Superskalare Architekturen (3)

Hinweis: Heute (02.12.) keine Übungen wegen studentischer Vollversammlung!

Register Renaming (1)

- Jeder Eintrag im Reorder Buffer benötigt Register zum Speichern von Operanden und spekulativen (= noch nicht bestätigten) Ergebnissen.
- Diese heißen **Schattenregister** oder **Rename Registers** (im Unterschied zu den **Befehls-satz-Registern**; **Architectural Registers**)
- *Was* wird denn hier gespeichert?

Register Renaming (2)

- Was im Reorder Buffer steht...

Befehlssatz-Register 0..n

t_0			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FMUL yk,yk,xk	MUL	▶	$4	$3	—	—	$4 (ungültig)
FADD yk,yk,q	FPU	❚❚	—	$2	FMUL	—	$4 (ungültig)
—							
—							

Register-Nummer (0..n) dito dito (nur Hinweis, wo Ergebnis später hin gehört; sowie Status)

Verweis auf Ergebnis im ROB

Reg.-Nr. (0..n) für Ziel, Status **und(!)** Ergebniswert

t_4			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FMUL yk,yk,xk	—	✔	$4	$3	—	—	$4
FADD yk,yk,q	FPU	▶	$4*	$2	—	—	$4 (ung.)
SET xk,temp1	INT1	✔	$10	×	—	—	$3
FMUL temp1,xk,xk	MUL	▶	$3*	$3*	—	—	—

Register Renaming (3)

- Pro „Zeile“ im ROB muss die Ziel-Spalte also Platz bieten für
  - eine Register-Nummer (Zielregister)
  - Status
  - einen berechneten Wert (in „Registergröße“, diese Teile des ROB heißen „Rename-Register“)
- Es ist ökonomisch sinnvoll, lediglich Pointer auf Rename Register in einem Register-Pool (für Zwischenergebnisse) zu speichern.

Register Renaming (4)

Register-Pool:

Wert	frei	gültig
	+	–
	–	+
	–	+
	–	–
	+	–
	–	+
	–	–
	–	+
	+	–
	–	+

Pointer aus ROB-Ziel-Spalte

frei: Eintrag nicht benutzt

gültig: enthält fertig berechneten Wert, kann für weitere Berechnungen genutzt werden

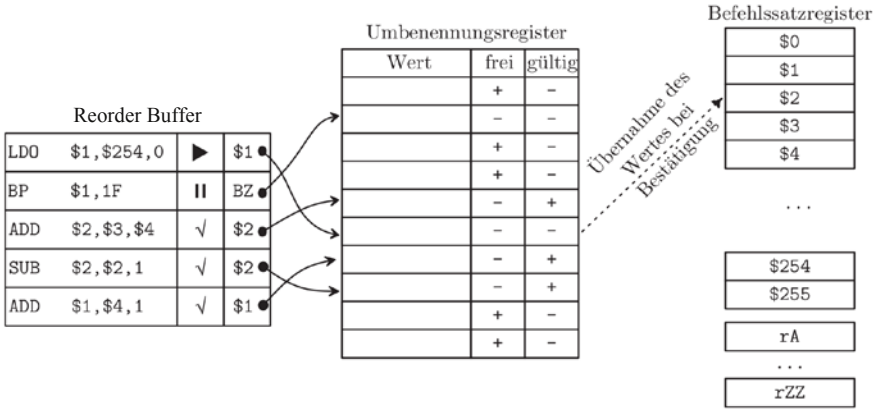
Register Renaming (5)

Implizite Umbenennung

- Zwei getrennte Sätze physischer Register für Befehlssatz-Register und Rename Register
- Es ist nirgends explizit vermerkt, wo sich der jüngste spekulative Wert eines Registers befindet. Finden des Wertes: ROB durchsuchen
  - Neuer Befehl `ADD $1,$3,$3` kommt in den ROB. „Welches“ $3 nehmen? Reg.-Inhalt? ROB-Inhalt? Warten auf Berechnung von $3?
- Das Verwerfen spekulativer Werte (bei Interrupts oder falsch vorhergesagten Sprüngen) ist einfach.

