ARQUITECTURA E INGENIERÍA DE COMPUTADORES

Tema 2.5

Miquel Gómez Corral

Ing. Informática ETSINF, UPV

Tema 2.5

Lanzamiento múltiple de instrucciones

Se plantean Procesadores **superescalares**. Con Tomsulo tenemos CPI ≈ 1. Pues ahora queremos tener menos de 1. **Lanzar varias instrucciones a la vez**. Ya que `Te = I x CPI x T`. Podemos reducir alguno de ellos aún más.

Reducir CPI: Aumentar las instrucciones lanzadas a ejecución en un ciclo. **Procesadores superescalares**.

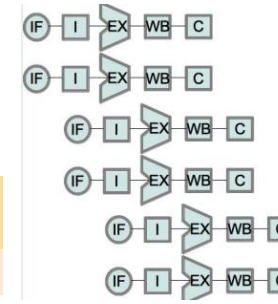
Reducir I: Aumentar el trabajo que realiza cada instrucción máquina. **Procesadores VLIW**. *A Intel le mola diu.*

Reducir T: Segmentar el ciclo de instrucción en más etapas aún. **Procesadores supersegmentados**. *No tan usado.*

PROCESADORES SUPERESCALARES

Se **lanzan "m" instrucciones por ciclo de reloj** --> $CPI = \frac{1}{m}$. Ahora hablaríamos de **Intructions Per Cycle** ($IPC = m$).

m: **Numero de vías del procesador superescalar** (*número de instrucciones lanzadas a ejecución en un ciclo*). Como lanzamos más nos mejora el tiempo de ejecución:



- $Te = I\ x \frac{CPI}{m}\ x\ T$. La aceleración respecto al no superescalar es de "m".

Requisitos hardware

Fase IF

Cache de instrucciones hay 1. Si quieres que varios lean en ella a la vez la **caché** tienes que hacerla el **doble de ancha**: Eso quiere decir que antes una instrucción te ocupaba 32 bits y tenias 1 por cada posición de la caché. Ahora haces que la caché sea más ancha, tenga **64 bits por posición** y que **haya 2 instrucciones en ella**: Tienes que ser consecutivas si no malamente. *Podrás aprovechas las 2 leídas si son consecutivas, con los saltos habrá problemas*.

Fase I

Para **decodificación de m instrucciones** hay que hacer una Comprobación de dependencias entre las m instrucciones buscadas simultáneamente y entre estas y las lanzadas anteriormente.

El **Decodificador será más complejo/rápido**. Múltiples ciclos de decodificación .

Lectura de operandos: Como ahora tienes que leer del **banco de registros** debe tener **más puertos de lectura**. También **más puertos en el ROB**: ***2 × m*** *puertos de reed en el* ***Banco de registros***. ***2 × m*** *puertos de reed en el* ***ROB***.

Fase EX

Habrá que **replicar operadores**, pero no "m" veces. Como hay batiburrillo de instrucciones, no siempre están todas en el mismo operando, por lo que no hace falta replicarlos todos tantas veces, solo algunas.

Fase WB

Ahora harán falta **m buses comunes de datos** y **m puertos de escritura en el ROB** para que varias hagan WB a la vez.

Fase C

Graduación de varias instrucciones a la vez (C). Hasta **m instrucciones** (*De las que estén en el ROB y por orden sean las primeras*) pueden **efectuar la fase Commit a la vez**. m puertos de escritura en el banco de registros y memoría.

Implicaciones

Probabilidad de riesgos: Ahora al **haber más instrucciones a la vez** hay un Aumento de la probabilidad de riesgos. *Además de los riesgos entre las instrucciones lanzadas en ciclos distintos, se producen riesgos entre las m instrucciones lanzadas en el mismo ciclo*.

Penalización por riesgos: Como mandas más instrucciones ahora cada vez que pares un 1 ciclo **Aumenta de la penalización**. *Un ciclo de parada son m instrucciones no lanzadas*. Ahora Aumentar el ILP es más importante.

Ejemplo: (4 RS de suma fp)

```
fadd.d f1,f2,f3   IF I  A1 A2 WB C
fadd.d f6,f4,f5   IF I  A1 A2 WB C
fadd.d f2,f1,f3      IF I  -  -  A1 A2 WB C
fadd.d f2,f1,f2      IF I  -  -  -  -  -  A1 A2 WB C
fadd.d f3,f3,f5         IF I  I  A1 A2 WB -  -  -  C
fadd.d f4,f4,f6         IF I  I  -  A1 A2 WB -  -  -  C
inst                       IF IF I  ...
inst                       IF IF I  ...
```

Problema adicional de los saltos: Si **no están alienadas** las **instrucciones** o las **direcciones** donde va a saltar vas a desperdiciar instrucciones. En concreto: m-1 instrucciones si la dirección del salto está mal y otras m-1 si la instrucción de salto también está mal.

```
inst          IF I  EX WB C
beqz t1,L     IF I  EX WB C
L-4:             IF <no sirve>
L:               IF I  EX WB C
```

```
beqz t1, L    IF I  EX WB C
inst          IF <no sirve>
L-4:             IF <no sirve>
L:               IF I  EX WB C
```

Cosa buena: No se modifica el juego de instrucciones del procesador segmentado de partida → Son **compatibles a nivel binario** con el procesador segmentado de partida.

