

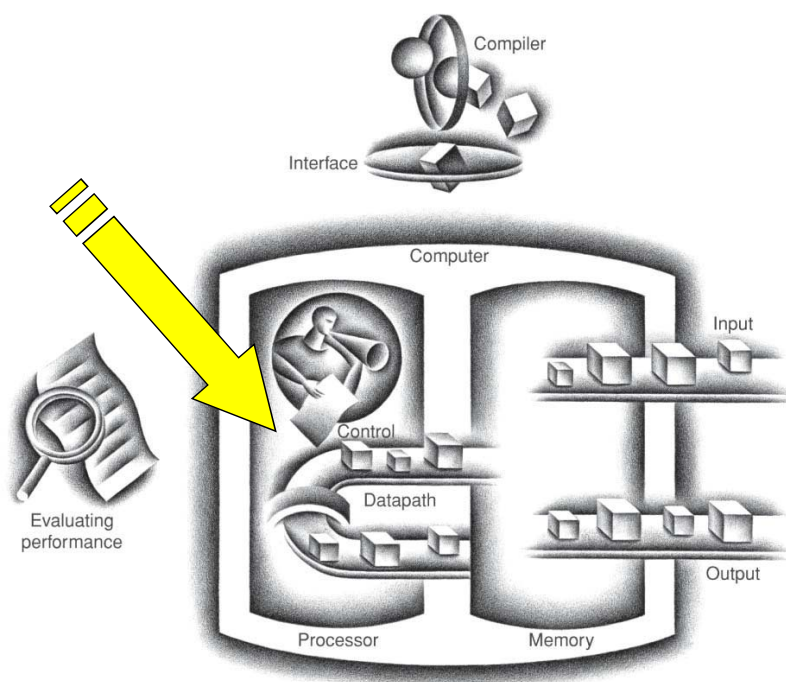
Technische Informatik 1

10. Prozessor - Instruktionsparallelität

© Lothar Thiele

Wo sind wir ?

Kapitel 4
Patterson/Hennessy

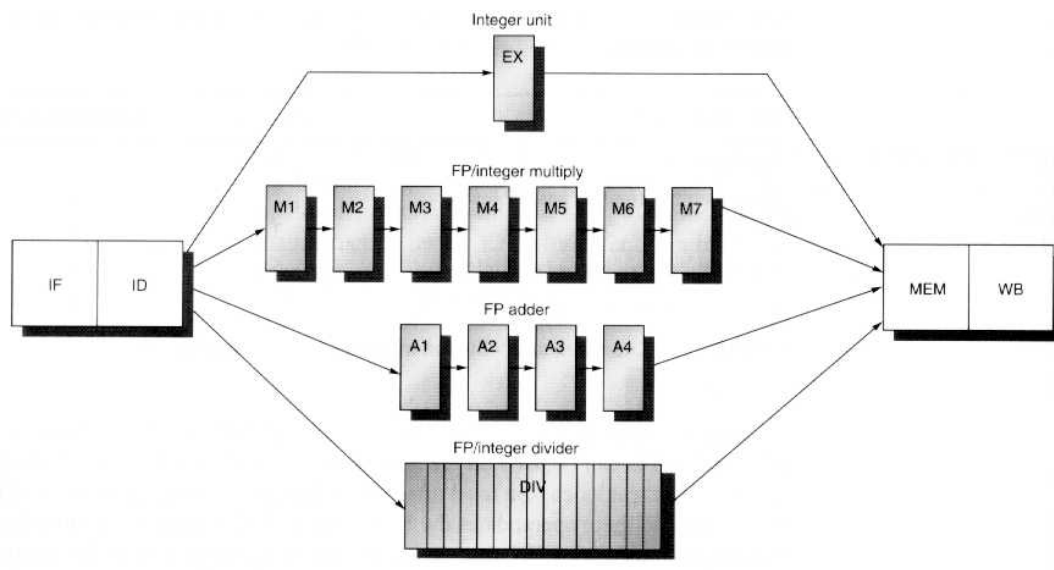


Instruktionsparallelität

- ▶ Instruction-Level Parallelism (ILP)
- ▶ Pipelining: Mehrere Instruktionen werden gleichzeitig bearbeitet.
- ▶ Vergrößerung der Instruktionsparallelität:
 - *Mehr Pipelinestufen, tiefere Pipeline*: weniger Rechenzeit pro Stufe und daher kürzere Taktperiode
 - *Mehrere Instruktionen pro Takt laden (multiple issue)*:
 - Vervielfältigung der Pipelinestufen
 - Gleichzeitiger Start mehrerer Instruktionen in einem Takt
 - Im Prinzip: $CPI < 1$, aber Abhängigkeiten zwischen Instruktionen vergrößern den CPI-Wert

Tiefere Pipeline - Superpipelining

Einführung zahlreicher Pipelinestufen, vor allem bei den arithmetischen Einheiten.