Register Renaming (6)

Implizite Umbenennung

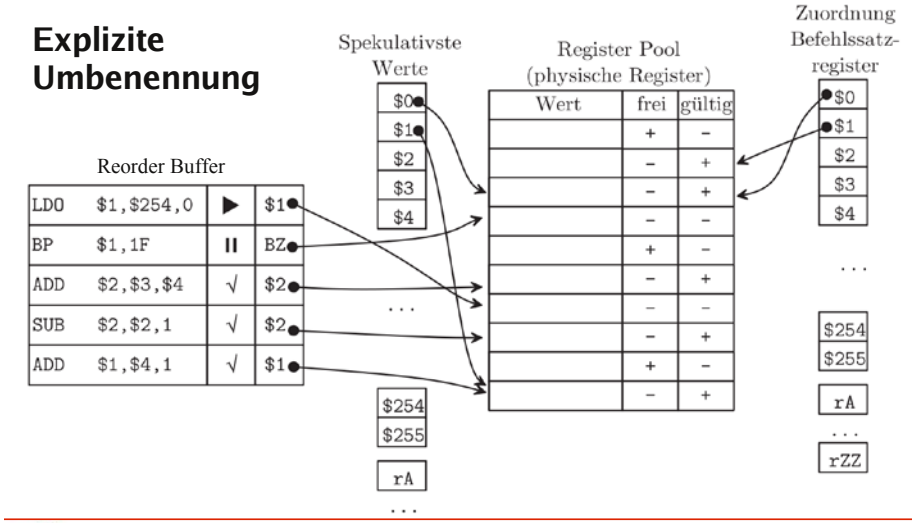


Register Renaming (7)

Explizite Umbenennung

- Es gibt nur einen Register-Pool (gemeinsam für Befehlssatz- und Rename-Register)
- Eine Tabelle gibt an, wo die jüngsten (spekulativsten) Werte der Register stehen
- Kein Umkopieren der Register nach Bestätigen eines Befehls;
  Pointer auf die physischen Register beschreiben Registerzustand

Register Renaming (8)



Register Renaming (9)

Datenabhängigkeiten

- **Read-after-Write** (**RAW**):
  führt stets zum Stillstand, bis das Ergebnis zur Verfügung steht.
- **Write-after-Read** (**WAR**) und
  Write-after-Write (**WAW**):
  Schattenregister können Stillstand verhindern (wenn es genügend freie gibt)

Register Renaming (10)

- „Register Renaming“ auch direkt aus dem Wort verständlich:

```
1. R1=M[1024]
2. R1=R1+2
3. M[1032]=R1
4. R1=M[2048]
5. R1=R1+4
6. M[2056]=R1
```

Nach Umbenennen von R1 in R2:

```
1. R1=M[1024]     4. R2=M[2048]
2. R1=R1+2        5. R2=R2+4
3. M[1032]=R1     6. M[2056]=R2
```

WAR-Konflikt zwischen (3) und (4);
Befehl (4) darf erst schreiben, wenn (3) fertig ist

„echte“ Pipelines

- echte Beispiel-Architekturen:
  - Pentium Pro: bis zu 16 Stufen
  - PowerPC 970: 16–24 Stufen

→ Übung

Sprungvorhersage

Sprungvorhersage

- feste „Vorzugsrichtung“ (keine Vorhersage)
  - Sprung nach vorne: unwahrscheinlich
- statische Sprungvorhersage
  - probable branch / branch (MMIX, PowerPC 601)
  - Setzen des Take/Don't Take Bit (TDTB) bzw. Static Prediction Opcode Bit durch optimierende Compiler
- dynamische Sprungvorhersage
  - Vorhersage hängt von Vergangenheit ab

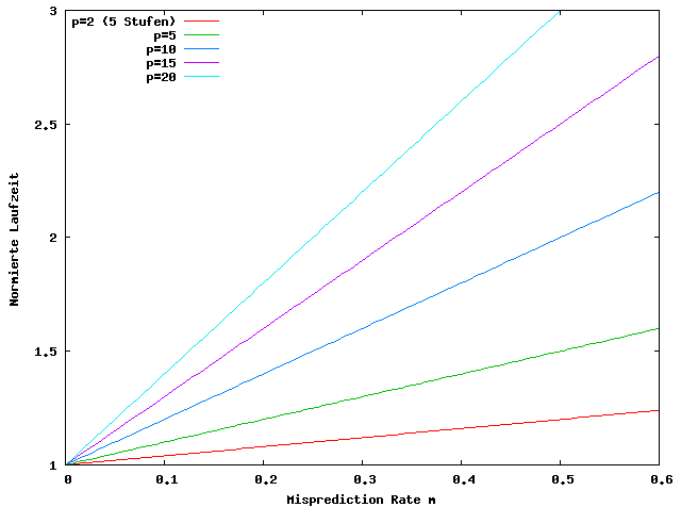
Leistungsverlust (1)

- Was bewirken falsch vorhergesagte Sprünge?
- Kenngrößen:
  - b branch rate, z. B. 20%
  - m misprediction rate, = 1 – prediction rate
  - p penalty (Strafe) für falsche Vorhersagen in Taktzyklen
- Laufzeit T eines Programms mit n Befehlen:

$$T = n + n \cdot b \cdot m \cdot p = n \cdot (1 + b \cdot m \cdot p)$$

(Auf- und Abbau der Pipeline ignorieren)

