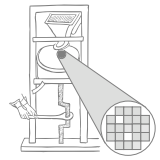


Eingebettete Prozessorarchitekturen

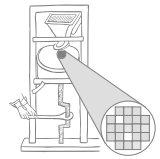
1. Einleitung

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen
Informatik, TU Darmstadt

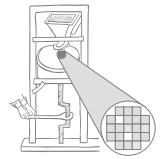
Wintersemester 2006/2007

Idee



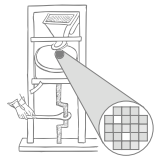
- Eingebettete Systeme (ES)
- Prozessor
  - . . . aber nicht zwangsläufig programmierbarer Mikroprozessor
  - Hier allgemeiner: datenverarbeitende Einheit
- Viele unterschiedliche Realisierungsmöglichkeiten
- Übersicht über vier Ansätze
  - Sicht aus dem Orbit
  - Viele Details weggelassen
- Trotzdem mit praktischem Anteil
  - Experimente am Rechner

Prüfung



- Lehrform: Integrierte Veranstaltung mit 2 SWS
- Vorlesungen und Übungen nach Bedarf gemischt
- Prüfung
  - Zwei vorlesungsbegleitende Teilklausuren
    - Zu je 45 Minuten
  - Erreichbare Punkte: Je 45
  - Geplant in KW 49 und KW 5
- Freiwillige Hausaufgaben
  - Ein Aufgabenblatt je Technologie
  - Erreichbare Punkte: Je 8
  - Erste Abgabe in KW 48
  - Abgabemodalitäten auf Aufgabenblättern

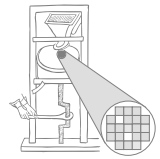
Hausaufgaben



Anrechnung der Hausaufgabenpunkte auf Klausurpunkte

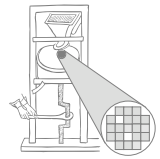
- Einfache Addition
- Kann Gesamtnote um bis zu einen Notenwert (1,0) verbessern
- Aber: Mindestens 32 Punkte müssen durch reine Klausurpunkte erreicht werden!

Rechnerzugang



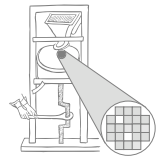
- Für Lösung der Hausaufgaben Zugang zu Software-Werkzeugen erforderlich
- Nicht auf RBG Pool-Rechnern verfügbar
- Nur auf Rechnern der FG ESA installiert
  - Ausnahme: Tools für erste Phase sind frei verfügbar
- Arbeit unter Linux
  - Gegebenenfalls vorher einarbeiten!
- Zugang von ausserhalb möglich (via SSH)
- Nach Absprache Arbeit auch im Praktikumsraum der FG ESA
- In jedem Fall für letzte Phase (echte Hardware)
- Weitere Details nächste Woche

Betreuung



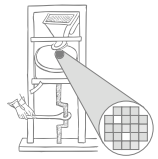
- Kein Tutor
  - . . . woher auch
  - Veranstaltung findet ja das erste Mal statt
- Allgemeine Fragen: Im Forum der Fachschaft stellen
  - Dann haben alle etwas von Antworten und Diskussionen
- Spezielle Fragen: Via E-Mail
- Wenn es hakt: In Sprechstunde kommen

Ablauf



1. Very Long Instruction Word-Prozessoren
   - Technologie von Hewlett-Packard und ST Microelectronics
   - Vier Wochen
2. Konfigurierbare Prozessoren
   - Technologie von Tensilica
   - Drei Wochen
3. Rekonfigurierbare Prozessoren
   - Technologie von Stretch
   - Drei Wochen
4. Adaptive Computer
   - Technologie von Xilinx und der FG ESA
   - Hier auch Experimente mit echter Hardware möglich
   - Drei Wochen

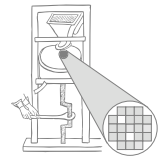
Ablauf



Was passiert in den letzten zwei Wochen?

- Wenn alles klappt
- . . . ist noch nicht ganz sicher
- Zwei Bonusvorträge
  - Nicht klausurrelevant
- Thema: IBM/Sony/Toshiba Cell-Processor
- Vortragende: Cell-Entwickler von IBM Böblingen
- Hardware: Prozessorarchitektur
- Software: Programmierung und Linux-on-Cell

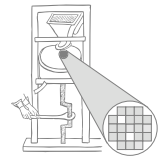
Robustheit der Veranstaltung



- Veranstaltung wird das erste Mal angeboten
- Brandaktuelle
  - Hardware-Architekturen
  - CAD-Werkzeuge
- Vorträge und Lehrmaterial noch in Entwicklung

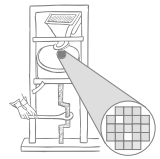
➥Beta-Version, es kann also manchmal holprig werden . . .

- Selber Dokumentation lesen
  - Steht komplett zur Verfügung
- In Experimenten Sachen ausprobieren
  - Auf Fehler und Probleme hinweisen
- Auch der Vortragende ist nicht allwissend!

