EMES: Eigenschaften mobiler und eingebetteter Systeme

Ausführungszeitvorhersage

Dr. Felix Salfner, Dr. Siegmar Sommer

Wintersemester 2010/2011

Gliederung

1. Einleitung
2. Theoretische Vorbetrachtungen
3. Woher kommt der Pessimismus?
4. Technische Grundlagen der Referenzarchitektur
5. Ein Beispielalgorithmus
6. Messungen an Prozessoren

Motivation

- Schedulingverfahren basieren auf Kenntnis der Ausführungszeit einer Task
- Zeitliche Garantien können nur gegeben werden, wenn man Ausführungszeiten kennt

Problem: Wie bestimmt man eine Ausführungszeit?
Zwei Ansätze:

- Messen (Stoppuhr? Logic-Analyzer?)
- Berechnung auf Grundlage von Handbüchern, etc.

Worst Case Execution Time I

- Kenntnis **einer** Ausführungszeit genügt nicht, denn Laufzeit eines Programmes kann von den Eingaben abhängen
- Interessant ist die längstmögliche Ausführungszeit (WCET)
- WCET hängt ab von
  - Programmcode
  - Compiler, benutzte Bibliotheken
  - Pfaden durch den Programmcode (abhängig von Eingaben)
  - der benutzten CPU
  - Parametern der Umgebung der CPU (Speicher, Caches)
  - sowie Wechselwirkungen zwischen diesen Parametern

Längste Pfade I

Beispiel: Wie hoch ist die WCET dieses Programmstückes?

```
int i,k;
int j = readsensor();
while(i<j)
  {
     k+=sensorfunction(i);
     i++;
  }
```

Längste Pfade I

Beispiel: Wie hoch ist die WCET dieses Programmstückes?

```
int i,k;
int j = readsensor();
while(i<j)
  {
     k+=sensorfunction(i);
     i++;
  }
```

Problem: Wie oft wird diese Schleife durchlaufen?

Lösung: Angabe von Obergrenzen

Längste Pfade II

Beispiel: Wie hoch ist die WCET dieses Programmstückes?

```
int i,k;
int j = readsensor();
while(i<j)
  {
     k+=sensorfunction(i);
     i=somefunction();
  }
```

Längste Pfade II

Beispiel: Wie hoch ist die WCET dieses Programmstückes?

```
int i,k;
int j = readsensor();
while(i<j)
  {
     k+=sensorfunction(i);
     i=somefunction();
  }
```

Problem: Terminiert diese Schleife überhaupt?

Generell: Halteproblem ist nicht entscheidbar!

Längste Pfade III

- Im allgemeinen ist die Suche eines längsten Pfades nicht entscheidbar
- Unter speziellen Annahmen sind Berechnungen dennoch möglich:
  - Kenntnis der maximal möglichen Anzahl von Durchläufen einer Schleife
  - Keine rekursiven Aufrufe
  - Keine dynamischen Datenstrukturen
- Folge: Strenge Einschränkung von Programmkonstrukten, die in Echtzeitprogrammen überhaupt “erlaubt” sind.
- Ansatz: Suche eines längsten Pfades im Kontrollflußgraphen

Bestimmung der WCET

Es gibt zwei generelle Probleme:

- Welches ist der längste Pfad durch ein Programm?
  Unterproblem: Wie bewertet man einen Pfad?

  $\longrightarrow$ **Highlevel Analyse**

- Wie berechnet man zu einem gegebenen Programmpfad die Ausführungszeit?

  $\longrightarrow$ **Lowlevel Analyse**

Bestimmung der WCET II

- Analysewerkzeug erstellt einen *Kontrollflussgraphen*
  $\longrightarrow$ Kontrollflussgraph besteht aus *Basic Blocks*
- Ausführungszeit aller Basic Blocks wird ermittelt (Lowlevel-Analyse!)
- Berechnung des längsten Pfades
- Gegebenenfalls weitere Optimierungen (ist längster Pfad *feasable*?)

Basic Blocks sind Blöcke von Maschinenbefehlen, die genau einen Ein- und einen Austrittspunkt haben.

Highlevel-Analyse mit dem Timing-Schema

$$
\begin{aligned}
W(X) &= \textit{Zahl, wenn X ein BB ist} \\
W(X;Y) &= W(X) + W(Y) \\
W(\textit{if } Z \textit{ then } X \textit{ else } Y) &= W(Z) + \max\{W(X), W(Y)\} \\
W(\textit{for } Z \textit{ loop } X) &= (n+1) \cdot W(Z) + n \cdot W(X)
\end{aligned}
$$

- Schleifendurchläufen sind durch n beschränkt
- Ausführungszeit eines BBs aus der Lowlevel-Analyse
- wird rekursiv für jedes Blatt des Syntaxbaumes ausgeführt
- einfachstes Timing-Schema

