

1)

- 1 addi \$s1, \$zero, 1024 //  $s1 := 1024$
- 2 loop: addi \$s1, \$s1, -1 //  $s1--$
- 3 jal subroutine // call subroutine()
- 4 bne \$s1, \$zero, loop // if ( $s1 \neq 0$ ) jump loop

- Vorhersage funktioniert schlecht wenn sie "bottom of the loop" ist.

Sie sollte ganz oben sein.

-  $s1 = 1024$ , mit jedem loop -1

~~1024~~ 1024 mal loop durchlaufen bis  $s1 = 0$

## 2. Antidependencies $\rightarrow$ Antihängigkeiten

Antidependenz ist wenn bei einem Befehl ein späterer Befehl, der vor einem früheren fertig gestellt ist, einen Datenwert produziert und einen anderen überschreibt / zerstört welcher noch von einem früheren als Source verwendet wird.

Bsp aus Vorlesung:

$$1. R3: R3 * R5$$

$$2. R3: R5 + 1$$

Wenn jetzt die Zeile 2. vor 1. fertig ist zerstört sie da dann Wert R3.

## 3. Branch Delay Slots

Ist einfach gesagt ein Warteplatz. Bedingte Sprünge werden verzögert. Der Befehl nach dem Sprung wird direkt ausgeführt, der Sprung selber aber noch nicht.

Bsp. add \$1, \$2, \$3      if \$2 = 0 then  
                        if \$2 = 0 then     $\rightarrow$  [add \$1, \$2, \$3]  
                        [delay slot]

4)

```

1 int t=0;
2 for(int i=0; i<8; i++) // outer loop
3 {
4     for(int j=0; j<8; j++) // inner loop
5     {
6         t=t+i*j;
7     }
8 }
```

addi \$s1,\$zero,0 // i=0

addi \$s2,\$zero,0 // j=0

addi \$s1,\$s1,1 // i++ } Outerloop  
bne \$s1,8,out // (i<8) loop } Outerloop

addi \$s2,\$s2,1 // j++ } Innerloop  
bne \$s2,8,in // (j<8) loop } Innerloop

Iteration	Pred. bit	Ac. outcome
1	*T	T
2	T	T
3	T	T
4	T	T
5	T	T
6	T	T
7	T	T
8	T	N ← wrong prediction

inner Loop 1.

Iteration	Pred. bit	Ac. outcome
1	N	T ← wrong prediction
2	T	T
3	T	T
4	T	T
5	T	T
6	T	T
7	T	T
8	T	N ← wrong prediction

inner loop 2.-8-

$$1 + 7 \times 2 + 1 = 16. \text{ Wrong}$$

Outerloop

2-bit predictor

eigentlich gleich wie wechsel aber den predictor nach dem Loop nicht weil er ja 2. mal in Folge falsch sein müsste. (8+1) wrong

BHT  $i=6, j=3$

1-bit Predictor  $i=6 \text{ BHT} \rightarrow 1, j=3 \text{ BHT}=1$

2-bit Predictor  $i=6 \text{ BHT} \rightarrow 11, j=3 \text{ BHT}=11$

5.) Pipelining benutzt Instruction-Level-Parallelism, doch es ermöglicht dem Prozessor nicht, mehr als eine Instruktion pro Zeitpunkt auszuführen.

Falsch. Je nach Befehl braucht man mehr als einen Takt. Instruction-Level-Parallelismus ist wenn Befehle aufgeteilt werden und mehrere Teilschritte gleichzeitig ausgeführt werden

- Multiple-issue Prozessoren können mehr als eine Instruktion pro Clock-Cyklus ausführen.

Richtig. Gibt 2 verschiedene Möglichkeiten statisch oder dynamisch (VOG F. 6)

Man benutzt dafür eine erweiterte Pipeline

- SIMD-Prozessoren sind statische Multiple-Issue Prozessoren.

Richtig. Bei SIMD werden Befehle gleichzeitig ausgeführt, das funktioniert bei statischen Multiple-Issue Prozessoren.