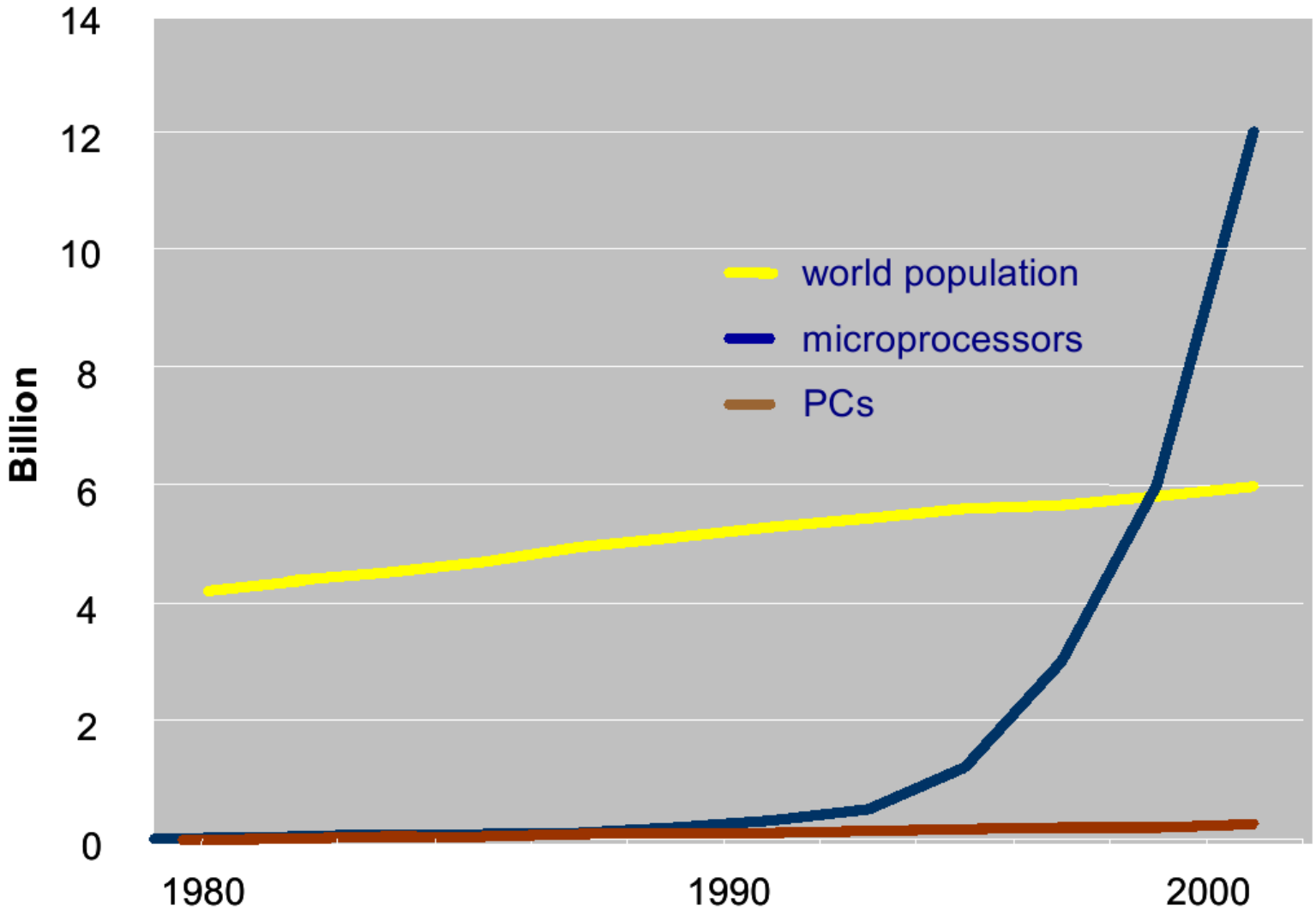


Eingebettete Systeme

Anzahl Mikroprozessoren

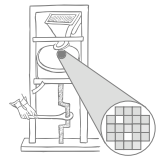


A. Koch

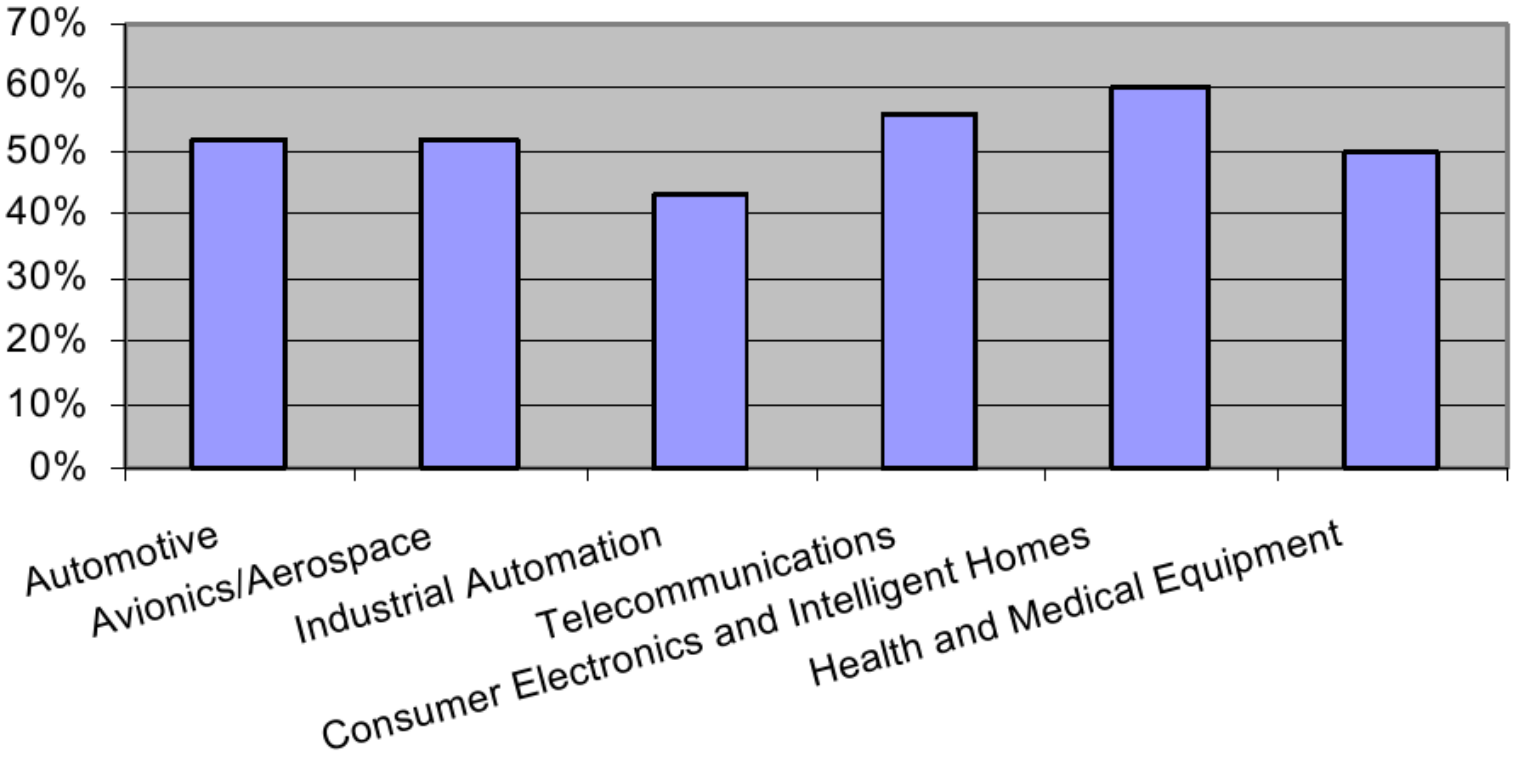


Quelle: FAST/TUM EU Report 25.11.2005

Anteil von ES an Produktkosten

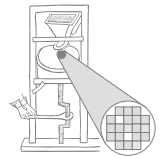


A. Koch



Quelle: FAST/TUM EU Report 25.11.2005

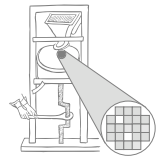
Eingenschaften von ES



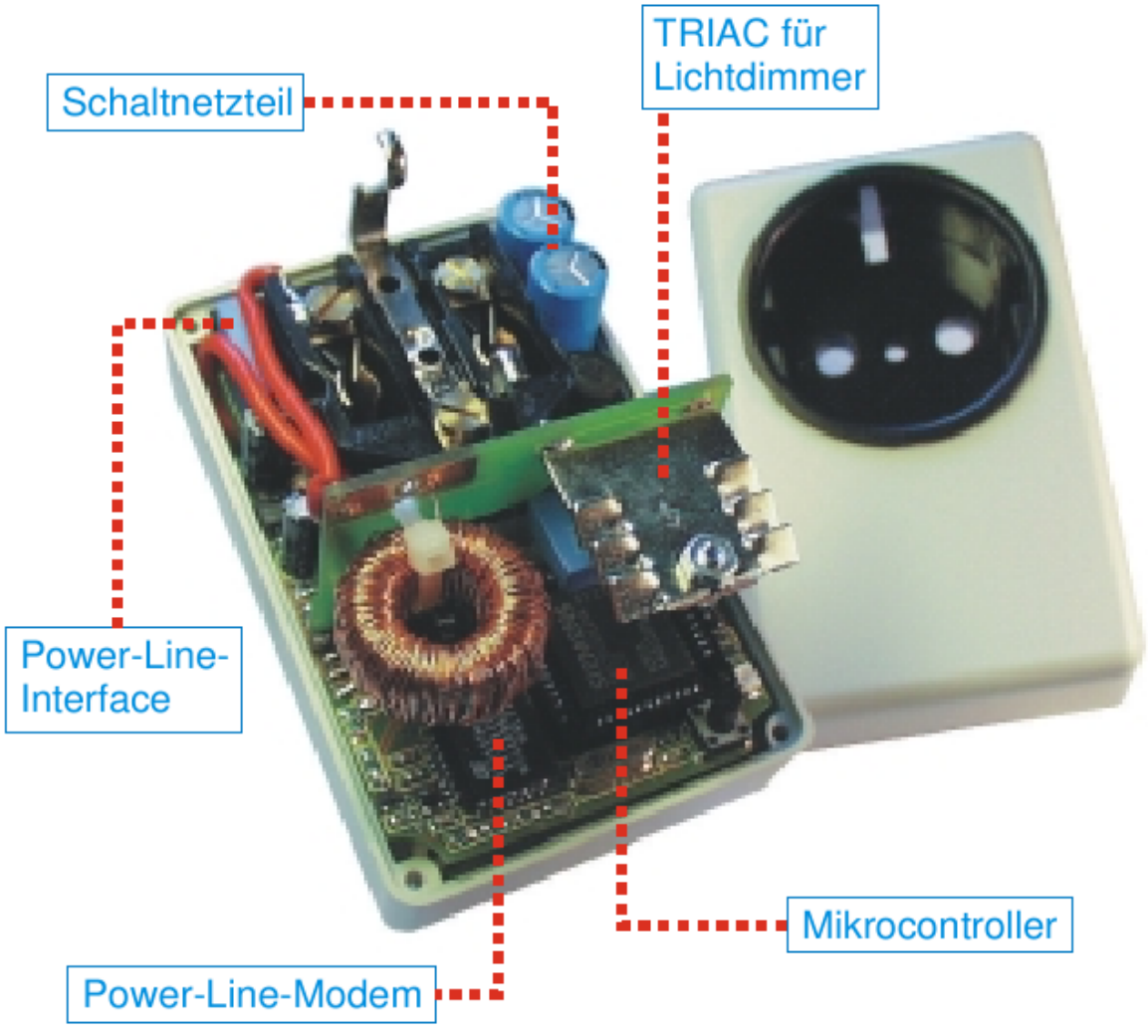
- Spezialisiert für bestimmte Aufgaben
  - Im allgemeinen nicht vom Benutzer veränderbar
- Oft harte externe Anforderungen
  - Rechenleistung
  - Energieverbrauch
  - Platzbedarf
  - Zuverlässigkeit
  - Umgebungsbedingungen
  - Kosten
- Häufig nicht mit Standardlösungen erfüllbar

ES: Fernschaltbare Steckdose

Von sehr klein und preiswert . . .



