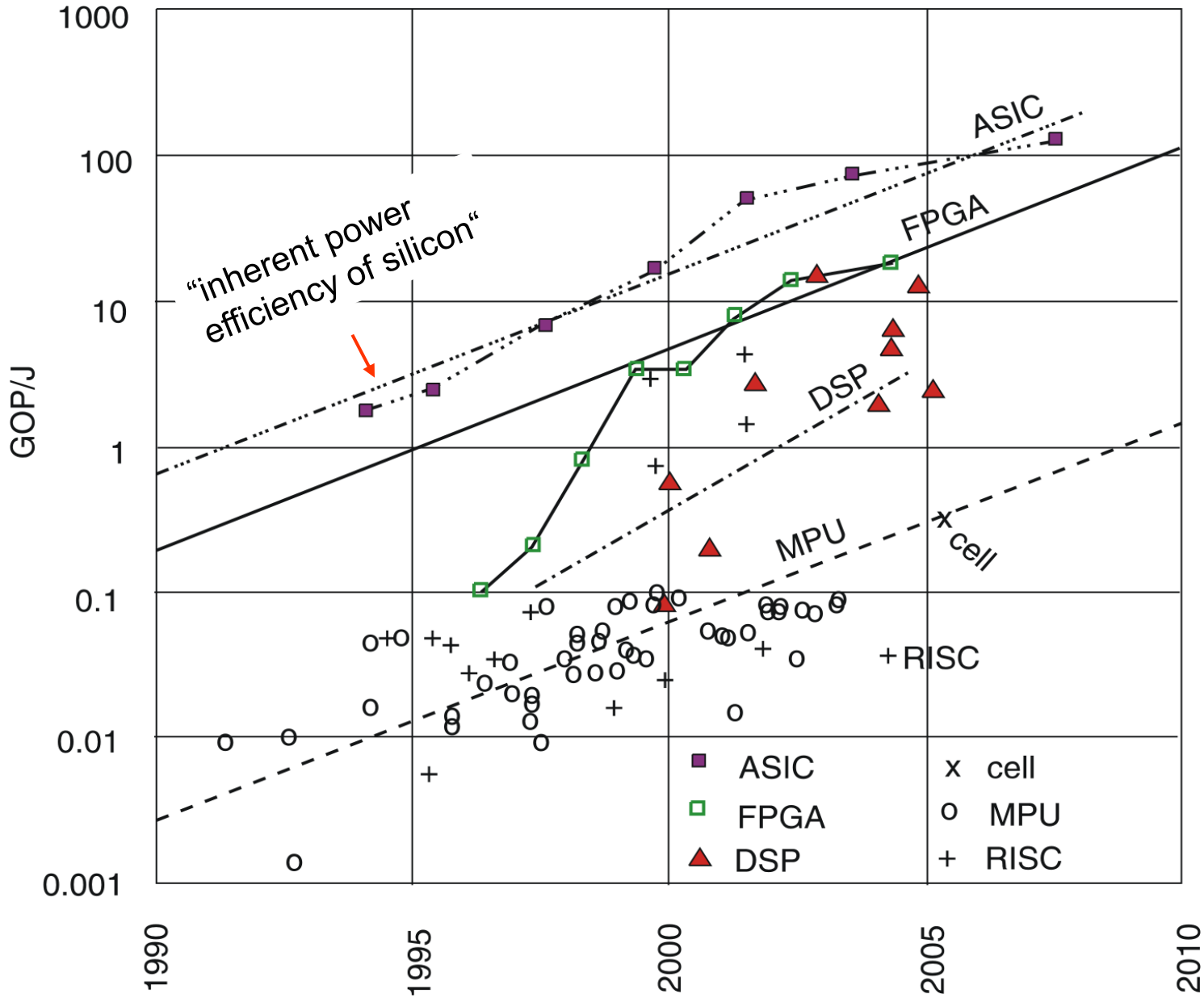
2.4 VLIW und EPIC-Prozessoren

Peter Marwedel
Informatik 12
TU Dortmund

2014年 04 月 16 日

Bedeutung der Energie-effizienz (und von thermischen Effekten)