Leistungsverlust (2)

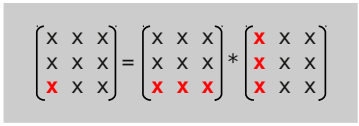


Sprungvermeidung (1)

- Je langer die Pipeline ist, desto „teurer“ sind falsch vorher gesagte Sprünge (Branch Penalty). Daher: viel Aufwand, um Sprungziele vorherzusagen
- Sprungvermeidung (Software-Technik)
  - Bedingte Befehle statt bed. Sprüngen verwenden
  - Loop Unrolling (machen Compiler oft automatisch)

Sprungvermeidung (2)

- Loop Unrolling, Beispiel: Matrixmultiplikation



```
int i,j,k;
for (i=0; i<3; i++)
  for (j=0; j<3; j++)
    for (k=0; k<3; k++)
      m3[i][j] += m1[i][k] * m2[k][j];
```

hängt gar nicht von k ab!

```
int i,j;
for (i=0; i<3; i++)
  for (j=0; j<3; j++)
    m3[i][j] += m1[i][0] * m2[0][j]
              + m1[i][1] * m2[1][j]
              + m1[i][2] * m2[2][j];
```


Statische Sprungvorhersage (1)

- Vorhersage dem Programmierer/Compiler überlassen:
  - Ein Bit im Opcode gibt an, ob die Verzweigung wahrscheinlich ausgeführt werden wird (TDTB: Take / Don’t Take Bit)
  - Bei MMIX: probable branches / normale branches
  - MMIX-Opcodes unterscheiden sich bei Bit 4;
    BZ (Branch if Zero) #42 = 10**0**0010
    PBZ (Probable Branch if Zero) #52 = 10**1**0010
- Besonders geeignet für Zählschleifen

Statische Sprungvorhersage (2)

- Beispiel: Insertion Sort

```
01  * Parameter
02  A       IS      $0
03  size    IS      $1
04  * Lokale Register
05  i       IS      $2      Offset für einzufügende Elemente
06  j       IS      $3      Laufindex für das Einfügen
07  k       IS      $4
08  x       IS      $5
09  y       IS      $6
10  tmp     IS      $7
11
12  * Zuerst Positionierung auf 2. Element. Alle Elem. 8 Bytes groß
13  * Daher werden die Indizes stets um Vielfache von 8 bewegt
14
```

Statische Sprungvorhersage (3)

```
15  :ISort  SET   i,8
16          JMP   1F
17
18  2H      LDO   x,A,i          nächstes einzufüg. Element
19          SET   j,i
20          JMP   3F
21  5H      STO   y,A,j          y rückt eine Pos. weiter vor
22          SET   j,k
23  3H      BNP   j,4F
24          SUBU  k,j,8
25          LDO   y,A,k
26          CMP   tmp,y,x
27          PBP   tmp,5B
28
29  4H      STO   x,A,j          Elem. x hat Platz gefunden
30          ADDU  i,i,8
31  1H      CMP   tmp,i,size
32          PBN   tmp,2B
33          POP   0,0            Kein Return-Wert
```


Statische Sprungvorhersage (4)

- Messergebnisse für Quicksort-Programm
  - Sortieren von 10.000 Zufallswerten
  - Unterprogramm *insertionsort.mms* wird 1551 mal aufgerufen, sortiert im Schnitt 5,44 Werte
  - Trefferraten:

Zeile/Bef.	Taken	Not Taken	Trefferrate
23: BNP	2.059	16.345	88,8 %
27: PBP	11.505	4.840	70,4 %
32: PBN	6.899	1.551	81,6 %

Dynamische Sprungvorh. (1)

- Vorhersage hängt von Vergangenheit ab
- CPU speichert zu jedem Verzweigungsbefehl einen Prädiktor
  - Prädiktor kann unterschiedliche Länge haben
  - jedes Bit speichert Ergebnis („taken“ oder „not taken“) einer vergangenen Verzweigung
  - z. B.: 2-bittig, TT = zuletzt 2x taken
    TN = zuletzt taken, davor not taken

Dynamische Sprungvorh. (2)

- Beispielprogramm:

```
01  YesNo   GREG   #AAAAAAAAAAAAAAAA
02
03  P       IS     $2
04  q       IS     $3
05
06          LOC    #100
07  Main    PBEV   YesNo,1F      Sprung falls gerade (even)
08          ADD    P,P,1         Misserfolge zählen
09  1H      AND    q,YesNo,1     rotieren: q = d für YN=ab...cd
10          SLU    q,q,63        q = d000
11          SRU    YesNo,YesNo,1 R-Shift YN: ab..cd -> 0a..bc
12          OR     YesNo,YesNo,q d vorne „einbauen“: -> da..bc
13          JMP    Main
```