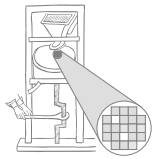
A. Koch



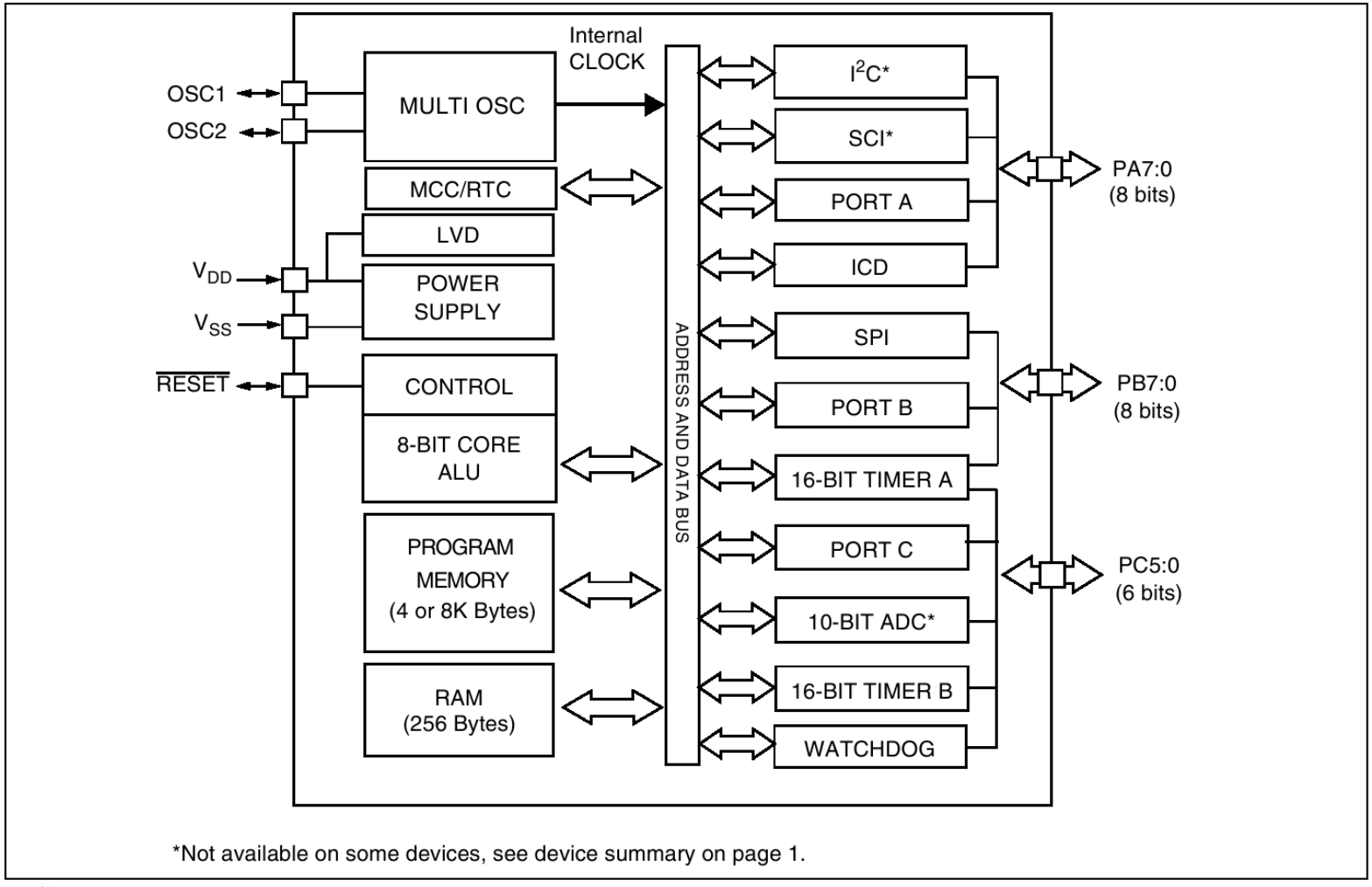
Quelle: Domologic

STMicroelectronics ST7 Microcontroller

On-Chip Komponenten



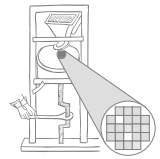
A. Koch



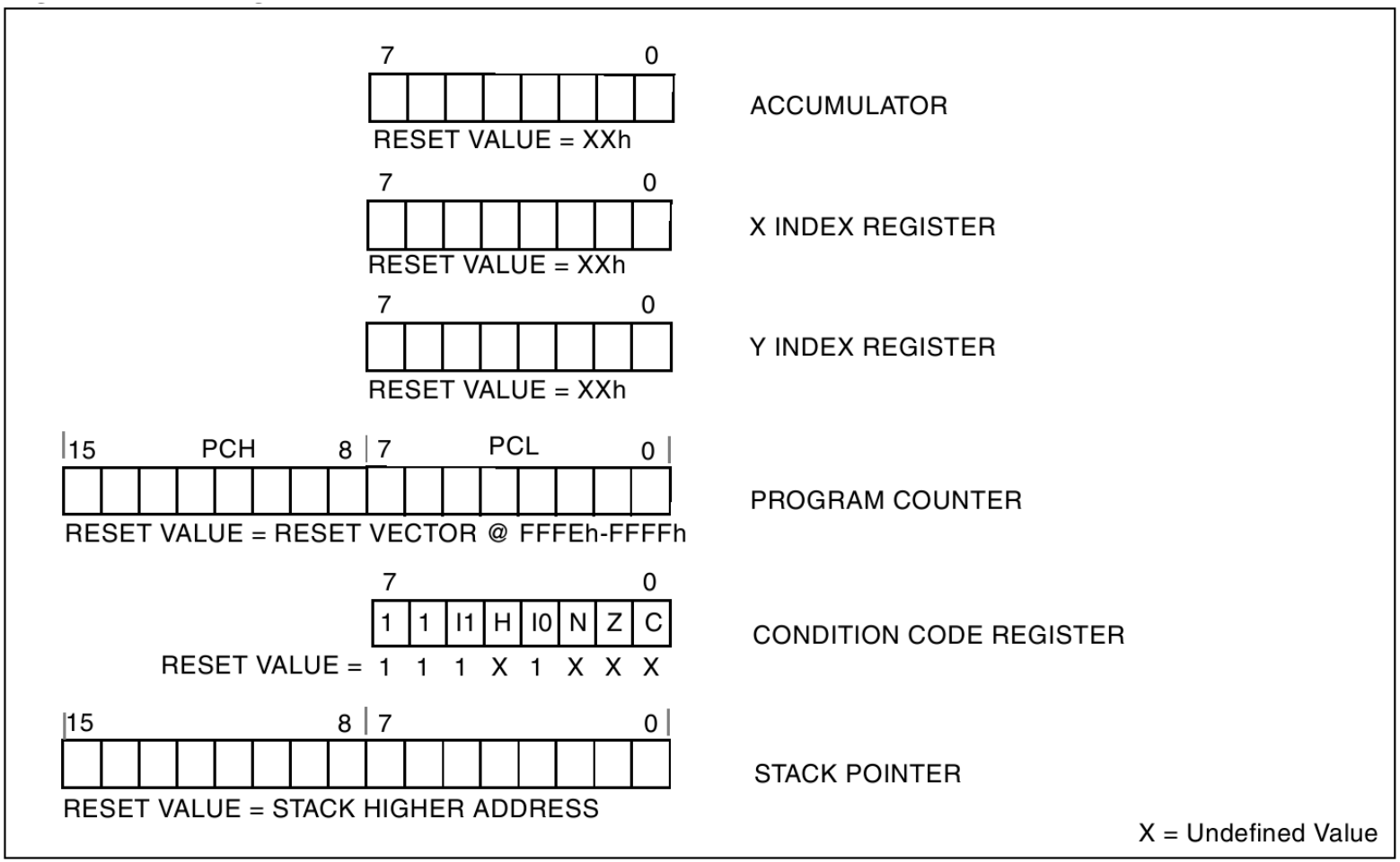
Quelle: STMicroelectronics Datasheet

STMicroelectronics ST7 Microcontroller

Instruction Set Architecture (ISA)



A. Koch



Quelle: STMicroelectronics Datasheet

Eingebettetes System: Internet Router

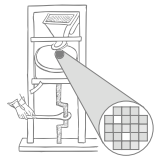
. . . bis extrem leistungsfähig

A. Koch

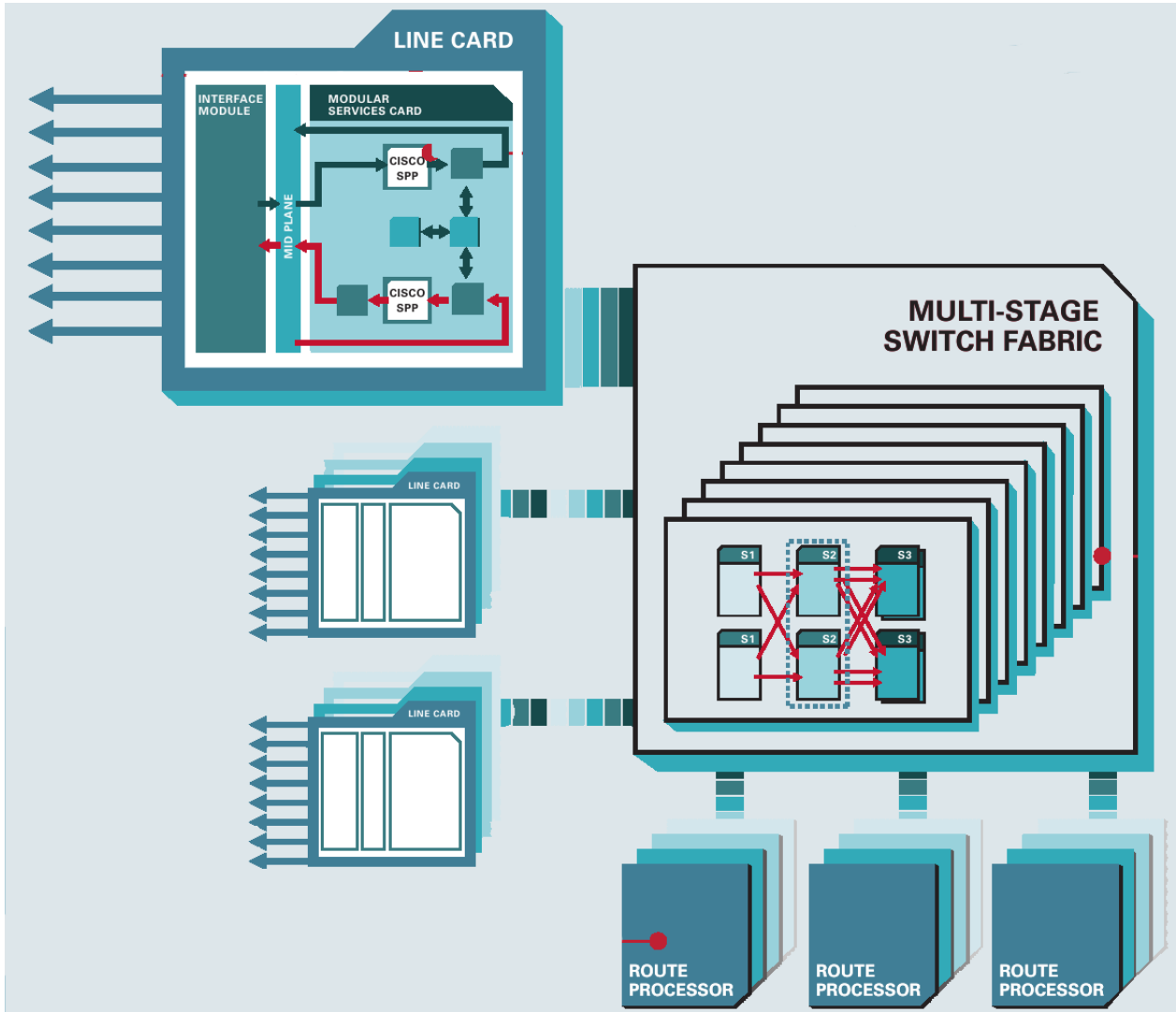
92 Tb/s Vermittlungskapazität

Quelle: Cisco Brochure

Cisco CRS-1
Systemarchitektur



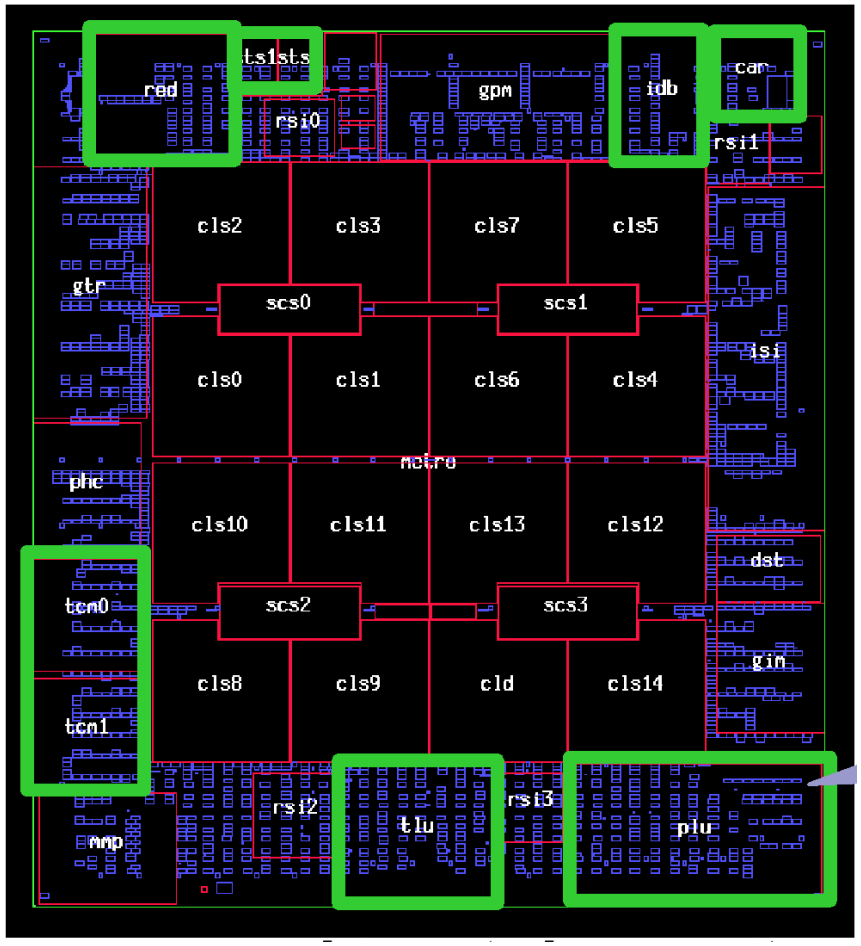
A. Koch



Quelle: Cisco Brochure

Cisco CRS-1

On-Chip Komponenten im Silicon Packet Processor (SPP)

