta la necesidad de incluir código que deshaga el efecto de la operación implementada, vigilar el comportamiento frente a las excepciones, etc.

En el caso más favorable este coste puede estar asociado únicamente con la ejecución de la operación que se ha adelantado, cuyo procesamiento no habría sido necesario.

A continuación, se describen una serie de estrategias que permiten garantizar el comportamiento del programa frente a excepciones cuando se permiten movimientos especulativos del código.

1. Ni el hardware ni el sistema operativo aceptan excepciones que causen la finalización del programa.
    - Las excepciones que no ocasionan que el programa termine, como por ejemplo un fallo de página, se aceptan normalmente. En cambio, para las excepciones que obligan a que el programa termine se devolvería un valor indefinido y el programa continuaría.

2. Las instrucciones especulativas nunca generan excepciones y se añaden elementos para comprobar las condiciones de excepción.
    - Las operaciones que se adelantan especulativamente se marcan como tales y no dan lugar a ningún tipo de excepción. En el lugar que ocupaba la operación que se ha adelantado se sitúa una operación con el propósito de determinar si se ha producido alguna de las causas que podrían generar una excepción.

3. Uso de bits de veneno (poison bits).
    - Se añaden una serie de bits, denominados bits de veneno, a los registros de la arquitectura. Si se produce una excepción al ejecutarse una operación especulativa se modifica el bit de veneno del registro en el que la operación especulativa escribe su resultado. Cuando una operación no especulativa intenta leer un registro se comprueba el bit de veneno y, si está activo, se atiende la excepción. Es decir, cuando una operación especulativa da lugar a una excepción, ésta se atiende en el momento en que el dato erróneo se va a utilizar.

4. Uso de centinelas.
    - El resultado de las operaciones especulativas se guarda en una estructura de almacenamiento temporal hasta que dicho resultado deja de ser especulativo. En ese momento el resultado de la operación se actualiza en la localización que indica la operación. El compilador marca las operaciones especulativas y se utiliza un contador que contiene información del número de saltos que se han sobrepasado al adelantar una operación especulativamente junto con las suposiciones que realiza el compilador. Así, el hardware puede determinar el bloque de código en que se supone que estaba la operación. En el lugar en el que estaba la operación se sitúa un centinela, es decir, un código que permite determinar el momento en que la operación que se adelantó deja de ser especulativa. Precisamente el hecho de acceder al centinela muestra que se habría ejecutado la operación que estaba donde ahora está el centinela.

Recursos para la especulación en los accesos a memoria.

- El compilador puede adelantar sin problemas las cargas de memoria (LOAD) con respecto a las escrituras en memoria (STORE) cuando no existe ambigüedad en las direcciones de memoria.

- Cuando no existe ninguna ambigüedad, el compilador adelanta los LOADs con respecto a los STOREs para reducir la longitud del camino crítico en el código.

- Cuando existe ambigüedad, como ésta lo resuelve en el momento de la ejecución del programa, el compilador no podría reorganizar los accesos a memoria para reducir el tiempo de ejecución sin que existan riesgos de error en el caso de que la dirección donde se escribe (STORE) sea, al final, la misma en la que se lee (LOAD).

- Es posible, no obstante, incluir recursos en la máquina que permitan que el compilador pueda reordenar los accesos a memoria aún en el caso de ambigüedad. Así, se puede añadir al repertorio una operación para comprobar la existencia de conflictos entre direcciones. Se utilizaría la siguiente forma:
    - La instrucción se sitúa en la posición original del LOAD que se ha adelantado al STORE correspondiente, actuando como centinela.
    - Cuando se ejecuta el LOAD especulativo, el hardware guarda la dirección a la que se ha realizado el acceso.
    - Las direcciones a la que acceden los STOREs que siguen al LOAD especulativo se comparan con la dirección del LOAD que se ha almacenado. Si ningún STORE accede a esa dirección, la especulación es correcta. En caso contrario, la especulación ha fallado.
    - Si la especulación ha fallado:
        - Si la especulación afecta al LOAD solamente, se vuelve a ejecutar cuando se llega al centinela.
        - Si se han ejecutado instrucciones que dependen del LOAD, puesto que utilizan como operando el dato que se ha leído en memoria, habrá que repetir todas esas instrucciones (se necesita mantener información de todas ellas en un trozo de código cuya dirección se incluye en la instrucción centinela). Las operaciones que deben repetirse para cada LOAD especulativo incluyen un trozo de código cuya dirección de inicio se indica en la instrucción centinela.