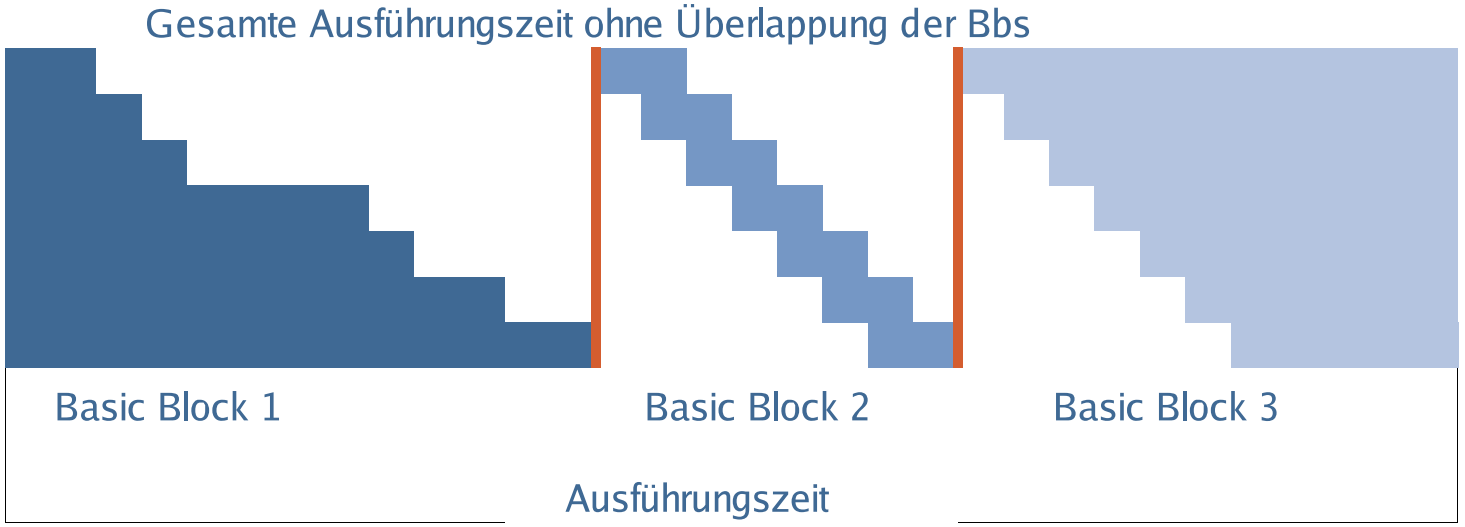
Lowlevel-Analyse – Voraussetzungen

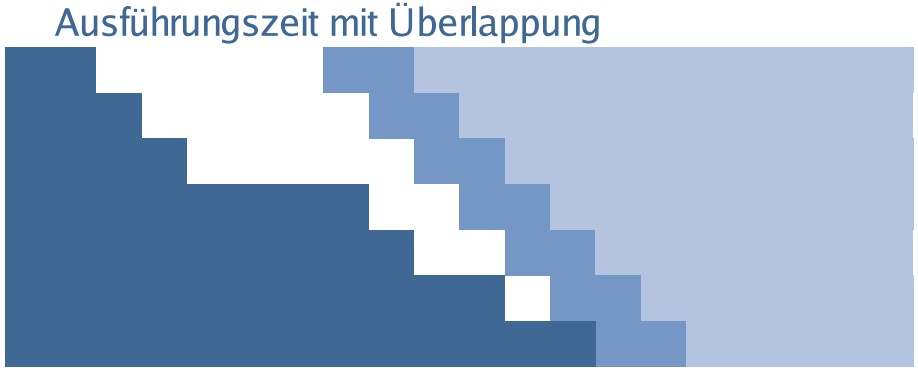
- Genaue Reihenfolge der Instruktionen muss bekannt sein, d.h.
  - keine (unvorhergesehenen) Interrupts
  - keine Exceptions
- Prozessor muss spezifikationsgetreu betrieben werden
- Prozessorzustand muss konstant bleiben
- DMA-Zugriffe sind verboten
- Detaillierte Beschreibung des gewünschten Prozessors/Systems muss vorliegen

Woher kommt der Pessimismus I

- **Vereinfachende Annahmen** zu
  - Synchronisationszeiten
  - Anzahl von Schleifendurchläufen
  - Zugriffszeiten
  - Cache/TLB/BHT (Branch History Table)
  - Prozessoreigenheiten
  - ...
- Unkenntnis des Systemzustandes
- Schleifen
- Haben wir auch wirklich den Worst-Case-Pfad gefunden?
- Ungenaue Lowlevel-Analyse (Überlappung)