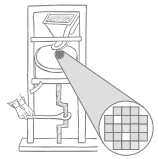


A. Koch

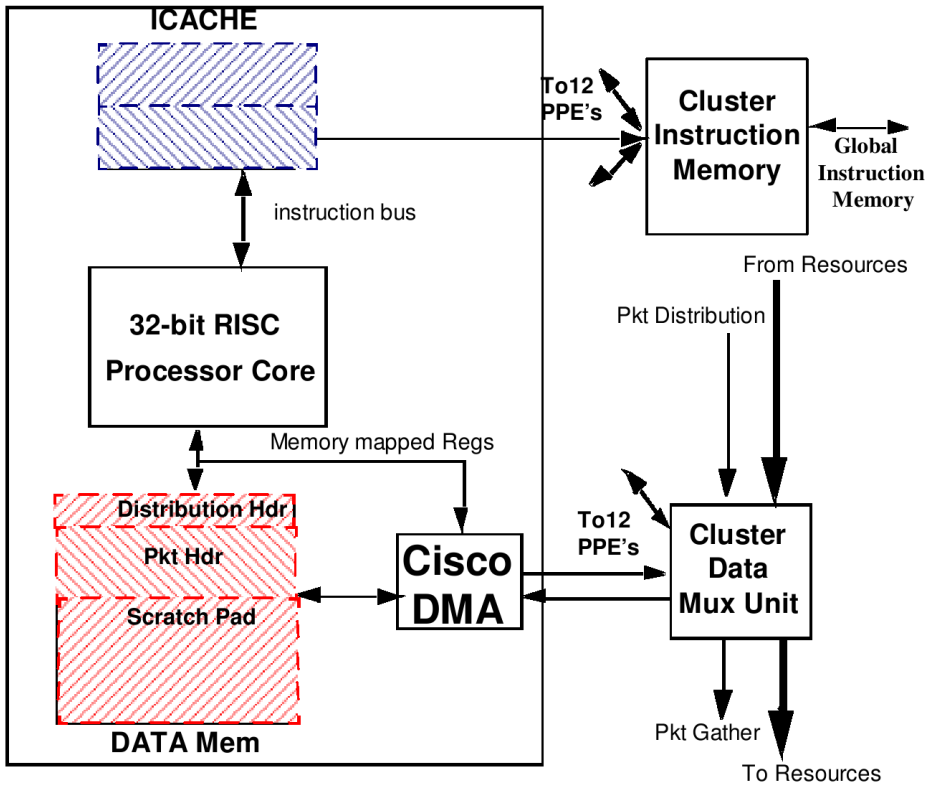
16 Cluster mit je 12 Packet Processing Elements (PPEs)

Quelle: W. Eatherton (Cisco), ANCS 2005 presentation

Cisco CRS-1
Packet Processing Element



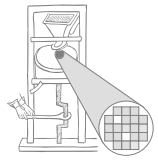
A. Koch



- Je PPE: 32b RISC-Prozessor ➜188 PPEs/Chip
  - Mit eigenem Speicher und Spezialhardware
- 2 Chips je Line Card
- Bis zu 40 Line Cards ➜Bis zu 15.040 PPEs pro System

Quelle: W. Eatherton (Cisco), ANCS 2005 presentation

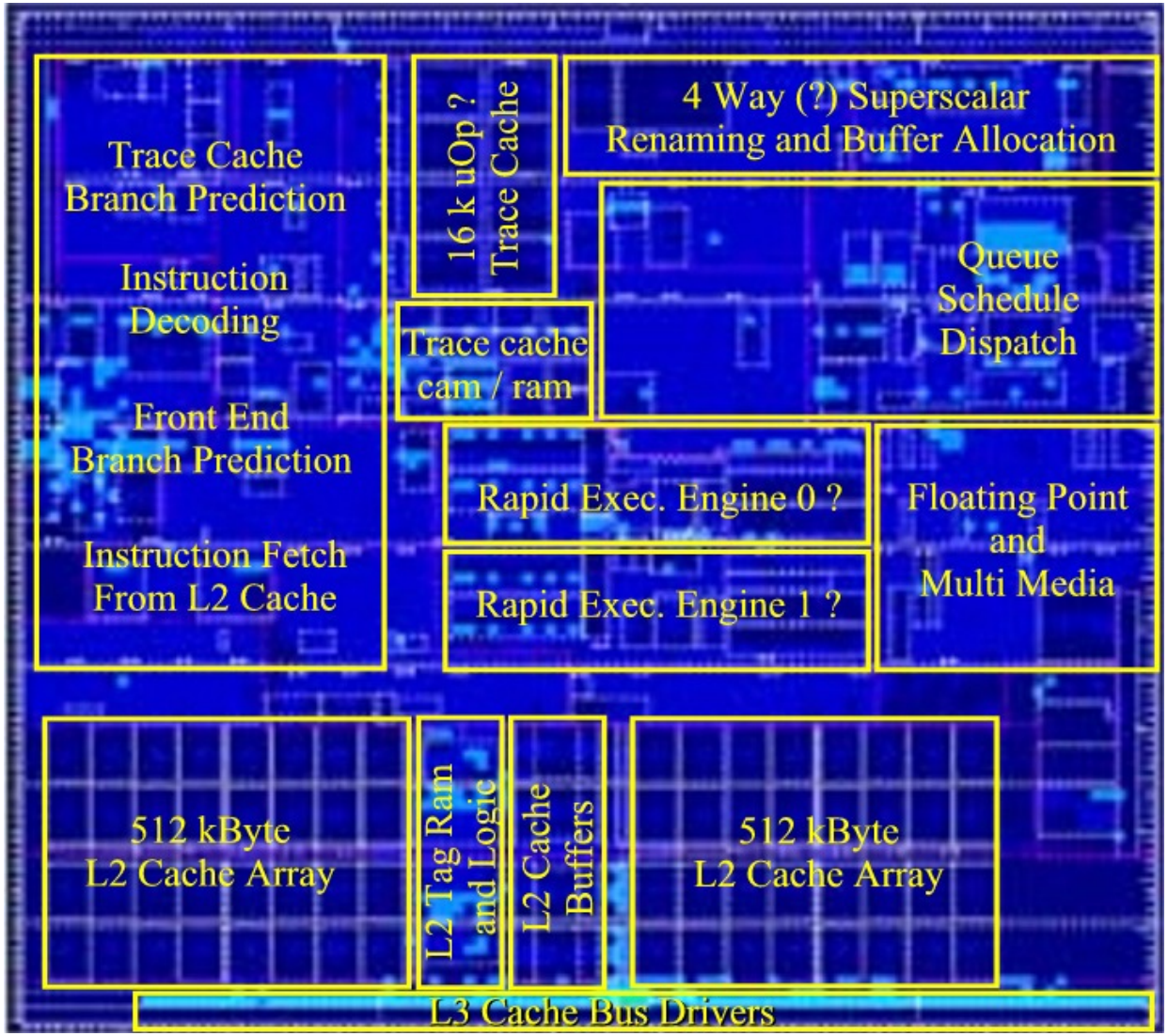
Ziel



- Effiziente Zurverfügungstellung von Rechenleistung
- Spezielle Architekturen
  - Systemebene
  - Prozessorebene

➥Nicht mehr “one size fits all”!