Tiefere Pipeline - Superpipelining

► *Konsequenzen:*

- Höhere Taktfrequenz durch geringere Laufzeit zwischen Pipelinestufen.
- Die Zahl der Stufen ist je nach Instruktion unterschiedlich lang. Daher werden sie zu unterschiedlichen Zeiten beendet („*out of order completion*“).
- Der Einfluss von Hazards auf die Ausführungszeit von Programmen wird grösser.

MULTD	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
ADDD		IF	ID	A1	A2	A3	A4	MEM	WB		
LD			IF	ID	EX	MEM	WB				
SD				IF	ID	EX	MEM	WB			

Multiple Issue

► *Statische Parallelität (Static Multiple Issue)*

- Compiler gruppiert Instruktionen, die gleichzeitig geladen werden sollen (VLIW – very long instruction word).
- Compiler detektiert und verhindert Hazards

► *Dynamische Parallelität (Dynamic Multiple Issue)*

- CPU lädt aufeinanderfolgende Instruktionen und bestimmt, welche davon als nächstes ausgeführt werden sollen.
- Compiler kann hierbei Vorarbeit leisten, in dem er Instruktionen umsortiert.
- CPU löst Hazards durch erweiterte Techniken zur Laufzeit auf.

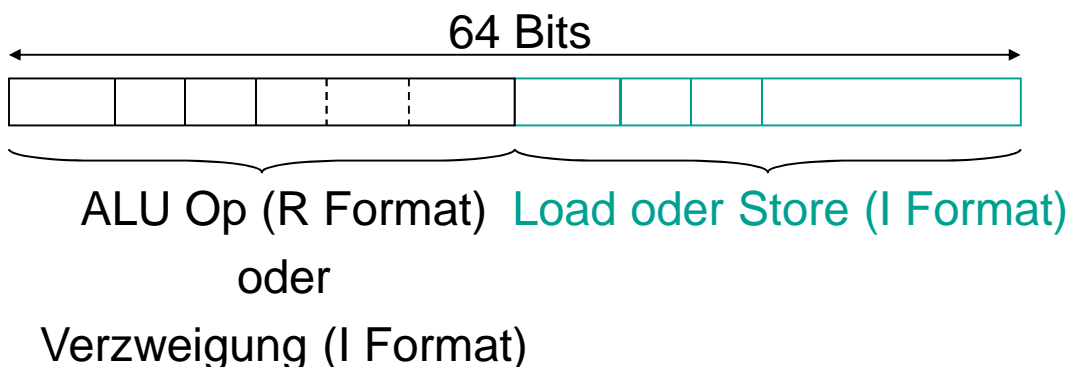
Statische Parallelität

- ▶ In jedem Takt werden mehrere Instruktionen dekodiert.
- ▶ Interne Komponenten werden dupliziert. Es gibt üblicherweise mehrere Arten von arithmetischen Einheiten, zum Beispiel Festkomma-, Gleitkomma-Einheiten, Multiplizierer, Dividierer.

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

Beispiel: VLIW MIPS

- ▶ MIPS Architektur mit einem doppelten Instruktionswort:



- ▶ Instruktionen werden immer paarweise geladen, dekodiert und ausgeführt.
- ▶ Die Rechnerarchitektur benötigt jetzt ein Registerfeld mit 4 Lese- und 2 Schreibzugriffen sowie einen separaten Addierer für Speicheradressen.

Compiler-Techniken

	ALU oder Verzweigung	Datentransfer	CC
lp:		lw \$t0, 0(\$s1)	1
	addi \$s1, \$s1, -4		2
	addu \$t0, \$t0, \$s2		3
	bne \$s1, \$0, lp	sw \$t0, 4(\$s1)	4

- ▶ 4 Taktzyklen für 5 Instruktionen
 - CPI = 0.8 (bester Fall wäre CPI = 0.5)
- ▶ Verbesserung durch *Schleifenentfaltung* (loop unrolling)
 - Replizieren des Schleifenkörpers so dass Instruktionen von unterschiedlichen Iterationen kombiniert werden können.