Beispiel: Überlappung





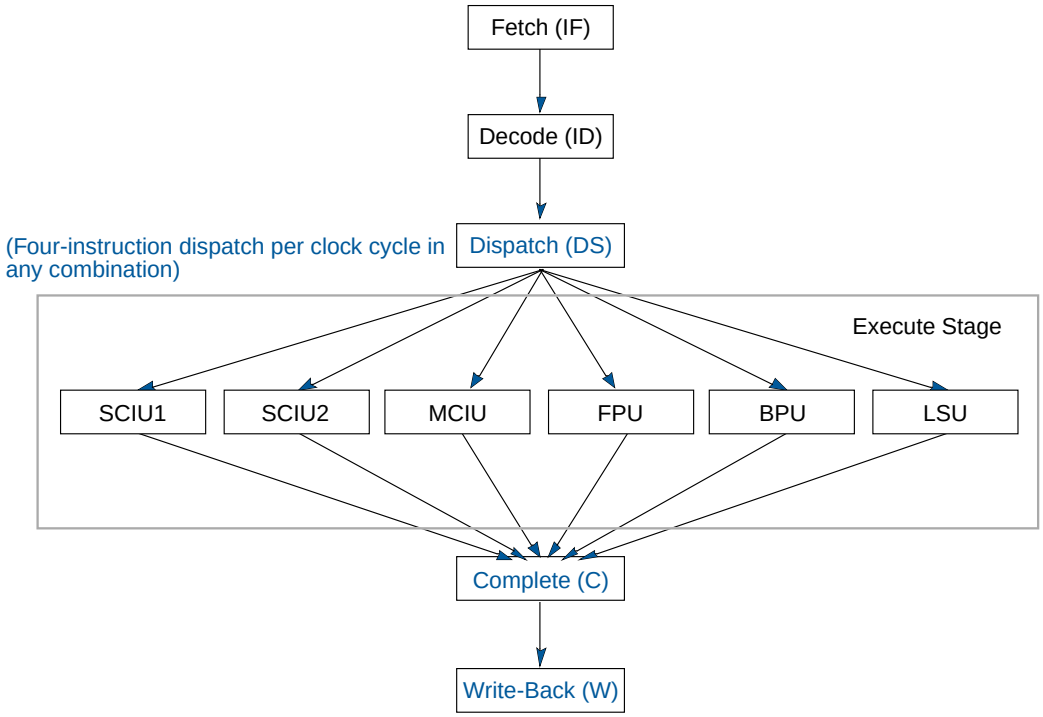
Wichtige Systemparameter

- Pipeline?
  - Funktionsweise der einzelnen Stufen
  - Länge
  - Organisation
- Funktionseinheiten
- Speicher
  - Aufbau
  - Struktur
  - Cachestruktur
  - Geschwindigkeit
  - Busorganisation
  - Burstmodi
  - Taktteiler Bus/CPU

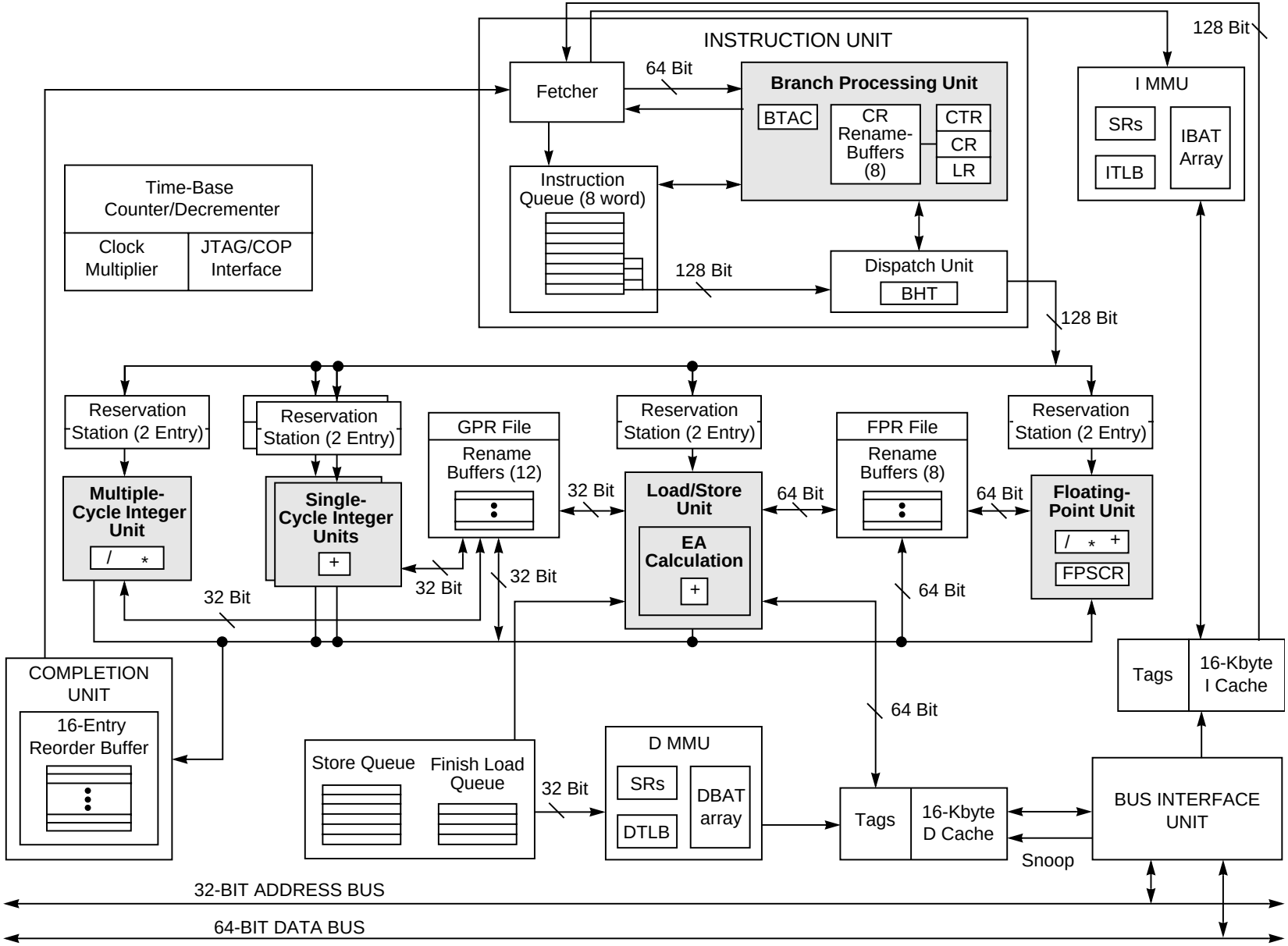
Referenzarchitektur PowerPC 604

- Vorgestellt Dezember 1994, hergestellt von IBM und von Motorola
- Mitglied einer ganzen Familie von Prozessoren
- Komplexität ist noch handhabbar
- Existiert als Hostprozessor und als eingebettetes Gerät
- RISC - Architektur
- „Performance Enhancements“ (superskalare Architektur)
  - Pipeline
  - Große Caches
  - Sprungvorhersage
  - Spekulative Ausführung
  - Out-of-Order-Execution
  - ...