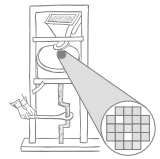
Standardprozessor: Intel P4 Prescott



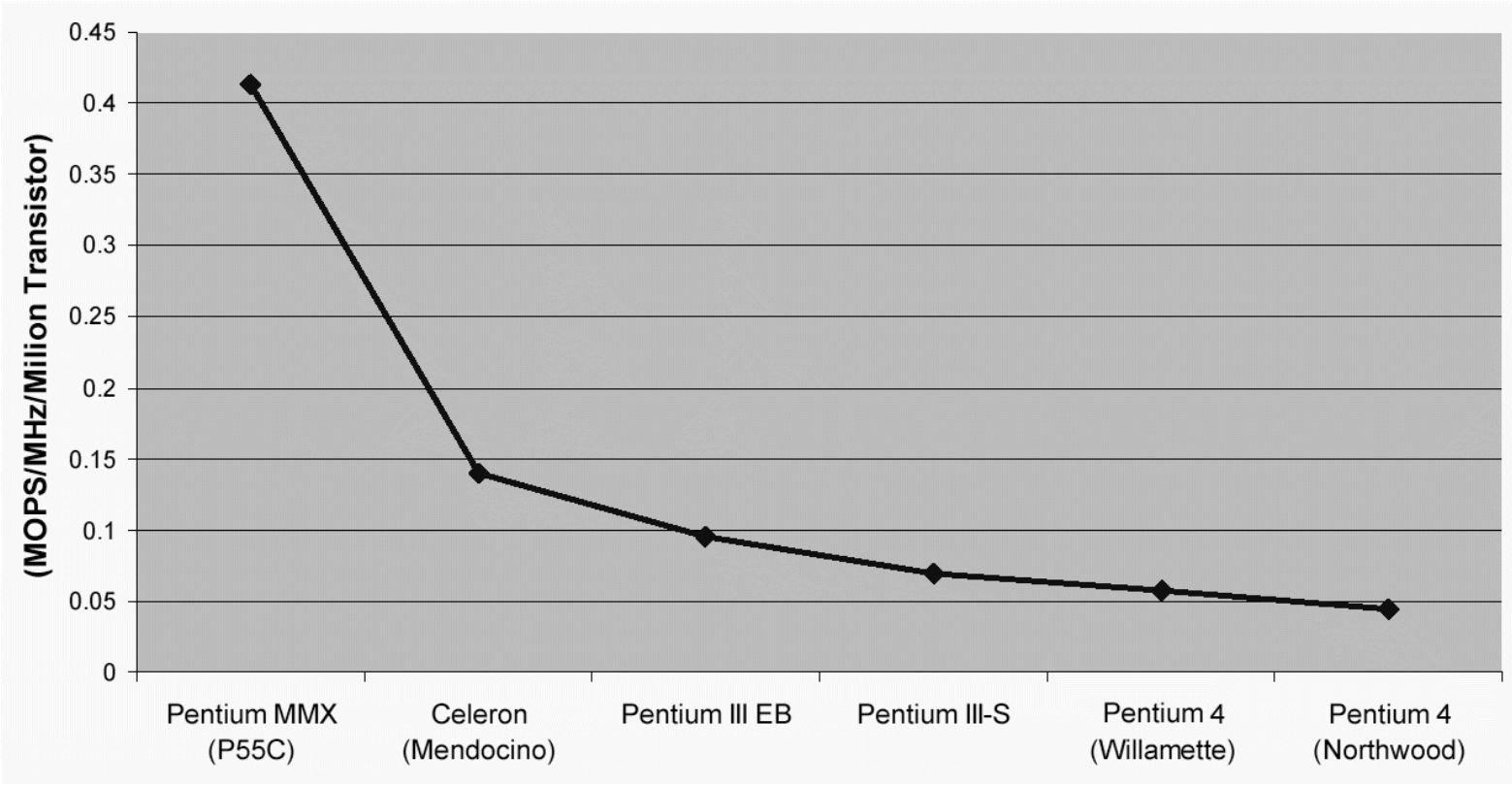
A. Koch

Quelle: H. de Vries

Effizienz von Standardprozessoren

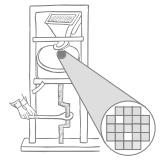


A. Koch



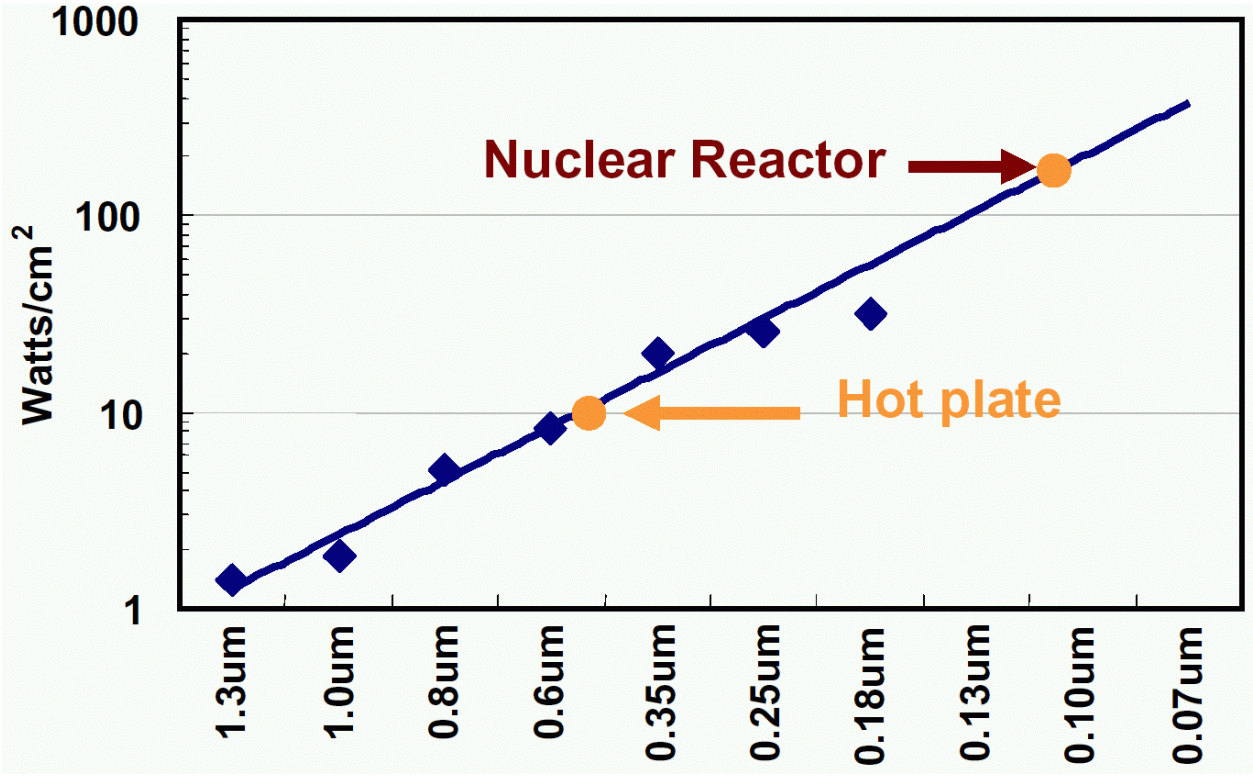
Quelle: John Wawrzynek, UCB

Energieverbrauch Leistungsdichte

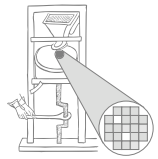


- Beim Umschalten von Transistoren
  - Immer höhere Taktfrequenzen also problematisch
- Leckströme
  - Extrem feine Chip-Strukturen verhindern vollständiges Abschalten von Transistoren

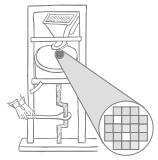
A. Koch



Lösungsansätze



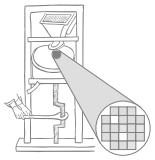
1. Anderer Ansatz für Universalprozessoren
   - Weniger Fläche für Verwaltungsaufgaben
2. Spezialisiere Universalprozessoren
   - Feste anwendungsspezifische Instruktionen
   - Variable anwendungsspezifische Instruktionen
3. Veränderbare Prozessor- und Systemarchitektur



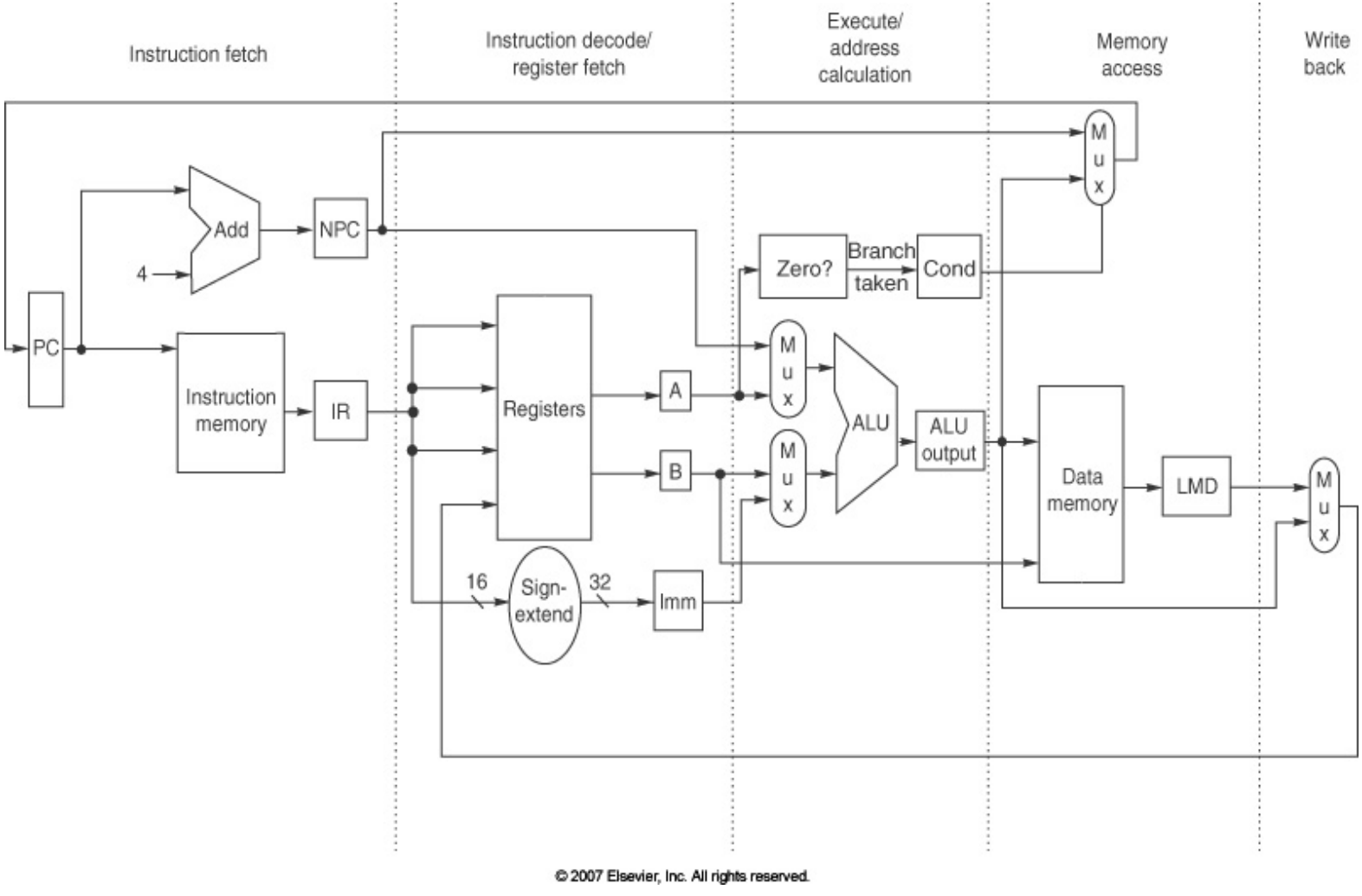
Höhere Rechenleistung durch parallele Ausführung von Operationen

