1. Bedingte Sprünge

a) Zeitverlust durch falsche Befehle in der Pipeline

```
Branch    FDX
Post1      FD
Post2       F
```

Im Idealfall wird in jedem Zyklus ein Befehl fertig. Da es ohne Sprünge insgesamt t Zeiteinheiten lang läuft, besteht das Programm also aus t Befehlen.

Wir betrachten die Auslastung der Pipeline, wenn gesprungen wird. Post1 und Post2 sind die beiden Befehle, die sequentiell auf den Sprungbefehl folgen (und also nicht ausgeführt werden sollen); Neu1, Neu2, ... sind die Befehle, die beim Sprungziel beginnen.

	T1	T2	T3	T4	T5	T6	T7	T8
Fetch	Bra	Post1	Post2	Neu1	Neu2	Neu3	Neu4	Neu5
Decode	(...)	Bra	Post1	---	Neu1	Neu2	Neu3	Neu4
Execute	(...)	(...)	Bra	---	---	Neu1	Neu2	Neu3
Memory	(...)	(...)	(...)	Bra	---	---	Neu1	Neu2
WriteBack	(...)	(...)	(...)	(...)	Bra	---	---	Neu1

D. h.: Für jeden Sprungbefehl, der ausgeführt wird, verlängert sich die Gesamtlaufzeit um zwei Zeiteinheiten.

(1) 100% tatsächl. Sprünge: 100% * 20% = 20%, $t_{100} = t + 20\% * 2\ t = 1{,}4\ t$

(2) 50% tatsächl. Sprünge: 50% * 20% = 10%, $t_{50} = t + 10\% * 2\ t = 1{,}2\ t$

(3) 10% tatsächl. Sprünge: 10% * 20% = 2%, $t_{10} = t + 2\% * 2\ t = 1{,}04\ t$

(Den Auf- und Abbau der Pipeline, der zu leichten Korrekturen an der Gesamtlaufzeit führt, vernachlässigen wird – nimmt man an, dass das Programm aus tausenden Instruktionen besteht, spielt das keine Rolle.)

Ist Ihnen die Rechnung mit Prozenten zu unverständlich, gehen Sie davon aus, dass das Programm aus 100 Befehlen besteht: Die brauchen normal ca. 100 Zeiteinheiten. Sind nun 20 Befehle ausgeführte Sprünge, kommen 20 x 2 Zeiteinheiten dazu, in der Summe sind das 140 (Faktor 1,4).

b) MMIX: probable branch vs. normaler branch

Die Probable-branch-Befehle weisen die CPU darauf hin, dass mit großer Wahrscheinlichkeit gesprungen wird (die geprüfte Bedingung also wahr ist). Die CPU kann dann die Pipeline nach dem Sprungbefehl mit den am Sprungziel stehenden Befehlen füllen und muss nur im (unwahrscheinlichen) Fall, dass nicht gesprungen wird, korrigieren.

Das ist z. B. für Schleifen wichtig, in denen man am Schleifenende zum Anfang zurückspringt, sofern die Abbruchbedingung noch nicht erfüllt ist – Sprünge sind hier viel häufiger, weil nur ein einziges Mal (am Ende der Schleife) nicht gesprungen wird.

2. Interrupts und Pipelines

a) Exception status vector:

Ziel ist, bei mehreren Exceptions, die in verschiedenen Befehlen in der Pipeline auftreten, die Exception-Behandlung in der normalen Reihenfolge zu erledigen. Nun kann ein späterer Befehl vor einem früheren Befehl eine Exception verursachen – solange nicht klar ist, dass kein früherer Befehl ebenfalls eine Exception erzeugt, darf diese aber nicht bearbeitet werden (sonst käme es zu einer Vertauschung der Reihenfolge).

Der exception status vector speichert für die Befehle in der Pipeline, ob sie Exceptions verursacht haben oder nicht. Befehle mit Exceptions werden zunächst weiter durch die Pipeline geschleust, es geschieht in den verbleibenden Phasen aber nichts mehr (insbes. wird nichts verändert: kein Schreiben in den Speicher oder in Register). Berechnungen oder Lesezugriffe wären noch möglich, sind aber für diese Befehle nicht mehr hilfreich.

In der Write-Back-Phase kann der Prozessor für einen Befehl mit Eintrag im exception status vector die Exception-Behandlungsroutine starten, denn jetzt ist klar, dass kein älterer Befehl auch eine Exception verursacht hat. (Es gibt in der Pipeline keine älteren Befehle als den, der in der letzten Phase steckt.)

b) Grund für die Verarbeitung der Exceptions in richtiger Reihenfolge

Bei einer Exception sollen alle Befehle vor dem fehlerhaften Befehl noch komplett durch die Pipeline laufen, alle Befehle dahinter werden abgebrochen. Beispiel: Befehle B1 und B2 (in dieser Reihenfolge) lösen beide eine Exception aus, B2 aber zeitlich vor B1.

Zeit	1	2	3	4	5	Art und Konsequenz der Exception
A1	X	M	W			
A2	D	X	M	W		
B1	F	D	X	M☹		(illegal address, Programmabbruch)
B2		F☹				(page fault, Speicherzugriff)

Zum Zeitpunkt 2 (Exception bei B2) ist B1 erst durch die Decode-Phase gelaufen, wird also abgebrochen (nur Befehle in der X- oder M-Phase werden vor der Exception-Behandlung abgearbeitet). Nach Abbruch von B1 wird die Exception von B2 bearbeitet, danach B1 neu gestartet. Im Beispiel führt der Zugriff auf eine ungültige Speicheradresse zum Programmabbruch; die Bearbeitung des Page-Fault-Handlers (aus B2) war also vergebens.