PowerPC 604 – Pipeline



- SCIU: Single Cycle Integer Unit
- MCIU: Multi Cycle Integer Unit
- FPU: Floating Point Unit
- BPU: Branch Processing Unit
- LSU: Load Store Unit


INSTRUCTION UNIT
Fetcher
64 Bit
Branch Processing Unit
BTAC
CR Rename-Buffers (8)
CTR
CR
LR
128 Bit
I MMU
SRs
IBAT Array
ITLB
Instruction Queue (8 word)
128 Bit
Dispatch Unit
BHT
128 Bit
Time-Base Counter/Decrementer
Clock Multiplier
JTAG/COP Interface
Reservation Station (2 Entry)
Reservation Station (2 Entry)
GPR File
Rename Buffers (12)
Reservation Station (2 Entry)
FPR File
Rename Buffers (8)
Reservation Station (2 Entry)
Multiple-Cycle Integer Unit
/ *
Single-Cycle Integer Units
+
32 Bit
Load/Store Unit
EA Calculation
+
64 Bit
64 Bit
Floating-Point Unit
/ * +
FPSCR
32 Bit
32 Bit
32 Bit
64 Bit
COMPLETION UNIT
16-Entry Reorder Buffer
Tags
16-Kbyte I Cache
64 Bit
Store Queue
Finish Load Queue
32 Bit
D MMU
SRs
DBAT array
DTLB
Tags
16-Kbyte D Cache
Snoop
BUS INTERFACE UNIT
32-BIT ADDRESS BUS
64-BIT DATA BUS

Pipelining

IF	1	2	3	4							
DEC		1	2	3	4						
DIS			1	2	3	4					
EX				1	2	3	4	4	4		
CPL					1	2	3			4	
WB						1	2	3			4

IF (Instruction fetch): Hole vier Befehle gleichzeitig
DEC (Decode): Ressourcen bestimmen (renamed registers, etc.)
DIS (Dispatch): Schicken an Reservation Station der Execution unit (in-order)
EX (Execute): Paralleles Ausführen wenn Operanden verfügbar sind (out-of-order)
CPL (Completion): Wenn keine Exception & alle vorherigen Ops fertig (in-order)
WB (Write-back): Schreiben der Ergebnisse an die Ziel-Operanden

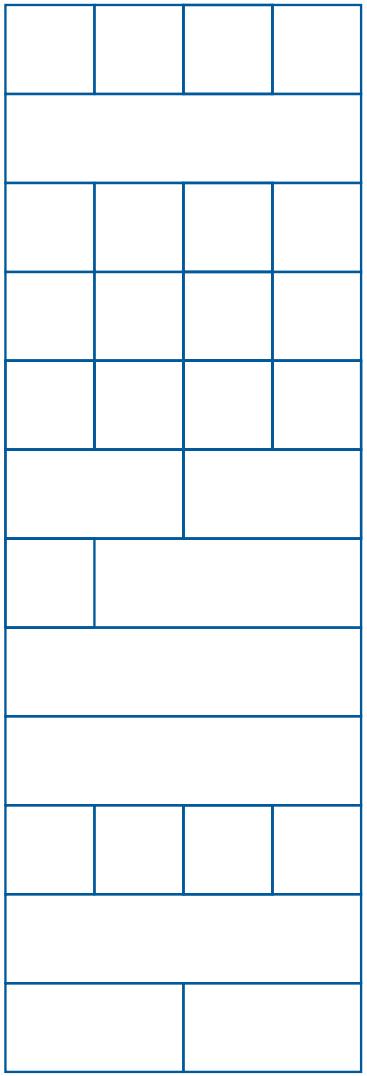
Branch Processing

- Dynamische „Branch prediction“
- Spekulative Ausführung der wahrscheinlicheren Branch
- Fetch und Execute über zwei Branch-Anweisungen hinweg
- Wenn sich die Vorhersage als korrekt herausgestellt hat: Übernehmen der Ergebnisse in der Write-back Phase.
- Wenn die Vorhersage falsch war: Entfernen der spekulativen Ausführungen aus den Reservation stages, Execution units und Queues.
- Wiederherstellung des alten Zustands.

Reservation Table

- Erlaubt Behandlung von Pipelines
- Aufbau repräsentiert Merkmale der Architektur
- Zusammensetzen mehrerer Tabellen ist durch Architektur eingeschränkt
- Einfaches Ablesen der Laufzeit (Spaltenanzahl)
- Intuitives Konzept und leicht graphisch darstellbar

Angepasste Reservation Table - Eine Spalte



Instruction Fetch I

Branch Prediction Unit (BPU)

Instruction Fetch II

Decode

Dispatch

Single Cycle Integer Unit (SCIU)

Multiple Cycle Integer Unit (MCIU)

Load Store Unit (LSU)

