Schlecht balancierte Pipeline (F13)

> i.a. sind die Phasen eines Befehls (IF, ID, OF, EX, WB) in ihrer Ausführung nicht gleich lang, insbesondere ist die EX-Phase meist länger

Jeder Befehl ist dadurch 5* Zeit von EX-Phase lang, d.h. nach der Zeit einer EX-Phase wirde ein Befehl fertig.

2.5. Superscalare Architektur

Struktur F14

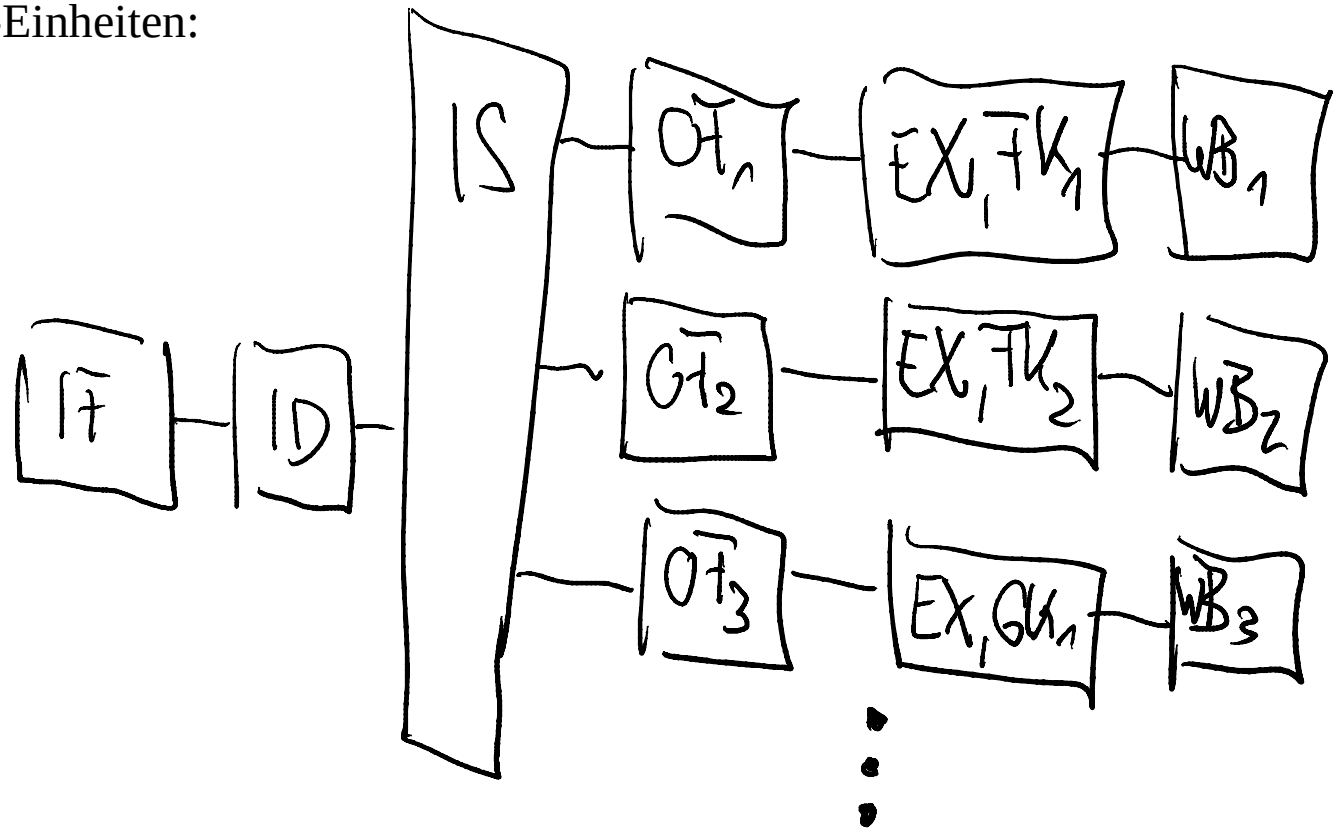
Befehlspipeline,
mit parallelen Zeigen für OF, EX, WB

➔ Auch bei längere EX-Phasen werden mehr Befehle in der gleichen Zeit fertig ohne dass die Phase selbst kürzer wird.

Zusätzlich notwendig: Instruction Scheduler (IS):
Aufteilen der über IF, ID sequentiell kommenden Befehle auf die parallelen Zweige nach zwei Kriterien:
Zweig ist frei (Phase OF)
Bei spezialsierte EX-Einheiten muss der Befehl auf die EX-Einheit passen.
Bsp. Spezialisierte Einheiten: Festkommaeinheit, Gleitkommaeinheit

Problem entsteht, wenn unmittelbar auf einander Befehle folgen, die die gleiche Ex-Einheit benötigen.

Eine sinnvolle Architektur kombiniert parallele gleiche und spezialisierte EE-Einheiten:



Alternative Phasenpipeline (F15):
Zerlegung Instruction Decoder (ID) in sequentiellen (ID1) und parallelen (ID2,1 bis ID2,n) Teil:

- ID1 erkennt die Befehlsgruppe (welche EX-Einheiten sind für den Befehl möglich?)
- ID2,i ermittelt den konkreten Befehlfü Exi (n mal vorhanden)

Effekte: ID ist zeitlich zerlegt und der zeitaufwendigere Teil ist parallelisiert,
7-stfige Pipeline ((hier wird keine Superpipeline zugrunde gelegt)
ID1 und ID2 werden in der Logik kleiner und dadurch auch etwas schneller.

