Introducción a los procesadores ILP (*Instruction-Level Parallel*)

- Herramientas básicas para **conseguir paralelismo entre instrucciones**:
  - **Encauzamiento** (*pipelining*)
    - ✓ Se usa en todo procesador ILP
  - Utilizar **varias unidades de ejecución**
    - ✓ Para conseguir aún mayor paralelismo
    - ✓ Se emiten varias instrucciones a la vez
      - Para que las unidades de ejecución estén paradas lo menos posible

Tipos de procesadores ILP

- Escalares (**encauzados** o **superencauzados**)
  - Se emite **una instrucción en cada ciclo** de reloj (salvo si lo impiden los riesgos)
- **VLIW** (*Very Long Instruction Word*)
  - **Varias unidades de ejecución**
  - **Cada instrucción codifica varias operaciones**
  - Se emite **una instrucción** (equivalente a **varias instrucciones** de un escalar) **por ciclo**
- **Superescalares**
  - **Varias unidades de ejecución**
  - Se emiten **varias instrucciones por ciclo** (salvo si lo impiden los riesgos)

Limitaciones al paralelismo de instrucciones

- **Limitaciones** de **recursos**
- Necesidad de preservar la **consistencia secuencial de la ejecución** de los programas
  - Ejecución paralela distinta a la secuencial =>
    - ✓ Hay que tomar medidas para que **los resultados de ambos tipos de ejecución sean los mismos**, es decir:
    - ✓ La ejecución paralela ha de ser consistente con la secuencial

Grados de consistencia secuencial

- Fuerte
  - La **secuencia** de operaciones (e interrupciones) se mantiene **igual** que en la ejecución secuencial
- Débil
  - La **secuencia** de operaciones **puede diferir** de la secuencial siempre que no se pierda la consistencia
    - ✓ Ej.: la secuencia de instrucciones original (I;J) puede cambiarse por (J;I) si no hay ningún tipo de **dependencias** entre I y J

Tipos de consistencia secuencial

- Del procesamiento de las **interrupciones**
  - Fuerte : interrupciones precisas
  - Débil: interrupciones imprecisas
- Del procesamiento de las **instrucciones**
  - **Consistencia del procesador**
    - ✓ Fuerte: las instrucciones finalizan en orden (lo **más común**)
    - ✓ Débil: las instrucciones finalizan fuera de orden pero **respetando las dependencias**
  - **Consistencia de memoria**
    - ✓ Fuerte: accesos a memoria (*loads* y *stores*) en orden secuencial
    - ✓ Débil: no se **respeta** el orden secuencial pero sí **las dependencias** (opción **más común**)

Dependencias entre instrucciones

- **Estructurales** o de recursos
  - Los avances tecnológicos permiten reducirlas
- De **control**
  - Un **obstáculo fundamental** para aumentar el paralelismo en los procesadores ILP
- De **datos** (RAW, WAR, WAW)
  - Sin considerar los bucles
  - ***Loop-carried* o inter-iteraciones**: entre la **ejecución de una instrucción** de un bucle **en una iteración y la ejecución de otra instrucción** del mismo bucle **en otra iteración**

Dependencias de control

- **Mantener** las **dependencias de control no es crítico**. Para que el programa se mantenga correcto **basta con** cumplir **dos propiedades críticas**
  - **Preservar** el **comportamiento** de las **interrupciones**:
    - ✓ Por ejemplo, reordenar la ejecución de las instrucciones no debe causar nuevas interrupciones
      - Así dada la secuencia `[BEQZ R3, L1; DIV R1, R2, R3]` (considerando saltos no retardados), **si colocamos** la instrucción **`DIV` antes** de `BEQZ`, la instrucción **`DIV` puede causar una excepción** (división por cero) **que no causaría si no la movemos**

Dependencias de control (cont.)