Compiler-Techniken

Schleife 4 mal entfalten (in unserem Beispiel):

```

lp:    lw    $t0, 0($s1)    # $t0=array element
        lw    $t1, -4($s1) # $t1=array element
        lw    $t2, -8($s1) # $t2=array element
        lw    $t3, -12($s1) # $t3=array element
        addu  $t0, $t0, $s2 # add scalar in $s2
        addu  $t1, $t1, $s2 # add scalar in $s2
        addu  $t2, $t2, $s2 # add scalar in $s2
        addu  $t3, $t3, $s2 # add scalar in $s2
        sw    $t0, 0($s1)  # store result
        sw    $t1, -4($s1) # store result
        sw    $t2, -8($s1) # store result
        sw    $t3, -12($s1) # store result
        addi  $s1, $s1, -16 # decrement pointer
        bne   $s1, $0, lp   # branch if $s1 != 0
    
```

Register umbenennen
um möglichst viele
Abhängigkeiten
zwischen Instruktionen
zu entfernen.

Nur ein Schleifentest
anstelle von 4 Tests.

Compiler-Techniken

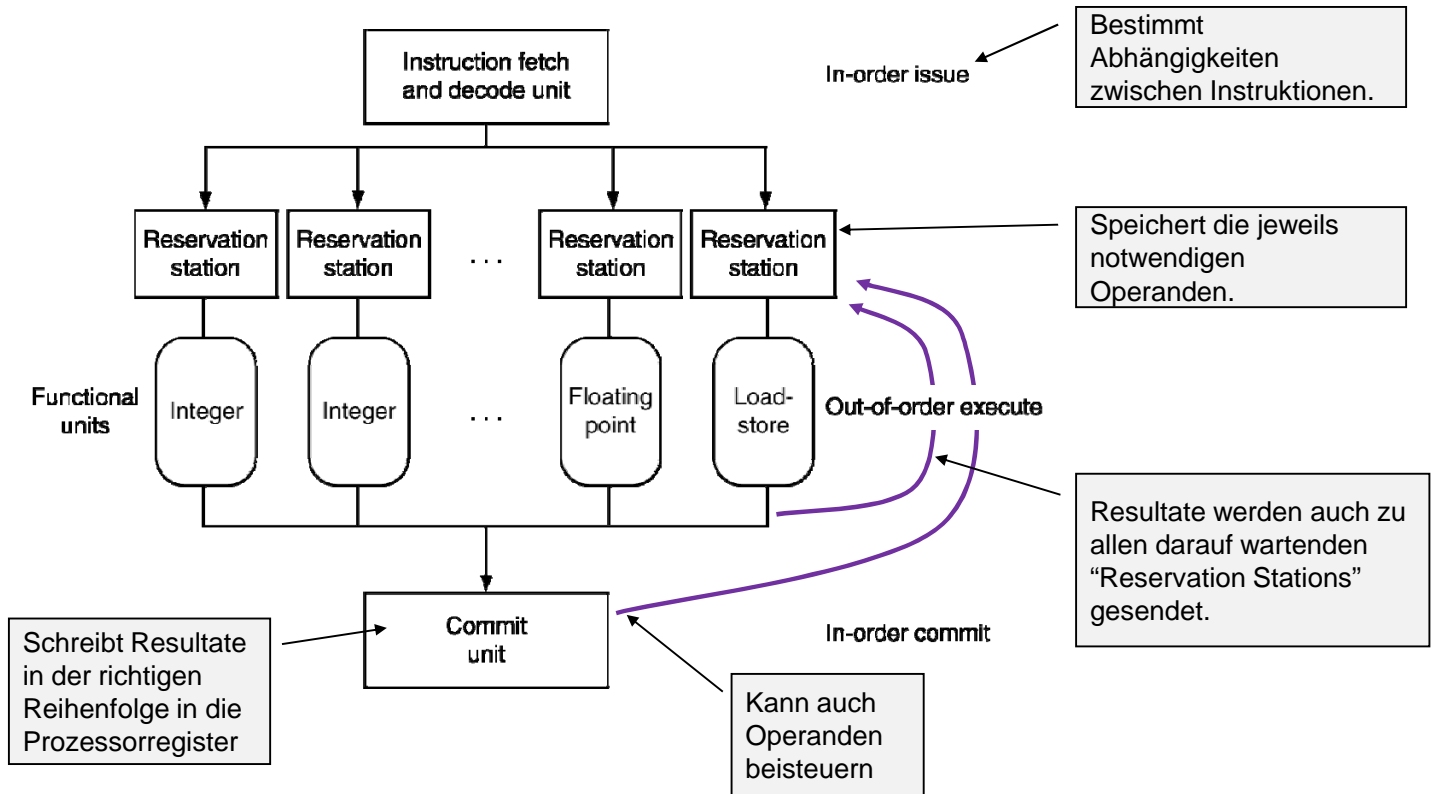
	ALU oder Verzweigung	Datentransfer	CC
lp:	addi \$s1, \$s1, -16	lw \$t0, 0(\$s1)	1
		lw \$t1, 12(\$s1)	2
	addu \$t0, \$t0, \$s2	lw \$t2, 8(\$s1)	3
	addu \$t1, \$t1, \$s2	lw \$t3, 4(\$s1)	4
	addu \$t2, \$t2, \$s2	sw \$t0, 16(\$s1)	5
	addu \$t3, \$t3, \$s2	sw \$t1, 12(\$s1)	6
		sw \$t2, 8(\$s1)	7
	bne \$s1, \$0, lp	sw \$t3, 4(\$s1)	8