Zeitkritisch in dieser Architektur: IF, da alle Befehle sequentiell gelesen werden müssen (ID1 ist nicht so kritisch, da hier nur die Befehlsgruppe ermittelt wird)

➔ IF schnell: Maßnahmen beim Speicherzugriff notwendig (-> Kap.3)
➔ IF parallel, dabei evtl. Einsparung von ID1

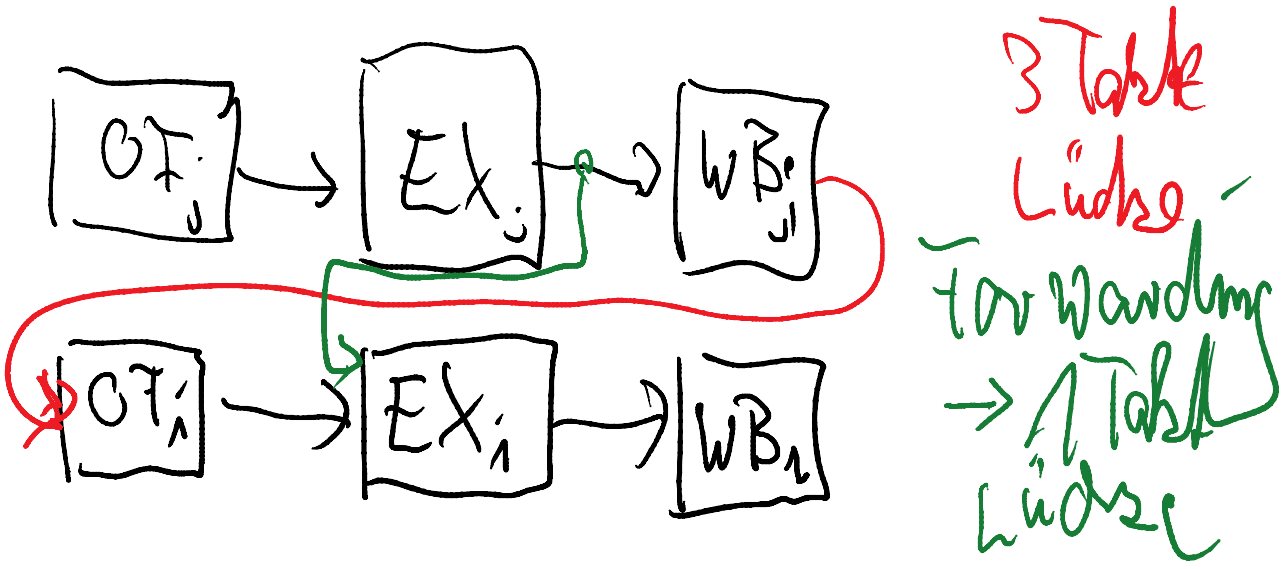
2.6. VLIW-Architetur

VLIW: Very Long Instruction Word
Grundidee: mehrere Befehle werden zu einem langen Befehlswort zusammengefasst, das in einem Schritt gelesen wird.

Alle Befehle lauf gleichzeitig ab. (F16)

Problem: Befehle müssen bei spezialisierten EX-Einheiten so im Speicher stehen, dass sie ohne Umsortieren auf die passende EX-Einheit kommen: Compiler erzeugt diese Reihenfolge aufgrund der Informationen zur Struktur des Prozessors.
(F16) korrigieren Pfeil von EX2 darf nicht zu WB1 sondern muss zu WB2 gehen)

Ablauf bei Datenkonflikt (EX-Einheit i benötigt Ergebnis von EX-Einheit j (i ist nicht j)



Verringerung des Problems: Compiler versucht so eine Reihenfolge zu erzeugen, dass die Datenabhängigkeit möglichst nicht entsteht.

Bsp.:

ADD A,B	Unsortieren->	ADD A,B
ADD C,A		SUB D,E
SUB D,E		ADD C,A

Geht, weil der SUB-Befehl nicht datenabhängig von den beiden ADD-Befehlen ist, die untereinander aber datenabhängig sind.

Problem2: Zwei Befehle benötigen eine EX-Einheit, die spezialisiert ist und nur einmal vorhanden ist.

➔ Falls möglich, Compiler verschiebt den zweiten Befehl bei n Befehlen im langen Befehlswort um n nach hinten (aber nicht immer möglich aufgrund der Datenabhängigkeiten)

Problem3: spezialisierte EX-Einheiten, Compiler kann die Befehle nicht so sortieren, dass für alle EX-Einheiten ein passender Befehl in das lange Befehlswort eingefügt werden kann.

➔ Leerbefehl (NOP, No Operation) einfügen.

Bsp. Zu F17

Befehlwort:

Branch-Befehl, INT-Befehl, Leerbefehl, FP-Befehl

d.h. zu diesem Zeitpunkt ist kein MEM-Befehl verfügbar.

Verringert die Leistung.

Ausweg: VLIW, dynamisch (F17)

Bei fehlendem Befehl wird versucht einen Folgebefehl zu finden, der gleichzeitig mit gelesen werden kann, oder

Im Befehlswort steht der Leerbefehl nicht, aber IQ & MI-Einheit erkenn das und ergänzt den Leerbefehl.

- ➔ Weniger Speicherplatz notwendig
- ➔ Evtl. trotzdem n Befehle einlesen, von denen aber einer erst später ausgeführt wird.