Wiederholung: Einfache Prozessor-Pipeline

GDI3 / TGDI2

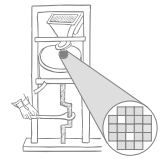


A. Koch



Quelle: Hennessy/Patterson CA:QA 4e

Ausnutzung von Pipeline-Parallelität



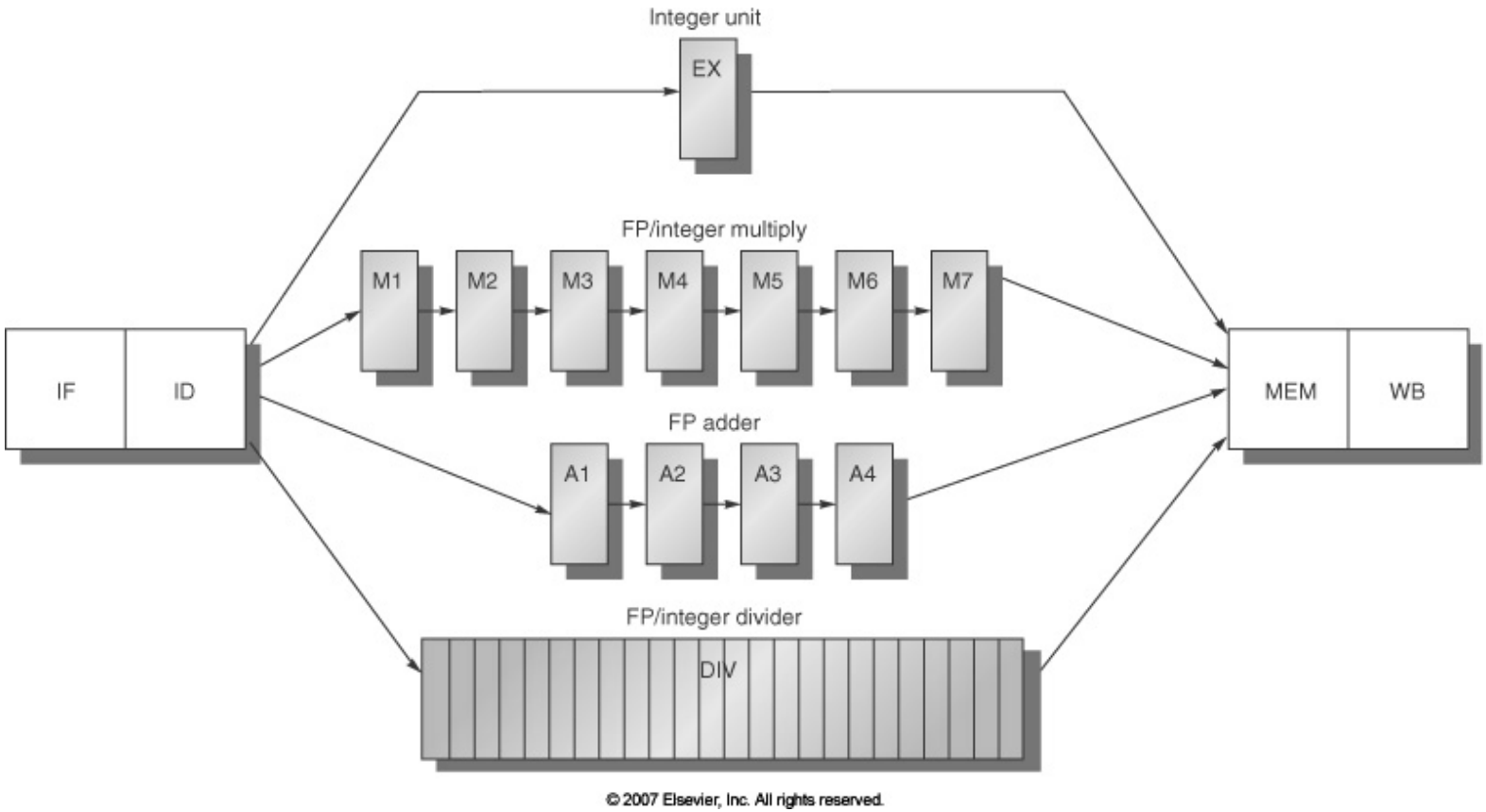
A. Koch

	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

Quelle: Hennessy/Patterson CA:QA 3e

Idee: Nicht nur eine ALU in EX-Stufe

Mehrere spezialisierte Recheneinheiten

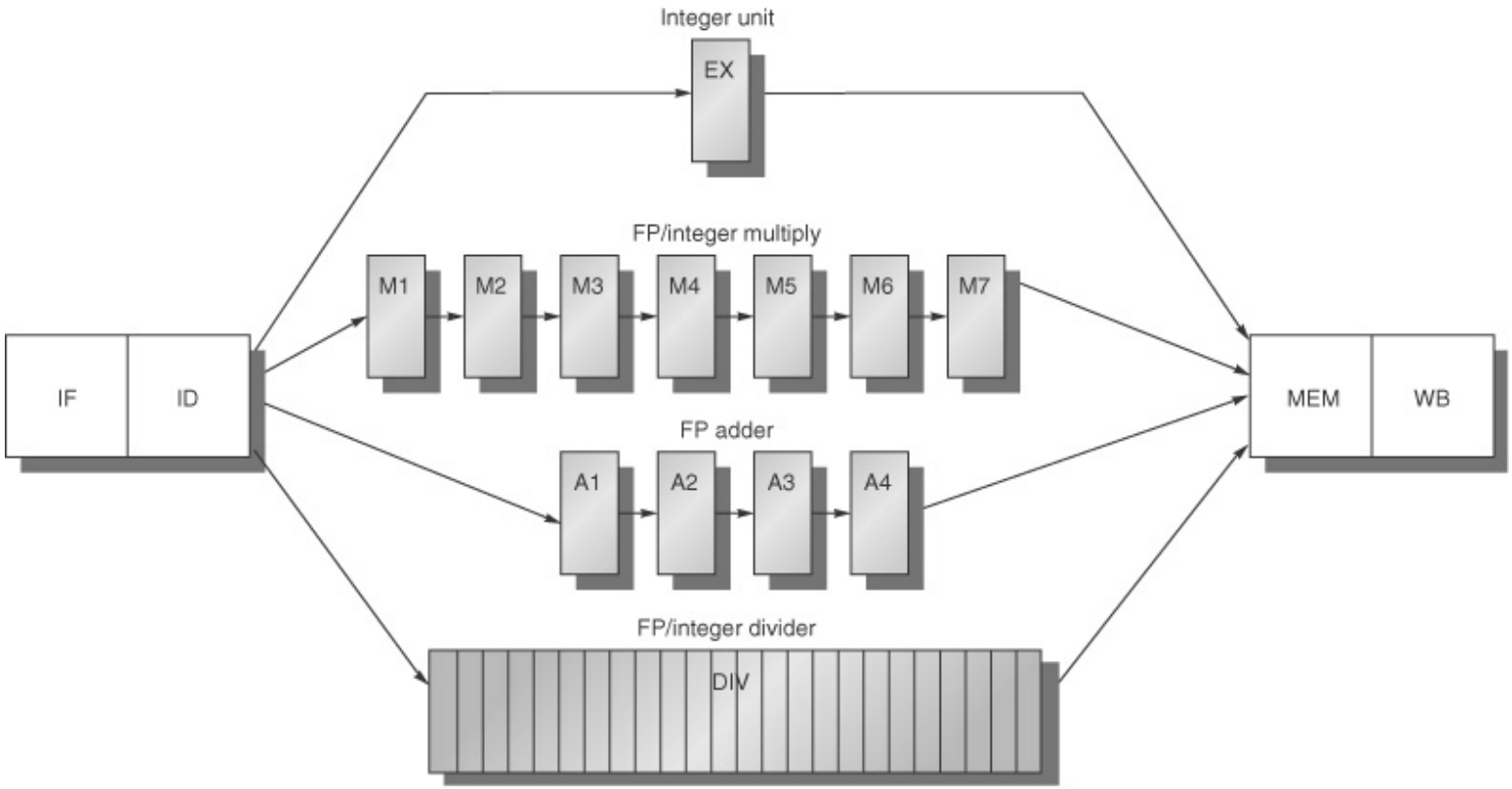


A. Koch

Aber nicht alle pipelined (z.B. Div zu groß . . .)

Quelle: Hennessy/Patterson CA:QA 4e

Jetzt mehrere Instruktionen parallel ausführen



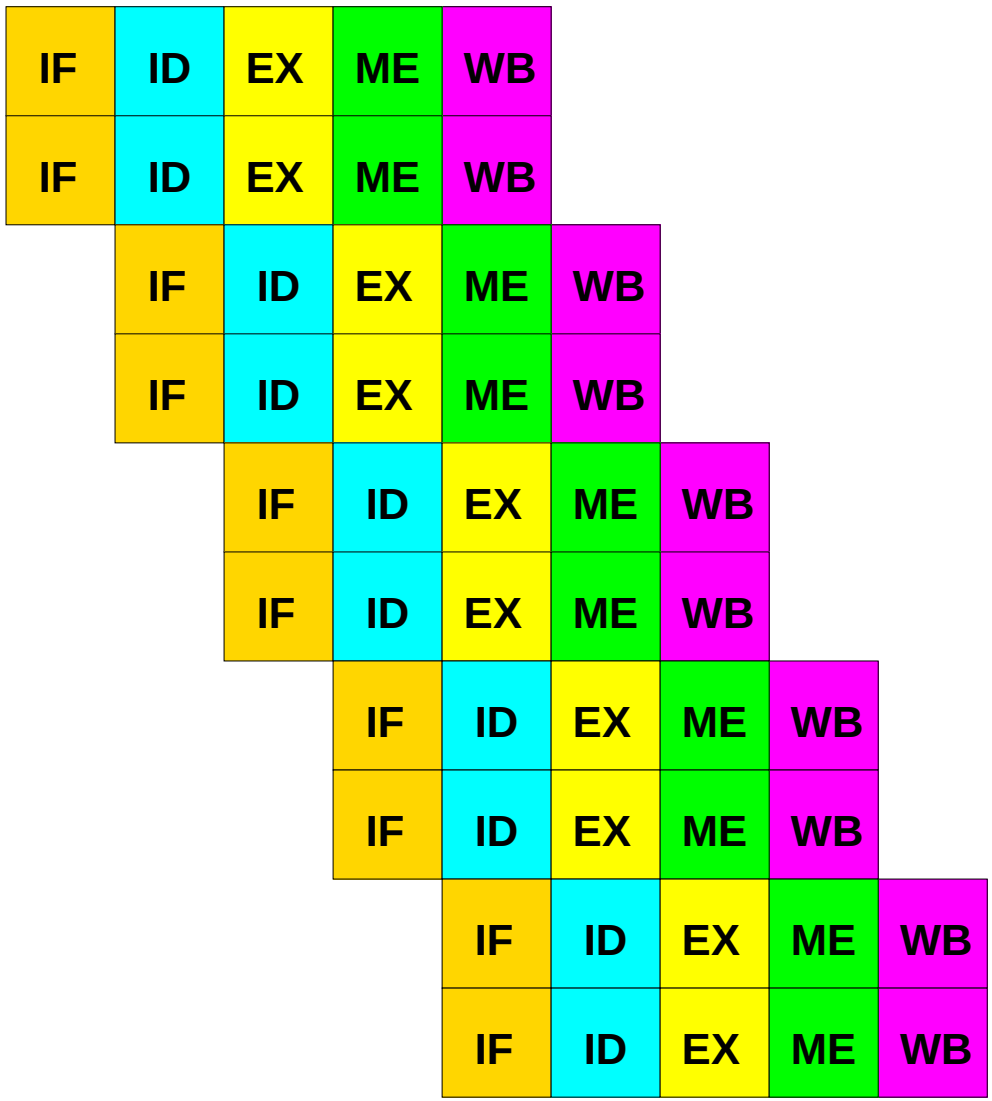
MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	**A4**	MEM	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