Procesadores superescalares no uniformes o con restricciones

Como puede ser mu caro replicar todos los operandos m **NO todos son replicados en la misma magnitud**. Algunos más y otros menos. A estos se les llama **superescalares no uniformes**. *Se les tendrán que poner restricciones sobre cuantas instrucciones a la vez se pueden lanzar etc.*

Implicaciones: Aparición de riesgos estructurales, Si no pueden entrar a un operador se esperarán. Si hay muchos esperando se nos llenan las colas de espera, haciendo que haya aún más ciclos de parada, pero ahora en I → Peor.

El compilador puede ayudar mucho colocando juntas aquellas instrucciones que cumplen los requisitos. Lo que pasa es que ahora La compatibilidad a nivel binario puede no ser tan eficiente. *Es todo muy funny jijijijijijjjjjjijijijijjjjjjj.*

PROCESADORES MULTIHILO

Antes considerábamos el ILP. Pero **lanzando más** y más **instrucciones** por ciclo **no aumenta mucho el IPC**. Ahora miramos el TLP: *Thread Level Parallelism*. Se pretende **aprovechar los ratos que estén los hilos quietos para procesar otros hilos** y usar mejor los recursos del procesador.

Se prende hacer varias cosas en paralelo: O bien **tienes diferentes procesos dentro de un núcleo** (*multihilo*) *haciendo que entre todos uses el 100% de la máquina* o **replicando núcleos**: *Tienes diferentes procesos y diferentes procesadores y en cada uno algo*.

- **Aumentar la productividad**: Ejecutar **múltiples programas**: Interpretar cada hilo como programa indepe.
- **Reducir el tiempo de ejecución**: En un **programa** compuesto **por múltiples hilos** independientes usar multithreaded → cada thread es parte de un programa paralelo y se interpreta como un programa diferente.

Multithreading: Múltiples hilos comparten los recursos del procesador. *Ya que el procesador está infrautilizado le tiramos más trabajo*. Ahora el procesador debe mantener el estado de cada hilo que ejecute:

- Banco de registros por hilo, Contador de programa (PC) por hilo y Tabla de páginas por hilo.
- La conmutación entre hilos se realiza por hardware (*mucho más rápido que un cambio de contexto del sistema operativo*).

Tipos de sistemas Multithreding

Hay 3 tipos: Multithreading de **grano fino**: Se diseña sobre un procesador en-orden. Multithreading de **grano grueso**: Se diseña sobre un procesador en-orden. Multithreading **simultaneo (SMT)** → Se disena sobre un procesador o-o-o (*out of order*). *Procesadores en orden son lo de toda la vida y los o-o-o son lo out of order de tomasulo.*

Multithreading de grano fino

Conmuta entre hilos en intervalos de grano fino (*por ejemplo 1 ciclo de reloj*), entrelazando la ejecución de los diferentes threads. Generalmente se hace cíclicamente *(round-robin*), ***saltándose los hilos bloqueados***.

Ventajas: Como el tiempo necesario para cambiar de hilo es muy pequeño, **permite ocultar stalls de alta y de baja latencia**. Útil para cargas científicas con cientos de hilos (***datos en memoria***. *A estos se tarda mucho en acceder*).

Inconvenientes: Puede retrasar mucho la ejecución de cada hilo individual (*por ejemplo, algunos procesadores soportan más de 80 threads*) por lo que es algo Inadecuado como procesador de propósito general.

Multithreading de grano grueso

No necesita conmutar entre hilos "Rápidamente" ya que **oculta latencias largas**. Habitualmente, se vacía el pipeline (*con pipeline cortos*) a partir de la instrucción que ha causado el cambio de contexto y procede a buscar instrucciones del hilo entrante. La **conmutación se produce** cuando un hilo no puede avanzar (*por ejemplo fallo de L1*), por lo que se evita retrasar en exceso su ejecución y se mete a otro mientras.

Ventaja: **Aumenta las prestaciones de un procesador** de propósito general con un hardware mínimo.

