2.7. Out-of-Order-Architektur

F18: Struktur

3 Teile
IF
ID & RR
Teil 1: in order (Reihenfolge, wie Befehle im Speicher stehen)
Instruktion Pool: Gelesene und Dekodierte Befehle, die ausgeführt werden könnten, wenn ihre Eingangsdaten verfügbar wären)
Papallele EX-Einheiten zur parallelen Ausführung
Teil 2: out of order
Reorder Buffer: Enthält ausgeführte Befehle, die noch nicht vollständig beendet sind, wieder in die Reihenfolge Umsortierung, wie die Befehle gelesen wurden (wie sie im Speicher standen)
WB
Teil 3: in order

Grundprinzip:
Bei spezialisierten EX-Einheiten und In-Order-Ausführen, kann es vorkommen, dass von den EX-Einheiten einige nicht ständig ausgelastet sind -> langsamer bei der Programmabarbeitung.

Bei Out-of-Order werden Befehle vorgezogen, die diese freien EX-Einheiten belegen
(in F18 durch Scheduler und Retierement Unit realisiert, die Befehle aus dem Instruction Pool auswählt (zusätzliche Restriktionen müssen beachtet werden) -> vorgezogene Befehle dürfen nicht datenabhängig von noch nicht ausgeführten Befehlen sein.
Write Back erfolgt wieder In Order, Ergebnisse der EX-Einheiten werden im Reorder Buffer gesammelt und WB In Order zugeführt.
F 21,22 enthält eine zeitlichen Vergleich von superscalarem Prozessor mit zwei EX-Einheiten und einem Out-of-Orderprozessor mit zei EX-Einheiten.

Ablauf in Folie 21 (superscalar)

4 Bsp.-Befehle:
Move: Datentransport zwischen Registern (Prozessor-intern)
ADD: Addition von Registern, Ergebnis im Register
Beide sind 1-Takt-Befehle
Store: Transport vom Prozessor zum Speicher,
an sich wäre es ein 3-Taktbefehl, aber nach Bereitstellung des zu schreibenden Wertes durch den Prozessor nach 1 Takt kann schon der nächste Befehl arbeiten -> wie 1 Taktbefeh
Load: Transport vom Speicher zum Prozessor: 3 Taktbefehl, da die benötigten Daten erst am Ende des Befehls verfügbar sind

Oben rechts: Beispielprogrammstück
Unten links: den dazu gehörenden Zeitverlauf

Ablauf F22
Das gleiche Bsp.-Programm wie Folie 22 mit zusätzlicher Darstellung der Datenabhängigkeiten
Unten links: den dazu gehörigen Zeitverlauf

Auswertung des Zeitverhaltens: Out-of-Order benötigt für deas Beispiel-Programm 5 Takte, Superscalar 10 Takte

Bei Ressoucenabhängigkeiten (Register) wir sogenanntes „Register renamig“ durchgeführt, d.h. das doppelt benötigte Register wird pysisch durch ein anderes freies Register realisiert.

Beschreibung Ablauf Folie 21:

1. B1 und 2 gleichzeitig, warten bis Ende B2 (Load) 3 Takte
2. B3 allein, da B4 die gleiche Ex_Einheit braucht und Daten vom Befehl 3 1 Takt
3. B4 B4 geichzeitig 1 Takt
4. B6 B7 gleichzeitig Warten auf Ende von Load 3 Takte

5. B8 B9 gleichzeitig 1 Takt
6. B10 ohne Folgebefehl (bekannt)

Beschreibung Ablauf F23:

1. B1 und B2 starten gleichzeitig B1 Ende 1 Takt B2 Ende 3 Takte
2. Da B1 nach 1 Takt fertig und B2 benötigt keine EX-Einheit vorziehen von B5 und B6 (nicht datenabhängig von B2 B3 B4) Ende nach einem Takt
3. Wieder zwie frei Einheiten vorziehen von B6 B7 B8, B7 ist zar Datenabhängig von B5, dieser ist aber schon fertig
4. Jetzt Abarbeitung B4 möglich, parallel B6 beenden und B9 (keine Datenabhängigkeiten
5. Parallel B4 und B10, B4 war datenabhängig von B3, aber der ist fertig B10 war datenabhängig von B8, B9, aber diese sind auch fertig B4 und B10 sind Res.abhängig (Register A) -> Register-Renaming

2.7. Simultaneous Multithreding (SMT)

Struktur F20

Grundidee: Programm lässt sich in mehrere, weitgehend unabhägige Threads zerlegen (Programmteile, die parallel ausgeführt werden känne. Die Treads durchlaufen entsprechend der maximal parallel möglichen Anzahl (in F20 zwei) parallel IF,ID & RR bis zum Instruction Pool. Hier kann das Umsotieren auf Befehle der verschiedenen Threads zugriefen, die aufgrundihrere Thread-Eigenschaft unabhängig von einenader sind. Damit erfolgt eine weitere Verbesserung der Auslastung der Exeinheiten.

Möglicher Ablauf:

1. Für beide Threads paralel IF und danach ID und RR (Register renaming)
2. Ablegen der Befehle von beiden Gelesenen und dekodierten Befehlen im Instruction Pool

3. Auswahl der unabhängigen Befehle zur Abarbeitung (Unabhängigkeit im Thread wie bei Out of Order, zwischen den Threads sowieso gegeben)
4. Nach Abarbeitung der Befehle deren Ergebnisse im Reorderbuffer.
5. WB zugeordnet zu den Threads und in der Reihenfolge des Lesens der entsprechenden Befehle

Vergleich Superscalar, Out of Otrder VLIW -> F23
Spekulative Ausführung von Befehlen

- Ausführung eines Befehls auch wenn die Datenabhängigkeit noch nicht bekannt ist, benutzung des Ergebnisses aber erst nach Kennen der Datanabhängigkeit wenn keine da war) und erneute Ausführung des befehls, falls eine da war