Superescalares

Scheduling dinámico: algoritmo de Tomasulo

Introducción

- Scheduling dinámico significa que la CPU puede reordenar las instrucciones.
- La mayoría de las CPUs de escritorio son superescalares con scheduling dinámico.
- La mayoría emplean alguna variante de un método de scheduling dinámico conocido como “algoritmo de Tomasulo.”
- Desarrollado por Robert Tomasulo para el mainframe IBM 360/91.

Características

- No permite la ejecución de una instrucción mientras sus operandos no están listos.
- Así se eliminan los peligros RAW.
- Utiliza estaciones de reserva para renombrar registros y eliminar los peligros WAR y WAW.
- Permite especulación dinámica de brincos.
- Utiliza un ROB (reorder buffer) para guardar las instrucciones mientras se comprometen.

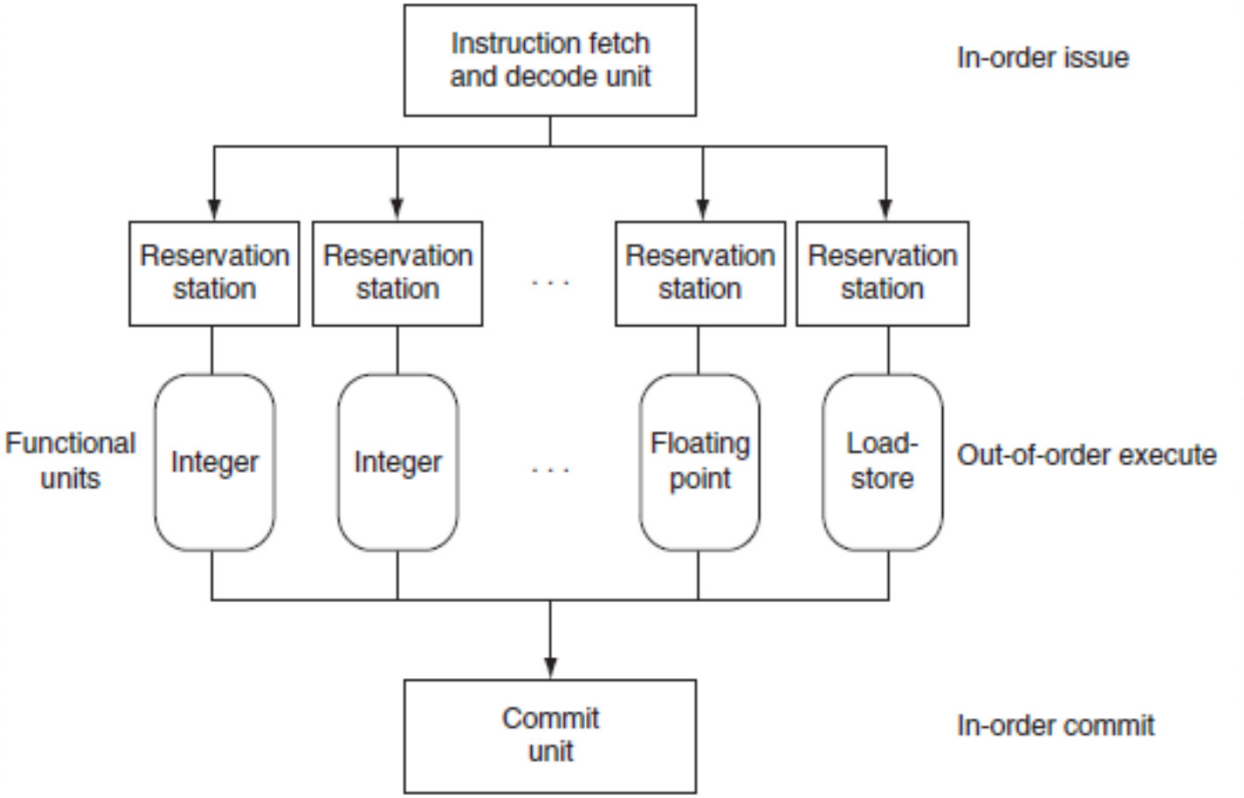
Características

- Cuando una instrucción se compromete, se le permite actualizar el banco de registros o la memoria de datos.
- Permite que las instrucciones se ejecuten en desorden.
- La emisión de instrucciones y el compromiso se hacen en orden.

Estaciones de reserva

- Reservation stations (RS).
- Mecanismo para renombrar registros.
- Guarda operandos tan pronto están disponibles.
- Las instrucciones con dependencias apuntan a la RS de donde obtendrán sus operandos.
- Conforme las instrucciones se emiten, los registros de los operandos pendientes son renombrados por una RS.
- Generalmente, hay más RS que registros físicos.