© Hugo De Man, IMEC, Philips, 2007

Fundamentale Eigenschaften von CMOS

Leistungsverbrauch von CMOS (ohne Leckströme)

$P = \alpha\ C_L\ V_{dd}^2\ f$ mit

α : Schaltaktivität

C_L : Lastkapazität

V_{dd} : Betriebsspannung

f : Taktfrequenz

Verzögerung von CMOS Schaltungen

$$\tau = k\, C_L \frac{V_{dd}}{(V_{dd} - V_t)^2}$$ mit

V_t : Schwellspannung

$(V_t < V_{dd})$

☞ Verkleinerung von V_{dd} reduziert P quadratisch*, während die Laufzeit linear zunimmt

** Lt. Besuch bei IBM, Böblingen, 2013, in der Praxis eher kubisch*

Gründe für die Einführung von Parallelität

- **Rechnen mit niedrigen Taktraten energetisch effizienter und kühler:**

Basisgleichungen

Leistung: $P \sim V_{DD}^2$,

Maximale Taktfrequenz: $f \sim V_{DD}$,

Energiebedarf für ein Programm: $E = P \times t$, mit: t = Laufzeit (fest)

Zeitbedarf für ein Programm: $t \sim 1/f$

Änderungen durch Parallelverarbeitung, mit α Operationen pro Takt:

Taktfrequenz reduziert auf: $f' = f / \alpha$,

Spannung kann reduziert werden auf: $V_{DD}' = V_{DD} / \alpha$,

Leistung für Parallelausführung: $P^\circ = P / \alpha^2$ pro Operation,

Leistung für α Operationen pro Takt: $P' = \alpha \times P^\circ = P / \alpha$,

Zeit zur Ausführung des Programms: $t' = t$,

Energie zur Ausführung des Programms: $E' = P' \times t = E / \alpha$

Zeigt Tendenz, aber diverse grobe Näherungen!

Was soll parallel ausgeführt werden und wer entscheidet?

Prinzipielle „Entscheidungsträger”:

- Hardware

➡ Beibehaltung der Befehlssatzschnitte sequentieller Prozessoren mit automatischer Erkennung möglicher Parallelität ☞ Kap. 3

Superskalare Prozessoren, im Mittel CPI<1

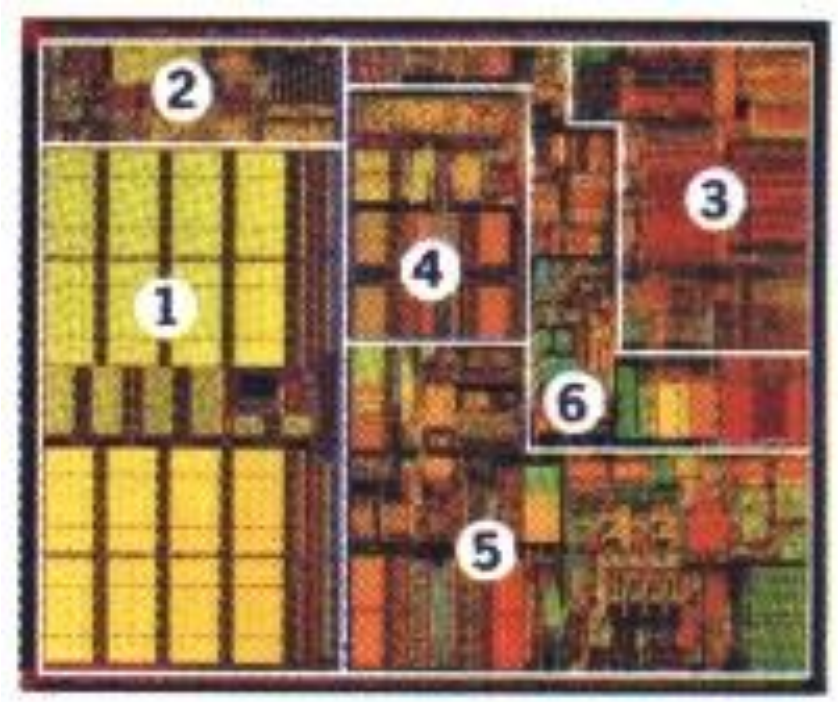
- Software (Compiler/Programmierer)
  - Lange Befehlsworte (VLIW/EPIC, Abschn. 2.4)
  - Parallele Operationen auf Daten (SIMD, Abschn. 2.5)
  - Multi-cores (Kap. 5)