- ▶ 8 Taktzyklen um 14 Instruktionen auszuführen:
 - CPI = 0.57 (gegenüber dem besten Fall CPI = 0.5)

Dynamische Parallelität

- ▶ „Dynamic Multiple Issue“: *Superskalare Prozessoren*
- ▶ CPU entscheidet, wie viele Instruktionen begonnen werden.
- ▶ Die CPU vermeidet *zur Laufzeit* (dynamisch) strukturelle Hazards und Datenhazards und optimiert die Parallelität:
 - Instruktionen werden nicht in der gegebenen Reihenfolge ausgeführt, aber die Ergebnisse werden in der richtigen Reihenfolge geschrieben.
 - Umsortierung von Instruktionen durch den Compiler nicht mehr notwendig (aber zuweilen hilfreich).
 - Weitere Techniken zur Verbesserung der Ausführungszeit:
 - Spekulation
 - Register-Umbenennung

Superskalare CPU



Register Umbenennung

- ▶ „Commit Unit“ und „Reservation Stations“ ordnen den *logischen Registern* (des Maschinenprogramms) *freie physikalische Register* zu.
- ▶ Für eine Instruktion in einer „Reservation Station“:
 - Falls der Operand im Registerfeld oder in der „Commit Unit“ vorhanden ist, wird es in die „Reservation Station“ kopiert. Falls es nicht mehr benötigt wird, kann es im Registerfeld überschrieben werden.
 - Falls der Operand noch nicht vorhanden ist, wird auf die generierende Funktionseinheit gewartet. Eventuell muss dann das Datum gar nicht in das Registerfeld geschrieben werden.