Estaciones de reserva



Fuente: COD 5, p. 340.

Estaciones de reserva

- Cada RS tiene 7 campos:
  - Op – La operación a realizar en los operandos S_1 y S_2.
  - Q_j, Q_k – Las RS que producirán los operandos. Un 0 indica que los operandos ya están en V_j o V_k, o que son innecesarios.
  - V_j, V_k – El valor de los operandos. Para loads, V_k se usa para guardar el offset.
  - A – Guarda información de la dirección de memoria para una carga o store. Inicialmente, el offset se guarda aquí. Después del cálculo de la dirección, la dirección efectiva se guarda en A.

Estaciones de reserva

- Busy – Indica que la RS y la correspondiente unidad funcional están ocupadas.

- Además, el banco de registros tiene 1 campo extra:
  - Q_i – El número de RS que contiene la operación cuyo resultado va a ser guardado en este registro. Si Q_i es 0, no hay ninguna instrucción activa que vaya a guardar algo en este registro.

Bus de datos común

- CDB – Common Data Bus.
- Permite pasar el resultado de una instrucción, antes de que se comprometa, a otras instrucciones que necesiten ese valor.

Compromiso

- Las instrucciones pasan por una etapa llamada *compromiso* (commit).
- Para poder comprometerse, una instrucción debe ser no especulada y estar al frente del ROB.
- Sólo entonces se le permite escribir al banco de registros o a la memoria.
- Las instrucciones de ejecutan fuera de orden pero se comprometen en orden.

ROB

- El ROB (reorder buffer) es una cola FIFO.
- El ROB mantiene el resultado de una instrucción desde que termina hasta que se compromete.
- Como ni la memoria ni el banco de registros se actualizan hasta que cada instrucción se compromete, el ROB provee operandos a las instrucciones que dependendan de instrucciones que están dentro del ROB.

Instrucciones de memoria

- Las cargas y los stores se hacen en dos pasos:

1. La dirección efectiva se calcula cuando el registro base está listo y luego se guarda en la entrada del ROB correspondiente.
2. Las cargas accesan la memoria tan pronto la unidad de memoria esté lista. Los stores pueden tener que esperar el registro fuente.

- Para evitar peligros los loads y stores se hacen en orden hasta el paso 1.

Instrucciones de memoria

- Las cargas actualizan el registro destino al comprometerse.
- Los stores actualizan la memoria al comprometerse.

Algoritmo de Tomasulo

- Las instrucciones pasan por 4 etapas:
  1. Emisión (issue).
  2. Ejecución.
  3. Escribir resultados en el ROB.
  4. Compromiso (escribir en los registros / memoria).
- Cada etapa tarda un número arbitrario de ciclos.

Algoritmo de Tomasulo

1. Emisión.
   - Obtiene una instrucción de la cola de instrucciones.
   - La instrucción se emite si hay una RS libre y espacio en el ROB donde poner el resultado.
   - Si uno o los dos operandos están disponibles en el banco de registros o en el ROB se copian los valores en los campos correspondientes.
   - Si uno o los dos operandos dependen de otra u otras instrucciones se marcan las dependencias en los campos correspondientes.
   - Si todas las RS están ocupadas o el ROB está lleno, la instrucción se detiene (stall).

Algoritmo de Tomasulo

2. Ejecución.
   - La instrucción se puede ejecutar si sus operandos están disponibles en la RS.
   - En otro caso, el CDB se monitorea esperando que los operandos sean calculados.
   - Este paso elimina los peligros RAW.
   - Las cargas (lw) hacen dos pasos: 1) cálculo de la dirección; 2) lectura de la memoria de datos.

Algoritmo de Tomasulo

- Los stores solo calculan la dirección efectiva en esta etapa.

Algoritmo de Tomasulo

3. Escribir resultados.
   - Al terminar la ejecución, el resultado se escribe en el ROB y en el CDB para uso de otras RS que requieran el valor.
   - La RS se marca como libre.
   - Los stores necesitan atención especial:
     - Si el valor a guardar está libre, se escribe en el ROB.
     - Si no está disponible, el CDB se monitorea hasta que el valor aparezca y entonces se escribe en el ROB.