Erkennung möglicher Parallelität in Hardware

Superskalare Prozessoren verwenden einen Großteil der Hardware darauf, mögliche Parallelarbeit zu erkennen.

1. Cache
2. Kommunikation mit Umgebung
3. Rechenwerk
4. Cache
5. **Umsortieren (dyn. Scheduling) der Befehle**
6. **Buchführung über Speicherort von Daten**

[Bode]



Pentium III; © Intel; Bode TUM

Performanzsteigerung mit Einzelprozessoren

Kaum noch möglich, die Rechenleistung bei sequentieller Ausführung zu steigern

- >1 Befehl pro Takt starten (Superskalare Prozessoren): schwierig, fehlerfrei zu realisieren: „*the only ones in favor of superscalar machines are those who haven't built one yet*“ [late Bob Rau (hp Labs)]

[www.trimaran.org]

- Noch mehr Fließbandstufen und kürzere Laufzeiten innerhalb einer Stufe: bereits hoher Anteil an *Pipelining*-Fehlern bei Silizium-*Bugs*
- Taktraten aus energetischen Gründen können kaum noch erhöht werden

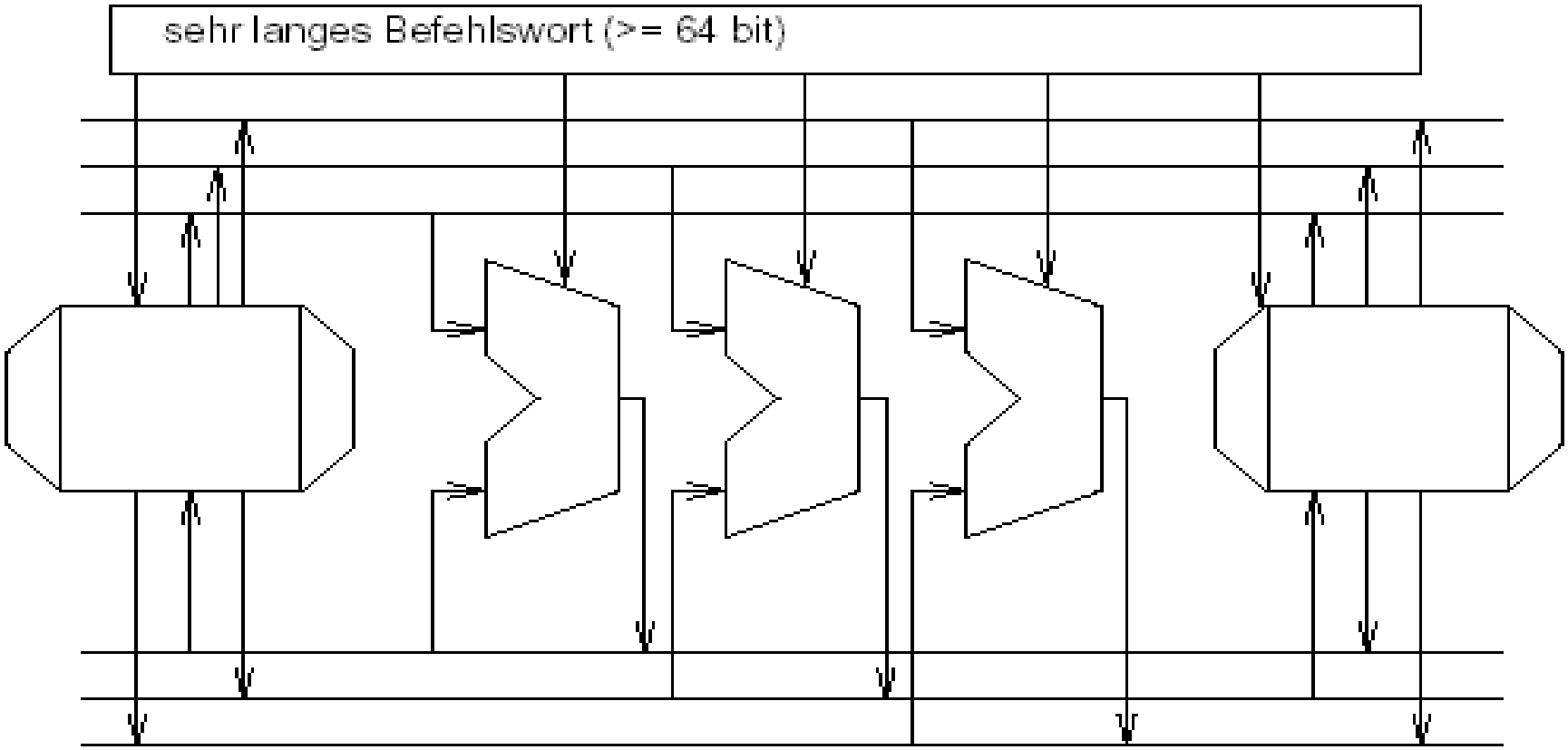
Was soll parallel ausgeführt werden und wer entscheidet?