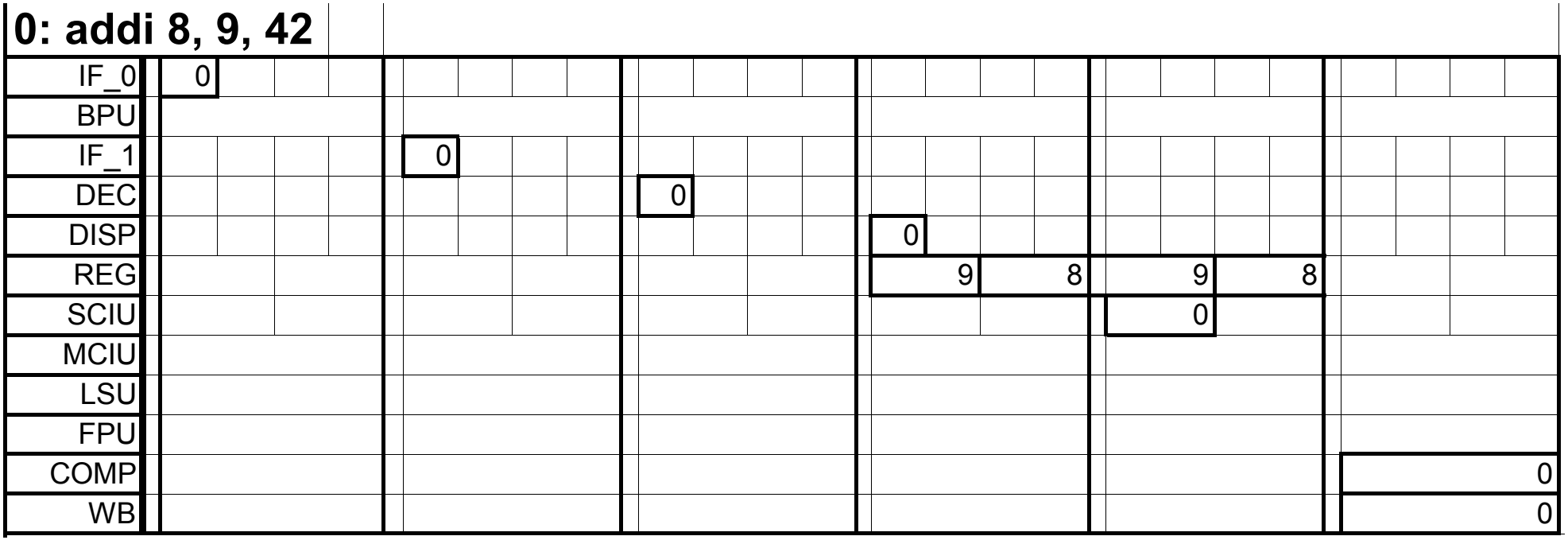
Floating Point Unit (FPU)