- **No cambiar** el **flujo de datos** entre instrucciones que producen resultados e instrucciones que consumen (leen) esos resultados
  - ✓ En

```
            DADDU  R1, R2, R3
            BEQZ     R4, L
            DSUBU  R1, R5, R6
L:          OR         R7, R1, R8
```

  - ✓ El valor de R1 usado por OR depende del camino seguido en la bifurcación
  - ✓ Hay que preservar la dependencia de control de OR y de DSUBU sobre BEQZ para evitar un cambio ilegal en el flujo de datos

Dependencias de datos

- Dependencias **de flujo**, **verdaderas** o RAW
  - Hay **un dato** que ***fluye*** de la primera instrucción a la segunda
  - Son **las más importantes** porque no se pueden eliminar
- Dependencias **falsas o de nombre** (WAR o **antidependencia y** WAW o **dependencia de salida**)
  - Dos instrucciones usan el mismo **registro o zona de memoria**, llamado ***nombre***, pero **no hay flujo** de datos asociado a ese nombre
  - **Pueden eliminarse mediante cambios en los nombres**
    - ✓ Lo puede hacer el hardware o el compilador
    - ✓ Más sencillo el renombrado de registros que el de memoria

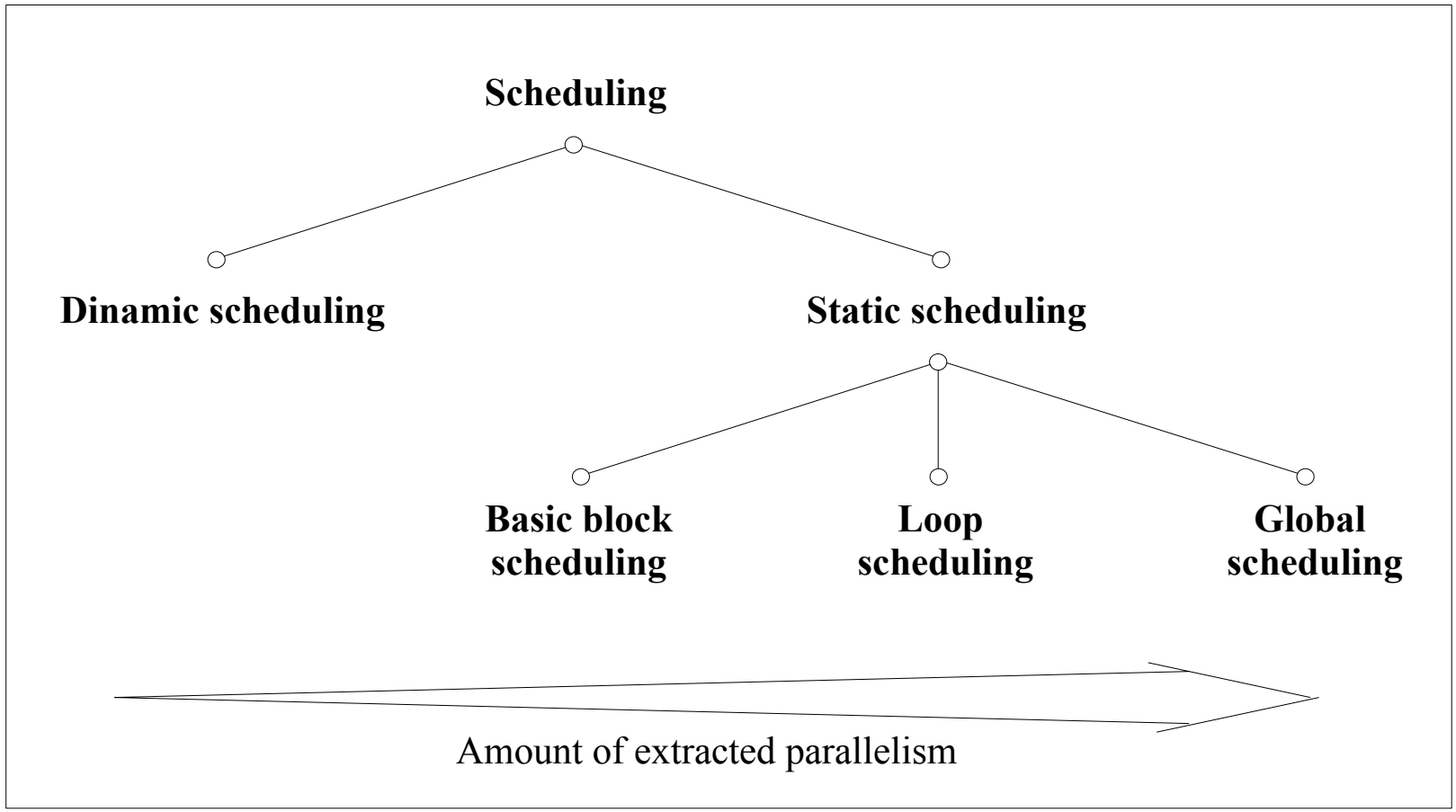
Dependencias inter-iteraciones (*loop-carried*)