Prinzipielle „Entscheidungsträger”:

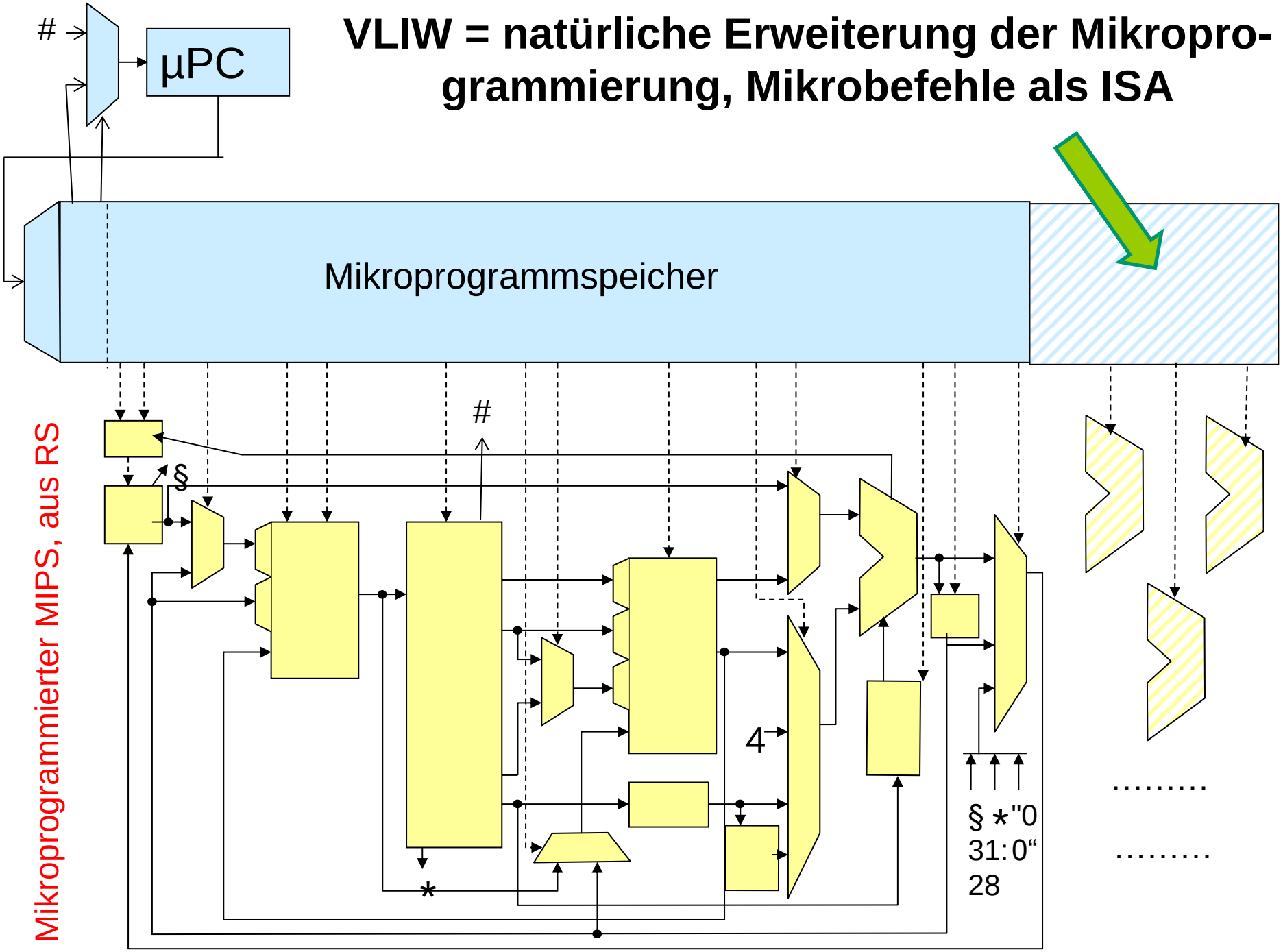
- Hardware
  Beibehaltung der Befehlssatzschnitte sequentieller Prozessoren mit automatischer Erkennung möglicher Parallelität ☞ Kap. 3
  Superskalare Prozessoren, im Mittel CPI<1
- Software (Compiler/Programmierer)
  - Lange Befehlsworte (VLIW/EPIC, Abschn. 2.4)
  - Parallele Operationen auf Daten (SIMD, Abschn. 2.5)
  - Multi-cores (Kap. 5)

Grundidee von VLIW

Prinzip: Operationen im Befehlspaket sprechen mehrere Recheneinheiten bzw. Registerblöcke an