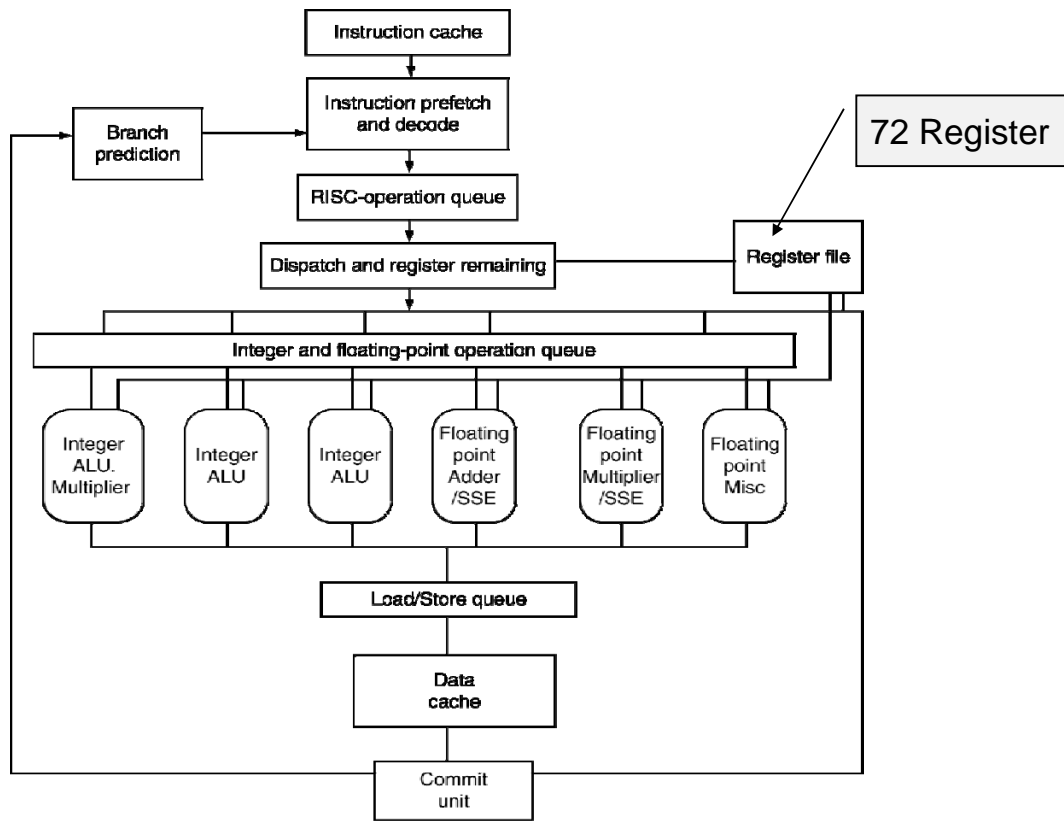
Spekulation

- ▶ „*Schätzen*“, was mit einer Instruktion geschehen soll:
 - *Starte* die Operation so bald wie möglich
 - *Prüfe*, ob die Entscheidung korrekt war:
 - Falls ja, führe die Instruktion aus.
 - Falls nein, stelle den alten Systemzustand wieder her.
- ▶ Wird sowohl bei statischer als auch bei dynamischer Parallelität benutzt.
- ▶ *Beispiele*:
 - Spekuliere auf das Ergebnis einer bedingten Verzweigung.
 - Spekuliere auf die Adresse und das Ergebnis einer Ladeinstruktion

Spekulation

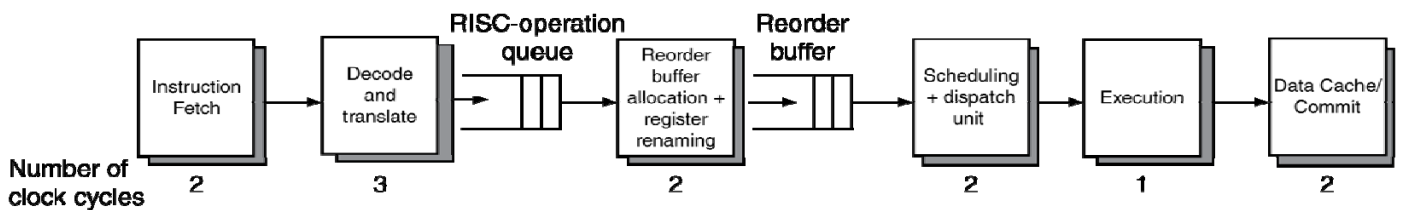
- ▶ *Compiler* kann Instruktionen umordnen (z.B. \perp_w Instruktion vor eine bedingte Verzweigung bewegen):
 - Eventuell müssen dann Instruktionen eingefügt werden, die bei einer falschen Vorhersage das Resultat korrigieren.
- ▶ *Hardware* kann Instruktionen vorziehen:
 - Resultate werden zwischengespeichert, bis sie tatsächlich benötigt werden und die richtige Schätzung bestätigt wurde. Erst dann werden sie in das Registerfeld eingespeichert („Commit Unit“).
 - Bei einer falschen Vorhersage werden die Zwischenergebnisse gelöscht.

Die Opteron X4 Architektur



Die Opteron X4 Pipeline

► Pipeline für ganzzahlige Operationen:



- Gleitkomma-Pipeline ist 5 Stufen länger
- Bis zu 106 RISC-Operationen gleichzeitig in Verarbeitung

■ Probleme:

- Komplexe Instruktionen mit weit reichenden Datenabhängigkeiten
- Verzweigungsvorhersage
- Latenz von Speicherzugriffen