Quelle: Hennessy/Patterson CA:QA 3e

A. Koch

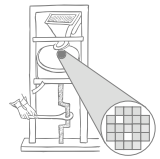
Superskalare Architektur

Hier mit zwei parallelen Pipelines



A. Koch

Wie ausnutzen?

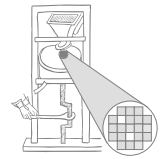


A. Koch

```
addi $t0, $t4, 1  ; L1
addi $t1, $t5, 2  ; L2
addi $t2, $t6, 3  ; L3
addi $t3, $t7, 4  ; L4
```

- Hier parallel ausführbar auf Beispielarchitektur
- Je zwei beliebige Anweisungen
- Keine Abhängigkeiten zwischen Einzelinstruktionen
- Wenn ausreichend viele Recheneinheiten vorhanden
  - Alles parallel
  - Würde im Beispiel aber nicht klappen
  - Nur zwei ALUs in Pipelines
  - Auch bezeichnet als *structural hazard*

Datenabhängigkeiten

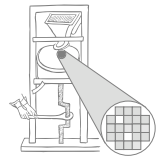



```
addi $t0, $t3, 1
addi $t1, $t0, 2
addi $t2, $t1, 3
```

- Nicht mehr parallel ausführbar
- Zwischenergebnisse müssen berechnet werden
  - Via t0 und t1

➥Datenabhängigkeiten verhindern parallele Ausführung

Arten von Datenabhängigkeiten



A. Koch

```
addi $t0, $t3, 1
addi $t1, $t0, 2
```

Echte Abhängigkeit (read-after-write)

```
addi $t0, $t3, 1
addi $t0, $t1, 2
```

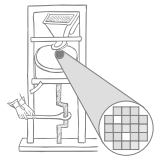
Ausgabeabhängigkeit (write-after-write)

```
addi $t0, $t3, 1
addi $t3, $t1, 2
```

Antiabhängigkeit (write-after-read)

Auflösen von Datenabhängigkeiten

Teilweise durch Umbenennen von Registern möglich



A. Koch

Vorher

```
addi    $t0, $t3, 1    ; L1
sw      $t0, 4($t1)    ; L2
addi    $t0, $t2, 2    ; L3
sw      $t0, 8($t1)    ; L4
```

- Ausgabeabhängigkeit (WAW) L1,L3

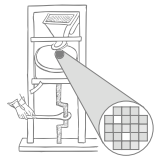
Nachher

```
addi    $t7, $t3, 1    ; L1
sw      $t7, 4($t1)    ; L2
addi    $t0, $t2, 2    ; L3
sw      $t0, 8($t1)    ; L4
```

- t0 teilweise in t7 umbenannt
- Ausgabeabhängigkeit aufgelöst
- Aber noch Datenabhängigkeit (RAW) L1,L2 und L3,L4

Mehr Parallelität durch Umstellen

Verschieben von datenunabhängigen Anweisungen



A. Koch

Vorher

```
addi    $t7, $t3, 1    ; L1
sw      $t7, 4($t1)    ; L2
addi    $t0, $t2, 2    ; L3
sw      $t0, 8($t1)    ; L4
```

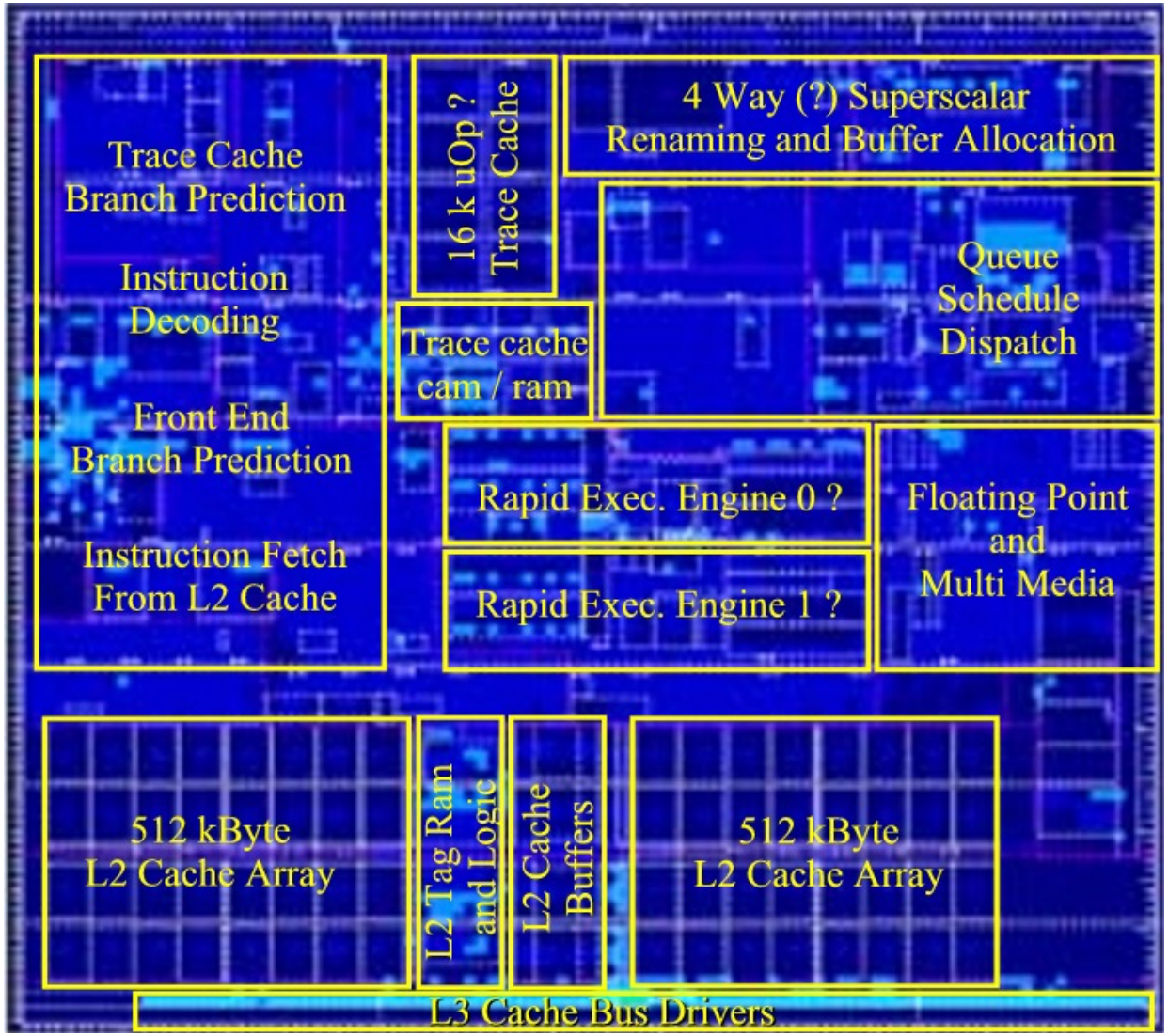
- RAW L1,L2 und L3,L4
- Nicht parallel ausführbar
  - Auf superskalarer Beispielarchitektur

Nachher

```
addi    $t7, $t3, 1    ; L1
addi    $t0, $t2, 2    ; L3
sw      $t7, 4($t1)    ; L2
sw      $t0, 8($t1)    ; L4
```

- L3 und L2 umsortiert
- Nicht datenabhängig
- Nun parallel ausführen: L1,L3 und L2,L4