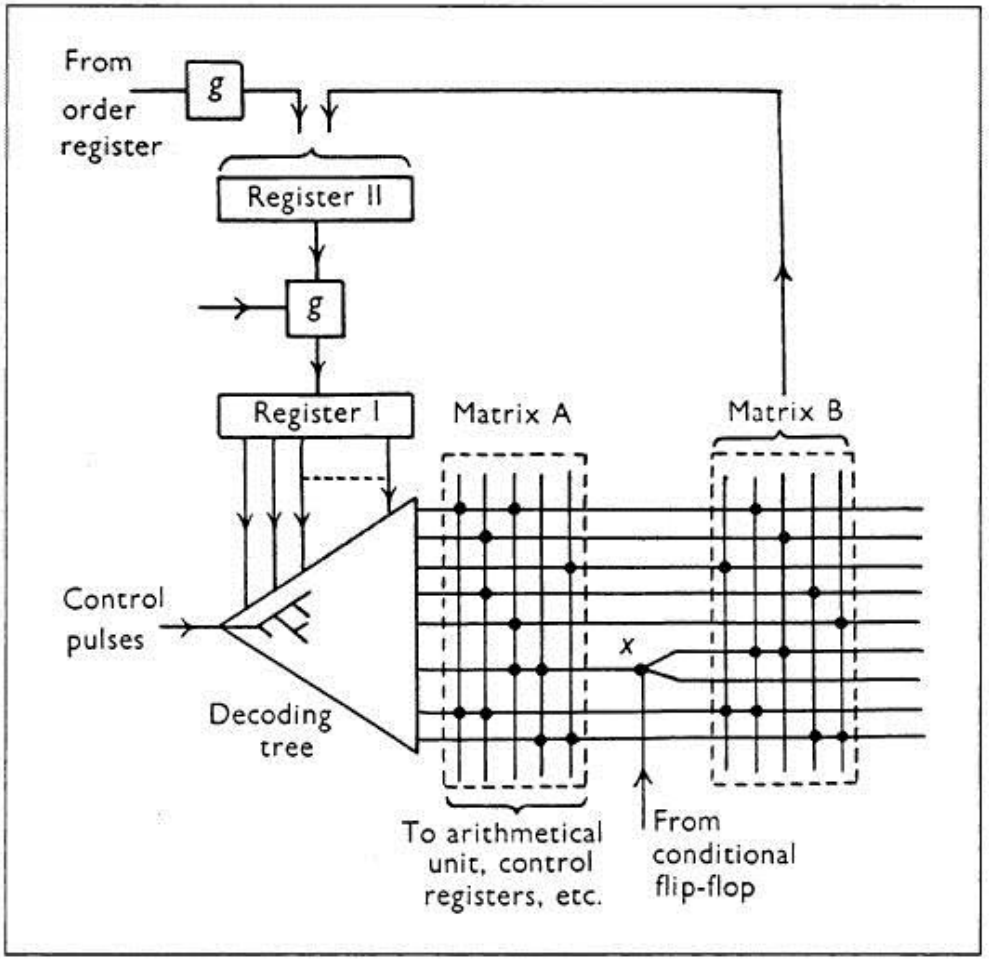
VLIW = natürliche Erweiterung der Mikropro-grammierung, Mikrobefehle als ISA



Meilenstein: Mikroprogrammierung nach Wilkes

Vorschlag zur Verwendung der Mikroprogrammierung

- M.V. Wilkes: The Best Way to Design an Automated Calculating Machine, Manchester University Computer Inaugural Conf., 1951, pp. 16-18.
- M.V. Wilkes, J.B. Stringer: Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer, Proc. of the Cambridge Philosophical Soc., v. 49, 1953, pp. 230-238; Nachdruck: D.P. Siewiorek, C.G. Bell, A. Newell. Computer Structures: Principles and Examples. New York: McGraw-Hill, 1982.