Complete

Write Back

Registers {R}|{W}

Repräsentation eines Addierbefehls

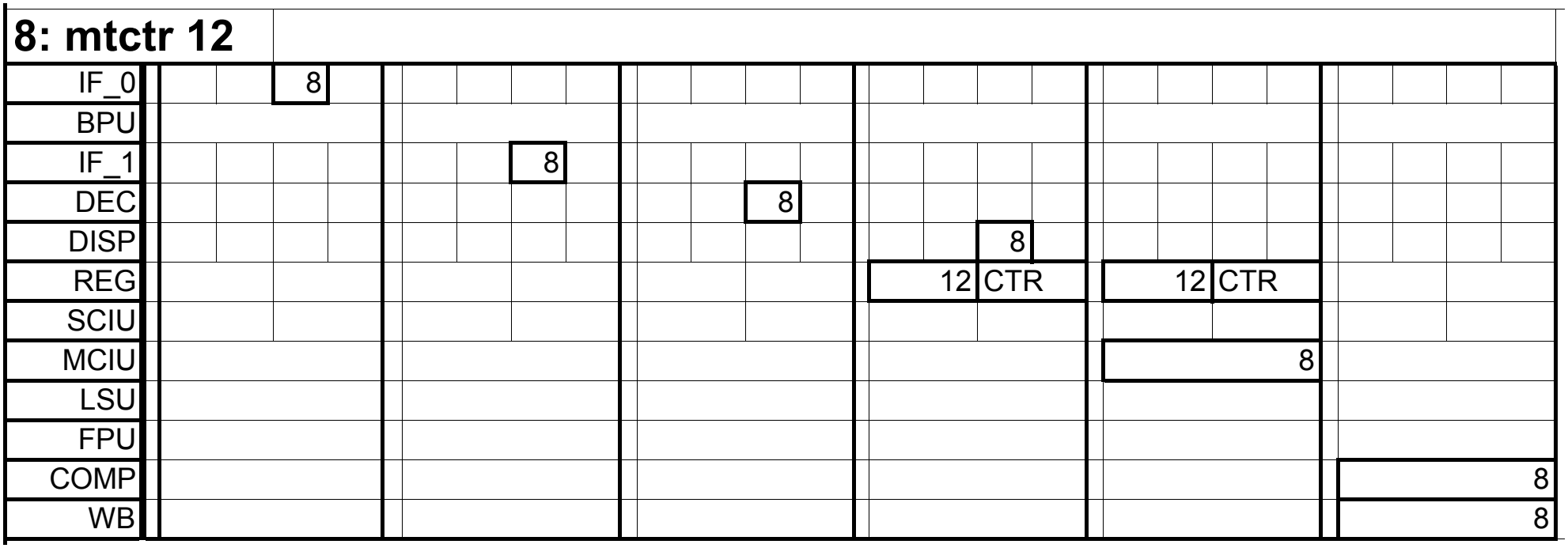


- Jeder Befehl wird durch seine Adresse dargestellt

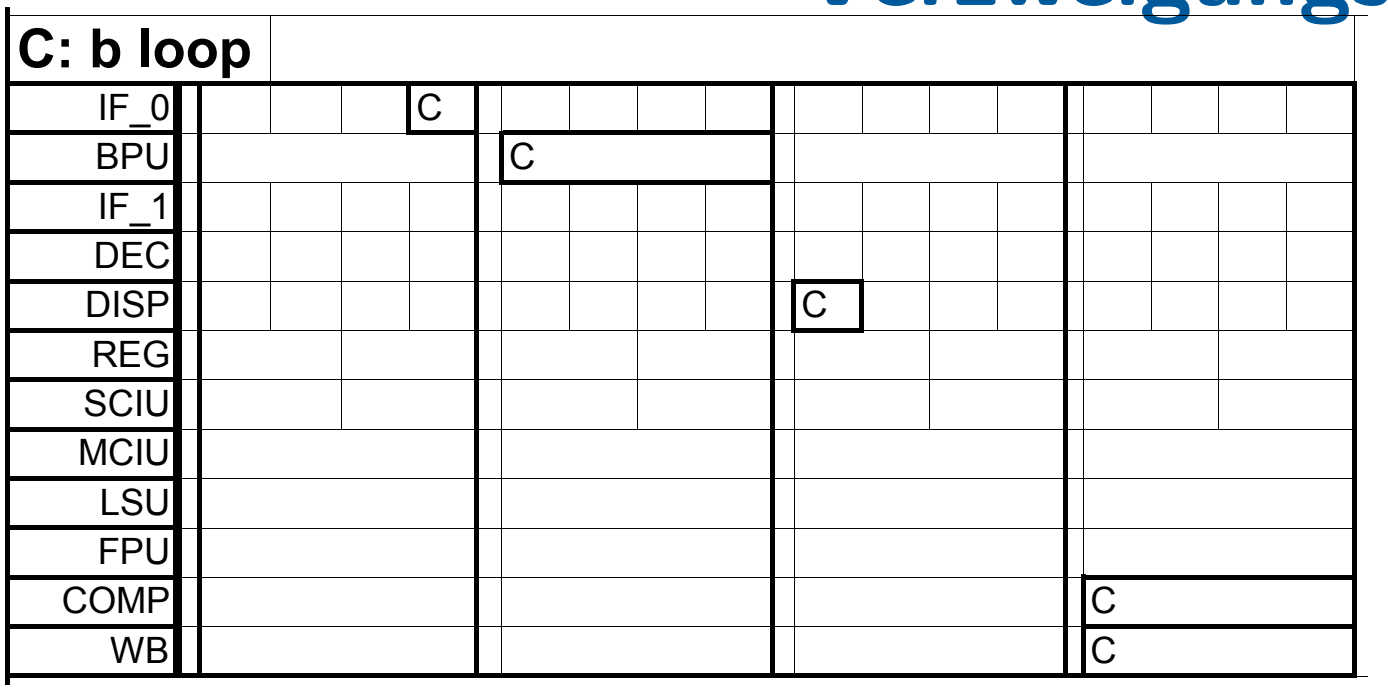
Repräsentation eines OR - Befehls

4: or 27, 11, 13																								
IF_0			4																					
BPU																								
IF_1							4																	
DEC										4														
DISP													4											
REG													11,13		27		11,13		27					
SCIU																			4					
MCIU																								
LSU																								
FPU																								
COMP																							4	
WB																							4	