0 (hex) = 0000 (bin)
A (hex) = 1010 (bin)
F (hex) = 1111 (bin)

  - YesNo bestimmt Taken/Not Taken-Sequenz
  - einfache Anpassung durch andere Wahl von YesNo

1-Bit-Prädiktor (1)

- Vorhersage: nächste Ausführung so wie die letzte
  - **0**: Verzweigung wird übereinstimmend mit der Vorhersage eingeschlagen („In Agreement“, kurz A)
  - **1**: Verzweigung wird sich entgegen der Vorhersage („In Opposition“, kurz O) verhalten

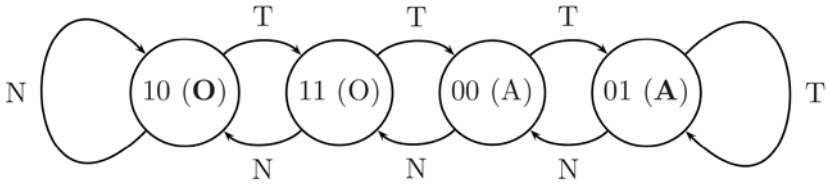
1-Bit-Prädiktor (2)

- Ausführung mit YesNo = #8888888888888888 = 10001000... (bin):

Durchlauf	Prädiktor	Verzweigung	Fehler
1	0 (T)	T	
2	0 (T)	T	
3	0 (T)	T	
4	0 (T)	N	*
5	1 (N)	T	*
6	0 (T)	T	
7	0 (T)	T	
8	0 (T)	N	*
9	1 (N)	T	*

2-Bit-Prädiktor (1)

- Saturierender Zähler in Zwei-Komplement-Darstellung:



Bitmuster	Wert	Vorhersage
00	0	in Agreement (A)
01	1	in Strong Agreement (**A**)
11	-1	in Opposition (O)
10	-2	in Strong Opposition (**O**)

2-Bit-Prädiktor (2)

- Ausführung mit YesNo = #8888888888888888 = 10001000... (bin):

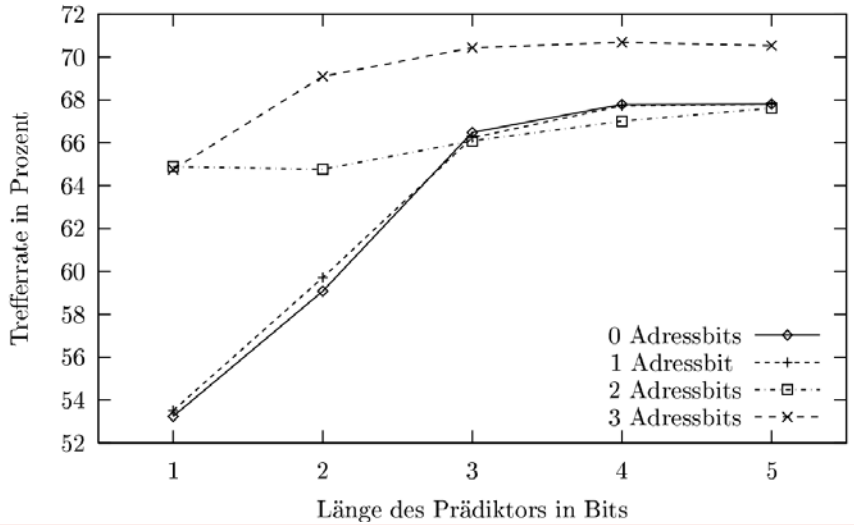
Durchlauf	Prädiktor	Verzweigung	Fehler
1	00 (T)	T	
2	01 (**T**)	T	
3	01 (**T**)	T	
4	01 (**T**)	N	*
5	00 (T)	T	
6	01 (**T**)	T	
7	01 (**T**)	T	
8	01 (**T**)	N	*
9	00 (T)	T	

Speichern des Prädiktors

- nicht für jede Speicheradresse einen Prädiktor speichern; auch nicht für jeden Sprung
- Adress-Hashing: Index mit 2 Adress-Bits

```
0000 00 00  →  00    0
0000 01 00  →  01    1
0000 10 00  →  10    2
0000 11 00  →  11    3
0001 00 00  →  00    0
0001 01 00  →  01    1
...
```


Messergebnisse für Quicksort



Vorschau 09.12.2010

- mehr Sprungvorhersage: History
- Einführung Speicher

Hinweis: Die heutigen Übungen entfallen wegen der studentischen Vollversammlung.