- Falsas: se pueden eliminar mediante técnicas de cambio en los nombres
- Verdaderas: en un bucle, hay que **leer un valor que ha sido calculado en una iteración anterior**
  - Ej.:
    ```
    for i:=1 to 90 do
    begin X[i]:= C+Y[i]; Y[i+1]:=Z[i]-V[i] end
    ```
  - El caso más importante es la **recurrencia**
    - ✓ Una **variable** está **definida en función** del valor de esa variable en **iteraciones anteriores**
    - ✓ Ej.: `for i:=1 to 90 do X[i] := X[i-2] + X[i-3]`
  - Estas dependencias suelen **dificultar la paralelización** de los **bucles**

Planificación de instrucciones en procesadores ILP

- Dos **tareas básicas**
  - Detección y **resolución de dependencias**
  - **Extracción** del **paralelismo** a nivel **de instrucción**
    - ✓ Por ejemplo, cambiar instrucciones de sitio no para resolver dependencias sino para utilizar unidades de ejecución que, en otro caso, estarían paradas
- Dos **modos básicos de planificación**
  - **Estática**: el **compilador resuelve** las dependencias **y optimiza** el código
  - **Dinámica**: el **hardware resuelve** las dependencias

Modos básicos de planificación y cantidad paralelismo extraído



Ventajas y desventajas de la planificación dinámica

- Ventajas
  - En determinadas situaciones se ignora en tiempo de compilación si hay dependencia o no
    - ✓ Ejemplo: X[i]:= Z; A:=X[j+5]+K (hay dependencia si i=j+5)
  - El compilador se simplifica
  - No hay que recompilar el código para que se ejecute eficientemente en otro procesador compatible
  - La predicción dinámica de bifurcaciones es más eficiente
- Desventajas
  - Mayor complejidad del hardware
  - El compilador obtiene más paralelismo porque, además de resolver las dependencias, optimiza el código

Modos de planificación, en la práctica

	Realizado por	¿Cómo recibe el código el procesador?	Procesadores en que se usa
Estática (reforzada por la optimización del código)	Compilador	Libre de dependencias conflictivas y optimizado para su ejecución paralela	**VLIW** y algunos encauzados
Dinámica (pura)	Procesador	No optimizado y con dependencias	Primeros procesadores ILP
Dinámica reforzada por el compilador	**Procesador y compilador**	Optimizado para ejecución paralela pero con dependencias	**Casi todos los procesadores encauzados y superescalares**

Técnicas de extracción del paralelismo y CPI

- **CPI = CPI ideal + paradas estructurales + paradas por dependencias de datos (RAW+WAR+WAW) + paradas de control**

Técnica	Término que reduce
Planificación estática de bloques básicos	Paradas RAW
Desenrollamiento de bucles	Paradas control
Encauzamiento software y planificación de trazas	Paradas dep. datos y CPI ideal
Emisión de varias instrucciones por ciclo	CPI ideal
Planificación dinámica con renombrado	Paradas dependencias datos
Predicción dinámica de bifurcaciones	Paradas control
Planificación dinámica con *scoreboarding*	Paradas RAW
Especulación hardware	Paradas datos y control