Repräsentation eines MOVE - Befehls



Repräsentation eines Verzweigungsbefehls

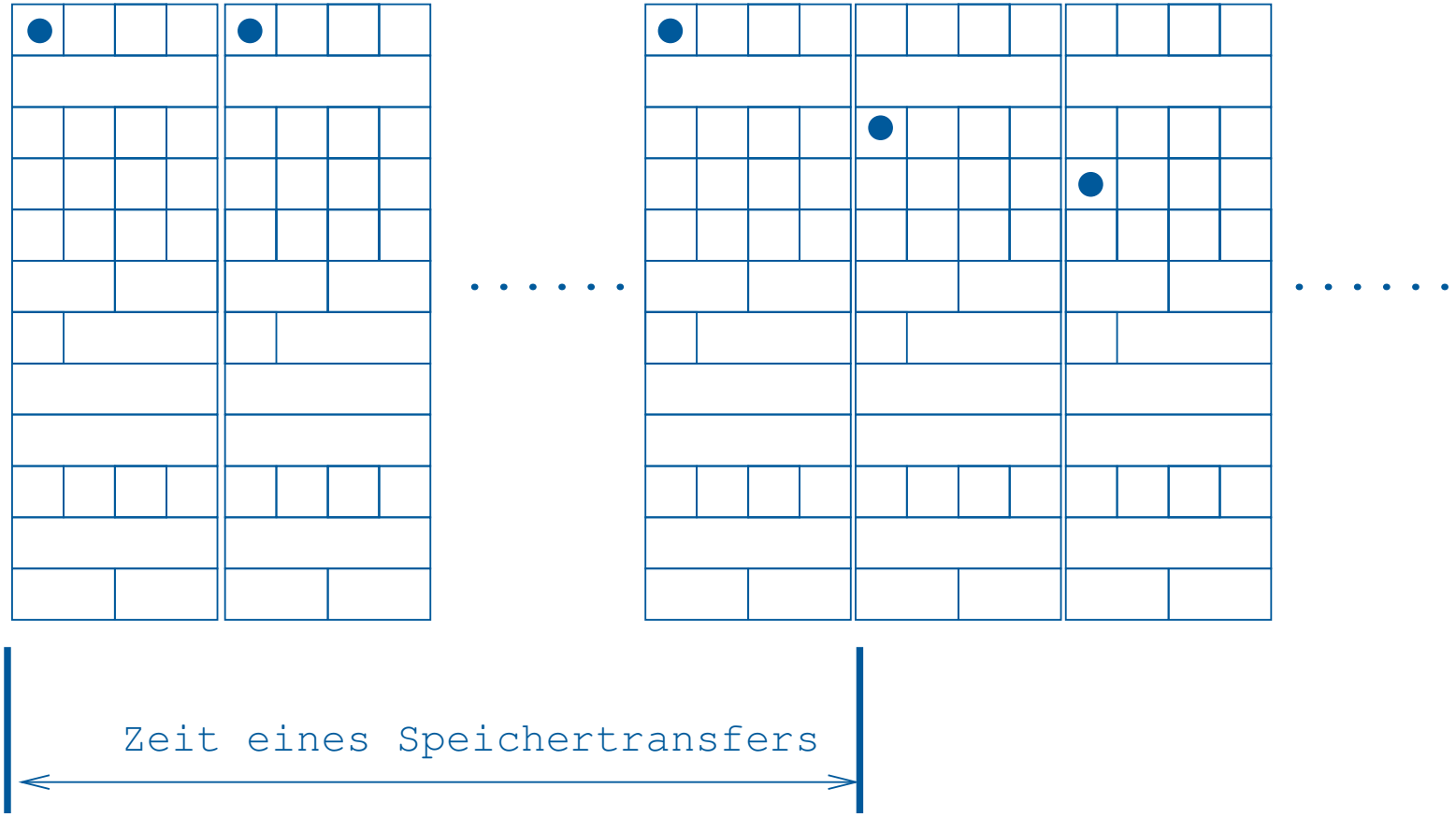


Verbindung der 4 Tabellen

IF_0	0	4	8	C																								
BPU					C																							
IF_1					0	4	8																					
DEC									0	4	8																	
DISP									C				0	4	8													
REG													9, 11, 12, 13	CTR, 27, 8			9, 11, 12, 13	CTR, 27, 8										
SCIU																		0	4									
MCIU																			8									
LSU																												
FPU																												
COMP													C				C				0, 4, 8, C							
WB													C				C				0, 4, 8, C							

- Alle vier Instruktionen laufen parallel ab
- Codealignment ist essentiell

Darstellung eines Cachesmiss



Algorithmus

1. Erstelle zu jeder Instruktion eine Tabellenrepräsentation
2. Verbinde die Tabellen unter Berücksichtigung architektureller Gegebenheiten
3. Lese die Anzahl der konsumieren Takte ab

Was noch fehlt

- Die Caches und Puffer innerhalb der CPU müssen vom Algorithmus verwaltet werden:
  - Daten-/Instruktionscache, TLB
  - Statusregister
  - Branch Target Address Cache (BTAC), Branch History Table (BHT)
  - Reorder Buffer
  - Store Queue, Finish Load Queue

Anderes Beispiel: Motorola 68000

Tabelle mit Ausführungszeiten

Move Byte and Word Instruction Execution Times

Source	Destination								
	Dn	**An**	**(An)**	**(An)+**	**–(An)**	**(d_{16}, An)**	**(d_8, An, Xn)***	**(xxx).W**	**(xxx).L**
Dn	**4**(1/0)	**4**(1/0)	**8**(1/1)	**8**(1/1)	**8**(1/1)	**12**(2/1)	**14**(2/1)	**12**(2/1)	**16**(3/1)
An	**4**(1/0)	**4**(1/0)	**8**(1/1)	**8**(1/1)	**8**(1/1)	**12**(2/1)	**14**(2/1)	**12**(2/1)	**16**(3/1)
(An)	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)
(An)+	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)
–(An)	**10**(2/0)	**10**(2/0)	**14**(2/1)	**14**(2/1)	**14**(2/1)	**18**(3/1)	**20**(3/1)	**18**(3/1)	**22**(4/1)
(d_{16}, An)	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
(d_8, An, Xn)*	**14**(3/0)	**14**(3/0)	**18**(3/1)	**18**(3/1)	**18**(3/1)	**22**(4/1)	**24**(4/1)	**22**(4/1)	**26**(5/1)
(xxx).W	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
(xxx).L	**16**(4/0)	**16**(4/0)	**20**(4/1)	**20**(4/1)	**20**(4/1)	**24**(5/1)	**26**(5/1)	**24**(5/1)	**28**(6/1)
(d_{16}, PC)	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
(d_8, PC, Xn)*	**14**(3/0)	**14**(3/0)	**18**(3/1)	**18**(3/1)	**18**(3/1)	**22**(4/1)	**24**(4/1)	**22**(4/1)	**26**(5/1)
#<data>	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)

*The size of the index register (Xn) does not affect execution time.

Messen der WCET

- Bei CPUs ohne Cache und Pipelines erfolgversprechende Methode
- Caches und/oder Pipelines:
  - Keine reproduzierbare Zeit in Abhängigkeit von Ausführungsumgebung (davor abgearbeitete Programme, Zustand des Speichers/Caches, Einfluß durch andere Programme (Preemption oder Interrupts))
  - Vollständiges Durchtesten aller möglichen Ausgangszustände ist durch hohe Komplexität unmöglich
- Methode kann benutzt werden, wenn ausreichende Annahmen über das System getroffen werden können (beispielsweise keine Preemption, stets gleicher Ausgangszustand, keine Interrupts)

Messungen am Prozessor

- PowerPC 604 hat Performance Monitor Register
- Vielfaltige Monitoringmöglichkeiten
  - konsumierte CPU-Takte
  - Cachemisses
  - Statistiken zu Instruktionen
  - fehlerhaft vorhergesagte Sprünge
  - Speicherzugriffsverzögerung

aber: PMC laufen nur im Supervisor Mode

- Tiefgreifende Modifikationen (Interrupts / Cache ausschalten) ebenfalls nur im Supervisor Mode möglich → Programmierung eines Kerneltreibers