Algoritmo de Tomasulo

4. Compromiso.
   - Una instrucción puede comprometerse cuando llega al frente del ROB.
   - Las acciones dependen del tipo de instrucción.
   - Si es un store, la memoria se actualiza.
   - Si es otra instrucción (excepto un brinco), el banco de registros se actualiza.
   - Si es un brinco mal adivinado, el ROB se limpia y la ejecución comienza en el lugar correcto.
   - Si es un brinco bien adivinado no pasa nada.
   - En cualquier caso, la entrada del ROB se libera.

Ejemplo

- Mostrar el status de las tablas cuando el mul.d está listo para comprometerse.

1. l.d f6, 32(r2) **# f6 = Mem[r2 + 32]**
2. l.d f2, 44(r3) **# f2 = Mem[r3 + 44]**
3. mul.d f0, f2, f4 **# f0 = f2 * f4**
4. sub.d f8, f2, f6 **# f8 = f2 – f6**
5. div.d f10, f0, f6 **# f10 = f0 / f6**
6. add.d f6, f8, f2 **# f6 = f8 + f2**

ROB

	Reorder buffer					
Entry	Busy	Instruction		State	Destination	Value
1	no	L.D	F6,32(R2)	Commit	F6	Mem[32 + Regs[R2]]
2	no	L.D	F2,44(R3)	Commit	F2	Mem[44 + Regs[R3]]
3	yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	yes	DIV.D	F10,F0,F6	Execute	F10	
6	yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Estaciones de reserva

	Reservation stations							
Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[44 + Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[32 + Regs[R2]]	#3		#5	

Banco de registros

	FP register status									
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F10**
Reorder #	3						6		4	5
Busy	yes	no	no	no	no	no	yes	. . .	yes	yes

Ejemplo

- Mostrar la ejecución del siguiente código:

```
1. loop: l.d f0, 0(r1)          # f0 = Mem[r1]
2.       mul.d f4, f0, f2       # f4 = f0 * f2
3.       s.d f4, 0(r1)          # Mem[r1] = f4
4.       daddiu r1, r1, #-8     # r1 = r1 – 8
5.       bne r1, r2, loop       # if r1 ≠ r2 goto loop
```

Ejemplo

- Suponiendo que:
  - Se especula que el ciclo se toma.
  - Se han emitido las instrucciones en el ciclo 2 veces.
  - El l.d y mul.d de la primera iteración ya se comprometieron y las demás instrucciones terminaron de ejecutarse.

ROB

	Reorder buffer				
Entry	Busy	Instruction	State	Destination	Value
1	no	L.D F0,0(R1)	Commit	F0	Mem[0 + Regs[R1]]
2	no	MUL.D F4,F0,F2	Commit	F4	#1 × Regs[F2]
3	yes	S.D F4,0(R1)	Write result	0 + Regs[R1]	#2
4	yes	DADDIU R1,R1,#-8	Write result	R1	Regs[R1] – 8
5	yes	BNE R1,R2,Loop	Write result		
6	yes	L.D F0,0(R1)	Write result	F0	Mem[#4]
7	yes	MUL.D F4,F0,F2	Write result	F4	#6 × Regs[F2]
8	yes	S.D F4,0(R1)	Write result	0 + #4	#7
9	yes	DADDIU R1,R1,#-8	Write result	R1	#4 – 8
10	yes	BNE R1,R2,Loop	Write result		

Banco de registros

- Las RS no se muestran porque no hay instrucciones ejecutando.

	FP register status								
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8
Reorder #	6				7				
Busy	yes	no	no	no	yes	no	no	...	no

Ejemplo

- Suponer que el primer brinco no debió ser tomado.
- Las instrucciones anteriores al brinco se comprometen.
- Cuando el brinco llega al frente, el ROB se limpia y se comienzan a sacar instrucciones del camino correcto.
- En la práctica, al momento de saber que se adivinó mal se limpia la parte del ROB que corresponde al camino equivocado.

Conclusiones

- El algoritmo de Tomasulo se inventó para la IBM 360/91 (1960s), pero solo fue práctico de implementar en microprocesadores hasta principios de los 90s.
- Ahora ha sido ampliamente adoptado en los procesadores de emisión múltiple.
- Consigue un gran rendimiento sin requerir que el compilador sea adaptado para cada arquitectura.
- Los peligros WAW y WAR se eliminan porque las instrucciones se comprometen en orden.

Conclusiones

- Los peligros RAW se eliminan al no permitir que una instrucción continúe hasta que sus operandos están listos.
- Los peligros RAW en la memoria se eliminan ejecutando las cargas y los stores en orden (no especulación).
- El scheduling dinámico es la forma preferida de scheduling usada actualmente en las CPUs de escritorio.