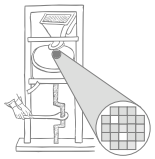
Standardprozessor: Intel P4 Prescott



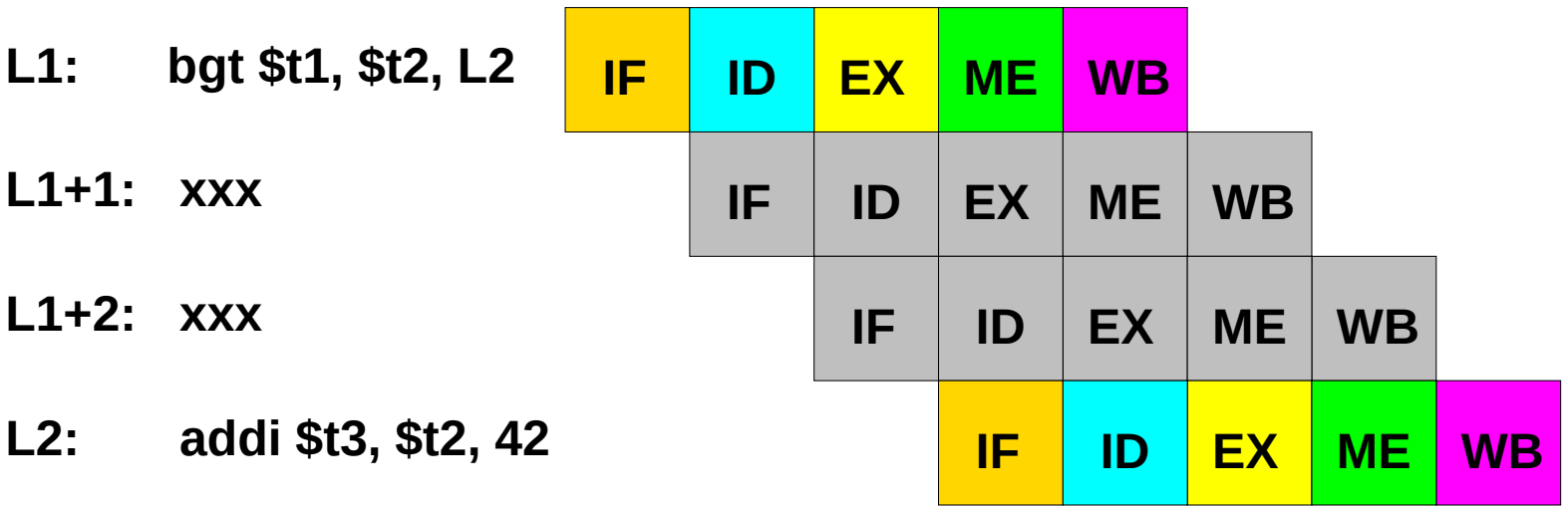
A. Koch

Fenster zum Umsortieren: Teilweise über 100 Instruktionen

Kontrollabhängigkeiten
Durch bedingte Sprünge

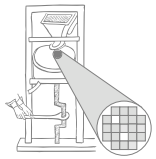


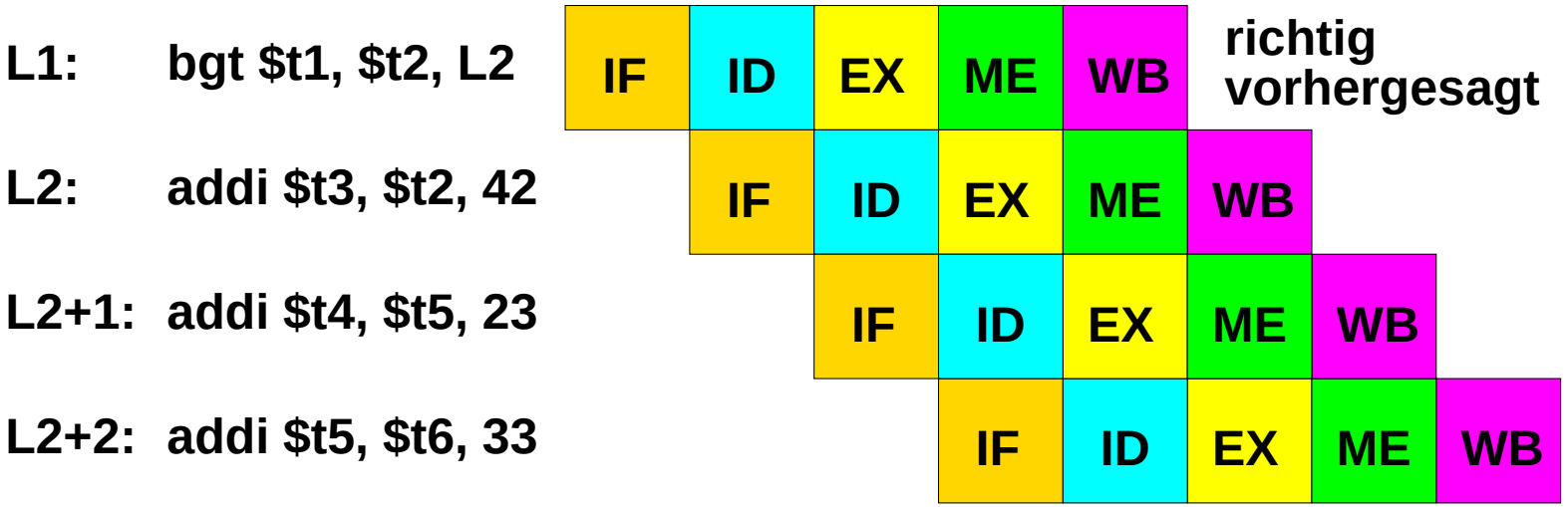
A. Koch



- Annahme: Sprungentscheidung wird am Ende von EX bestimmt
- Zwei verschwendete Instruktionen
- Verschwendete Zeit wächst mit Pipeline-Länge
- Intel P4 Cedar Mill: 31 Stufen

Abhilfemöglichkeit: Sprungvorhersage

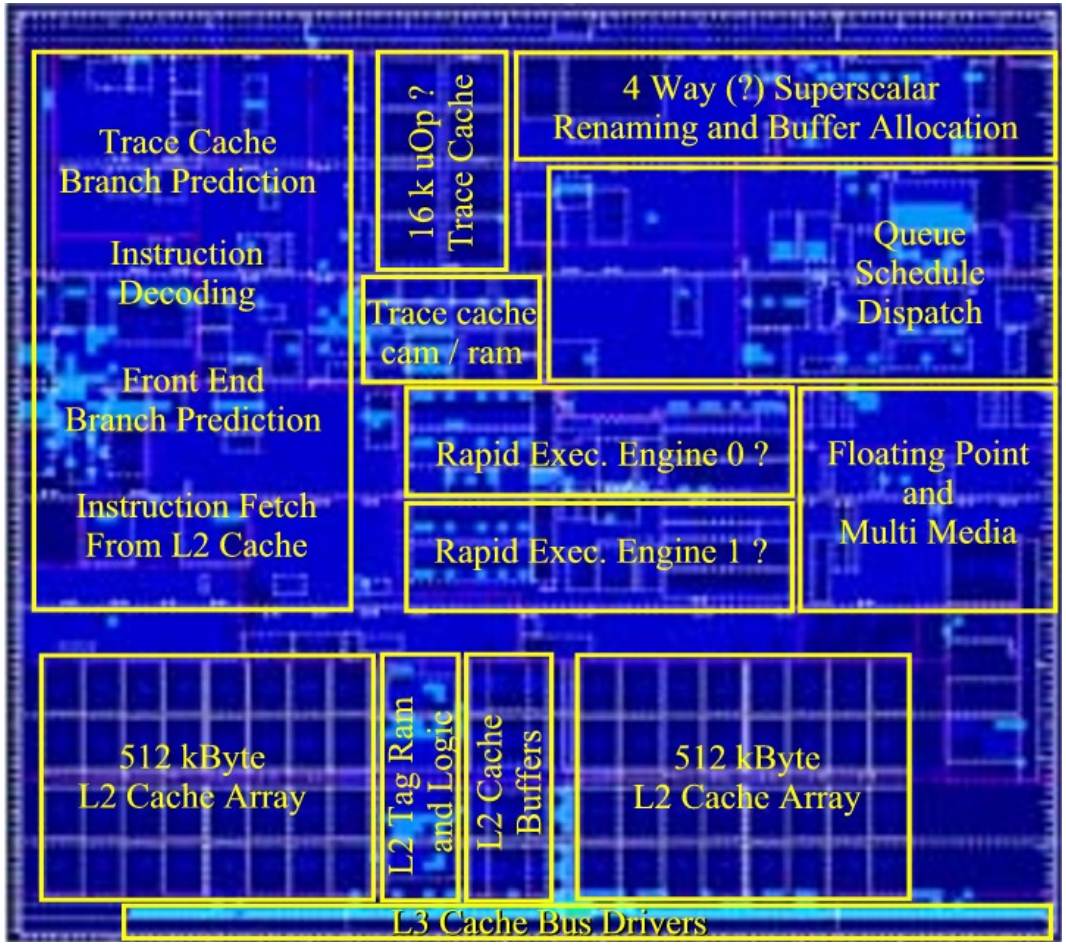




A. Koch

- Für jede Sprunginstruktion Historie führen
  - Genommen oder nicht genommen?
- Bessere Verfahren
  - Berücksichtigen global Folgen von vorhergehenden Sprunginstruktionen
  - Speichern Adresse des Sprungziels
    - *branch target buffer*, besser: *branch target cache*
- Erreichen $< 3\%$ falsch vorhergesagte Sprünge

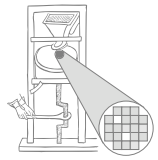
Standardprozessor: Intel P4 Prescott



A. Koch

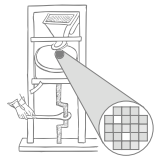
- Branch-Target-Buffer mit 4096 Einträgen
- Kombinierte globale/lokale Sprungvorhersage
  - Turnierverfahren

Kombinieren der Techniken



- Soweit erledigt
  - Auflösen von unechten Abhängigkeiten durch Umbennenen von Registern
  - Umsortieren von Anweisungen für mehr Parallelität
  - Reduzieren der Kosten von bedingten Sprüngen
- Aber Einschränkung:
  - Anweisungen können nicht über bedingte Sprünge hinweg verschoben werden
  - Compiler: Keine Optimierung über Basisblöcke hinweg

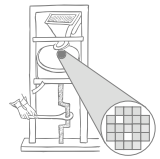
Beispiel




```
addi    $t0, $t3, 1    ; L1
bgt     $t0, $t2, L4   ; L2
addi    $t1, $t2, 42   ; L3
sw      $t0, 0($t5)    ; L4
sw      $t1, 4($t5)    ; L5
```

- L1 und L2 nicht gleichzeitig (RAW)
- L1 und L3 wären gleichzeitig möglich
- Wird vereitelt durch dazwischenliegende Sprunginstruktion

Abhilfe: Spekulative Ausführung

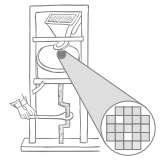


A. Koch

```
addi    $t0, $t3, 1    ; L1
bgt     $t0, $t2, L4   ; L2
addi    $t1, $t2, 42   ; L3
sw      $t0, 0($t5)    ; L4
sw      $t1, 4($t5)    ; L5
```

- L1 und L3 doch gleichzeitig ausführen
- L3 wird dabei spekulativ ausgeführt
- Falls anschliessender Sprung über L3 hinweg
  - Alten Wert von t1 wiederherstellen
  - Spätestens bei L5 (kein spekulatives Schreiben!)

Zusammenfassung



- Organisatorisches
- Spektrum eingebetteter Systeme
- Spezialisierte ./. Standardprozessoren
- Parallele Ausführung von Operationen
  - Auf Instruktionsebene
  - Abhängigkeiten
  - Techniken zum Auflösen
  - Aufwand in Hardware