Testprogramm

Expandierende For – Schleife

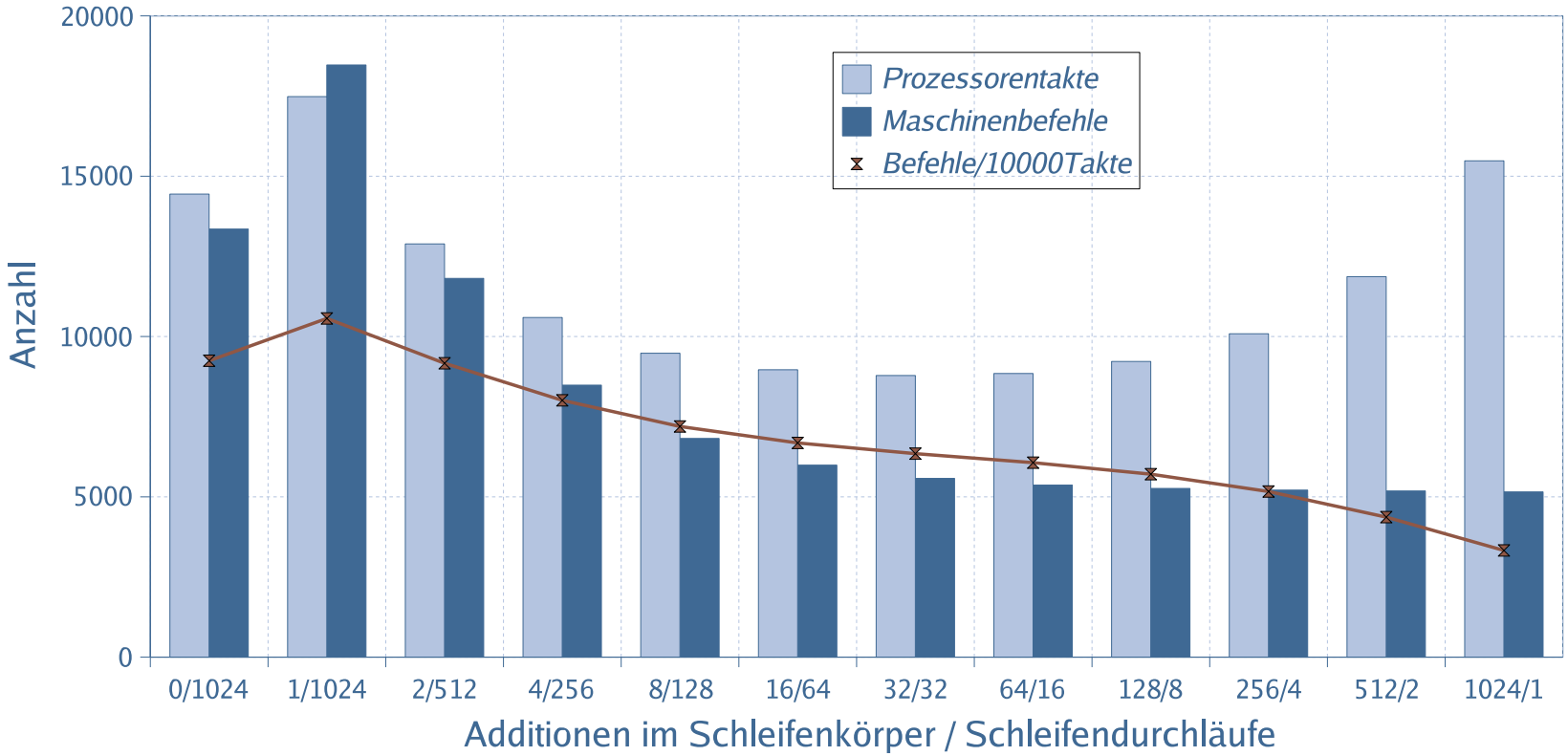
- n mal eine Zahl um eins erhöhen
- mehrmalige Ausführung
- Veränderung des Verhältnisses von Schleifenlänge und Sprunghäufigkeit
- Halbierung der Schleifenlänge bei Verdopplung der Sprunghäufigkeit
- Ausführung mit und ohne Cache
- Ausführung mit Registervariablen (wenige Speichertransfers, kurzer Code)
- Ausführung mit Variablen im Hauptspeicher (viele Speichertransfers, längerer Code)

Ausgabe des Treibers

```
Interrupts are:          Enabled
Disable Interrupts:                          [   OK   ]
ICache is:               Enabled             [   OK   ]
DCache is:               Enabled             [   OK   ]
Serialization is:        Disabled            [   OK   ]
Saving Settings                              [   OK   ]

PMC1_CYCLES=[14442 17482 12888 10593 9480 8964 8784 8847 9223 10086 11869
PMC2_LOAD_MISS_PENALTY=[14 0 0 0 0 0 14 0 0 0 0 0 0 ]
PMC1_RESERVATIONS_REQ=[0 0 0 0 0 0 0 0 0 0 0 0 0 ]
PMC2_INSN_DISPACHED=[13352 18469 11815 8485 6821 5989 5575 5367 5263 5211
PMC1_ICACHE_MISS=[5 3 4 4 8 13 23 42 82 162 322 645 ]
PMC2_DCACHE_MISS=[2 0 0 0 0 0 1 0 0 0 0 0 0 ]
PMC1_DTLB_MISS=[0 0 0 0 0 0 0 0 0 0 0 0 0 ]
PMC2_ITLB_MISS=[0 0 0 0 0 0 0 0 0 0 0 0 0 ]
PMC1_BRANCH_MISPREDICT=[1 1 1 1 1 1 1 1 1 1 1 1 1 ]
PMC2_BPU_OUT=[2052 2052 1028 516 260 132 68 36 20 12 8 6 ]
```

Messergebnisse



Vergleich von Messungen und Vorhersage