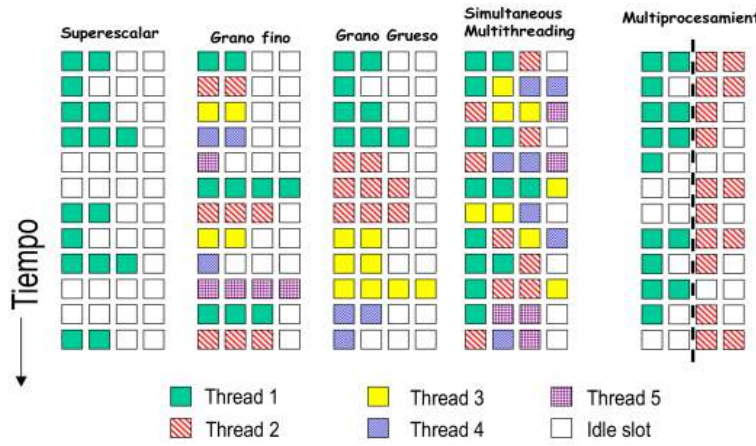
Inconvenientes: Bajas prestaciones respecto a un **procesador** o-o-o. Como el tiempo de cambio de hilo requiere múltiples ciclos, **no elimina la penalización debido a stalls cortos**

Este lo que hace es que solo hace el salto **cuando alguno de los hilos tarda mucho en hacer algo** y se tiene que bloquear mucho. Si se bloquea muy poco no hace nada y a diferencia de el de grano fino se espera.

Multithreading simultaneo (*SMT*) *Suck My Tities*

A **partir de un procesador o-o-o le tiro instrucciones de varios hilos**, por lo que al tirar (*con m = 4 por ejemplo*) 4 instrucciones a ejecución, a lo mejor 1 es del hilo 0, 2 del hilo 1 y otra del hilo 3… Tendremos que tener **un ROB por hilo** también (*además de los bancos de registros, contadores de PC etc.*), es importante etiquetar bien cada instrucción para se sepa de qué hilo es y cual es (ej 1.0, 4.3… Del hilo 1 la 0, del hilo 4 la 3…), pero lo tienes perfe.

A esto se le llama también "hyperThreding". Lo que se puede hacer si tienes 4 vías es tirar 2 programas a la vez, hacer que cada procesador "sea como 2 procesadores". Serían los procesadores "virtuales". Que no escalan tan bien.



Procesador virtual. Solo hay un procesador, pero como puede ejecutar 2 hilos a la vez, parece que sean 2 (aun que las prestaciones se reducen)

PROCESADORES VLIW (*Very Long Instruction Word*)

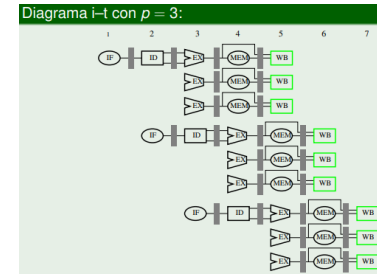
Ahora queremos **reduces la cantidad de instrucciones utilizadas**. *Reducir otro de los 3 parámetros que calculan el Tejecución*. Que **cada instrucción sea más grande** contenga varias "mini-instrucciones" que son una de las convencionales (p). Si **P = 5** podría ser que: *memoria, memoria, CF, CF, Entera/Salto.*

Problema: SI todas las instrucciones que haces son Enteras te quedas igual y no ganas nada. Pero si el compilador puede separar todas las instrucciones que tengas en 5 que entren bien "l" será 5 veces más rápido.

Implicaciones

Ahora **buscas una instrucción**, la **decodificas** y **decodificarla significa decodificar las 5 instrucciones** ahí todas boom. El hardware NO comprueba ninguna dependencia, será el compilador el que se encargue de eliminarlas. Sino puede eliminar alguna dependencia pone nops (*Bajan las prestaciones*). Mu chulo y tal pero una kk, solo llego a fracaso.

El compilador extrae el paralelismo a nivel de instrucciones, empaquetando varias operaciones independientes (*o NOPs, si no las encuentra*). Son técnicas de compilación especiales.



No son compatibles a nivel binario con las maquina escalar departida, ni con otras VLIW que tengan distinto hardware. Y Si el compilador no es muy bueno o el Código tiene ILP bajo *(todas las instrucciones son del mismo tipo o mucha dependencia…*), el tamaño del código generado es mucho mayor al convencional.

PROCESADORES SUPERSEGMENTADOS

Supersegmentar es ya **las etapas que teníamos**, pues esas, segmentarlas más aún. Con esto conseguimos que el tiempo de ciclo sea más pequeño. Si se segmenta todo en 3 etapas (t = 3), TC' = TC/T.

Implicaciones: **No necesita replicar unidades de ejecución**, pero hay que realizar una segmentación más "compleja". Y como hemos aumentado la frecuencia hay mayor sobrecarga potencial de los registros intermedios y del desfase del reloj.

Puedes no supersegmentar todo, sino solo las que sean más lenta / cuello de botella.