Aus Bell/Newell

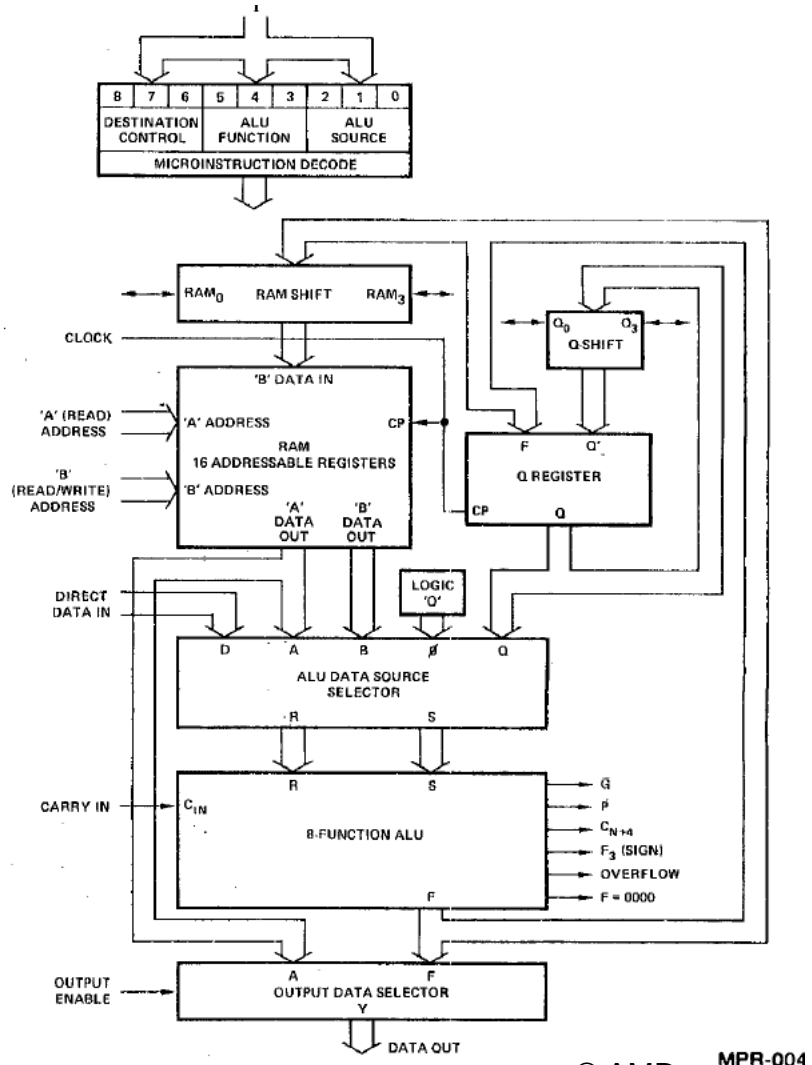
Meilenstein: Mikroprogrammierung in Großrechnern

- 1960/1970: Einsatz der Mikroprogrammierung zur Realisierung von Befehlssätzen, u.a. in Großrechnern
- Teilweise mit beschreibbarem Mikroprogrammspeicher

Meilenstein: Mikroprogrammierung mit *Bitslice*-Prozessoren

AMD 2900-Serie

- AMD 2901: 4-Bit-Rechen-chips, entsprechend Daten-wortbreite zu kaskadieren
- AMD 2910 Programm-ablaufchip
- Speicherchips
- Basis für eine Reihe von kleinen Rechnern



© AMD MPR-004

Meilenstein: Mikroprogrammierung mit MIMOLA

Zimmermann/ Marwedel: Automatisierte Erzeugung von mikroprogram-mierten Strukturen aus Algorithmen in PASCAL-artiger Notation in MIMOLA (=machine independent microprogramming language), ~1975-1987

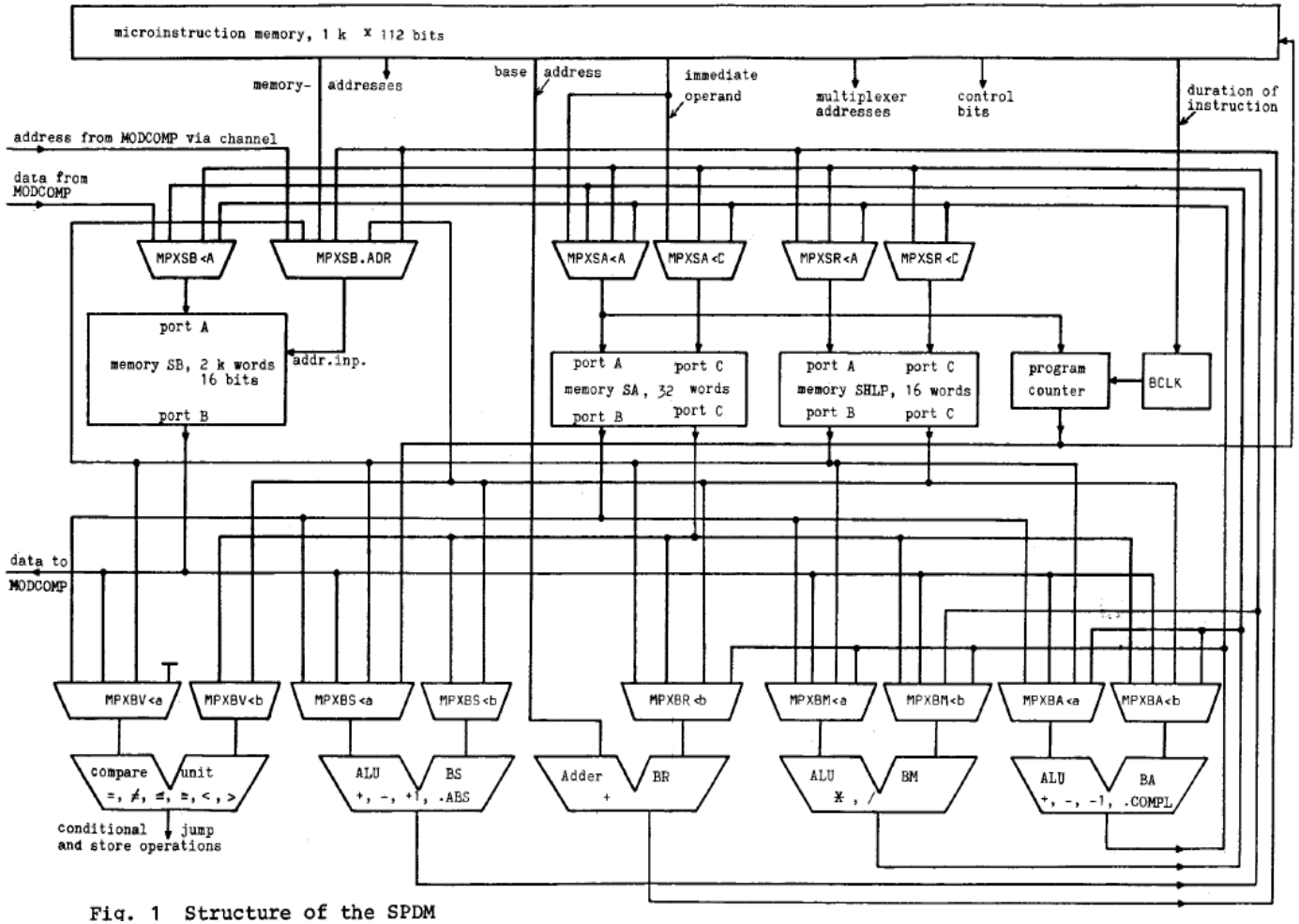


Fig. 1 Structure of the SPDM

Peter Marwedel: The Design of a Subprocessor with Dynamic Microprogramming with MIMOLA, Informatik-Fachberichte, 1980, Vol. 27, S. 164-177

Parallelisierbarkeit bei Datenabhängigkeiten?

- Operationen häufig abhängig von vorherigen Operationen
- Diese können nicht parallel ausgeführt werden!
- I.d.R. nur bestimmte Operations-Typen parallel möglich

Beispiel: Betrachten (auf Hochsprachenebene)

```
x = (a + b) * (a - c)
```

Sequentiell

```
x0 = (a + b)
x1 = (a - c)
x = x0 + x1
```

Parallele Realisierung

[ALU Nr. 1]

```
x0 = (a + b)
x = x0 + x1
```

[ALU Nr. 2]

```
x1 = (a - c)
```

Parallelisierbarkeit von Schleifen

Wie die potentielle Parallelisierbarkeit von Anweisungen in Schleifenrümpfen ausnützen?

☞ “Abrollen” des Schleifenrumpfs

Schleife:
```
for (i = 0; i < n; i++)
    x[i] += a;
```

Abgerollt (4-mal):
```
for (i = 0; i < n; i+=4) {
    x[i]   += a;
    x[i+1] += a;
    x[i+2] += a;
    x[i+3] += a; }
```

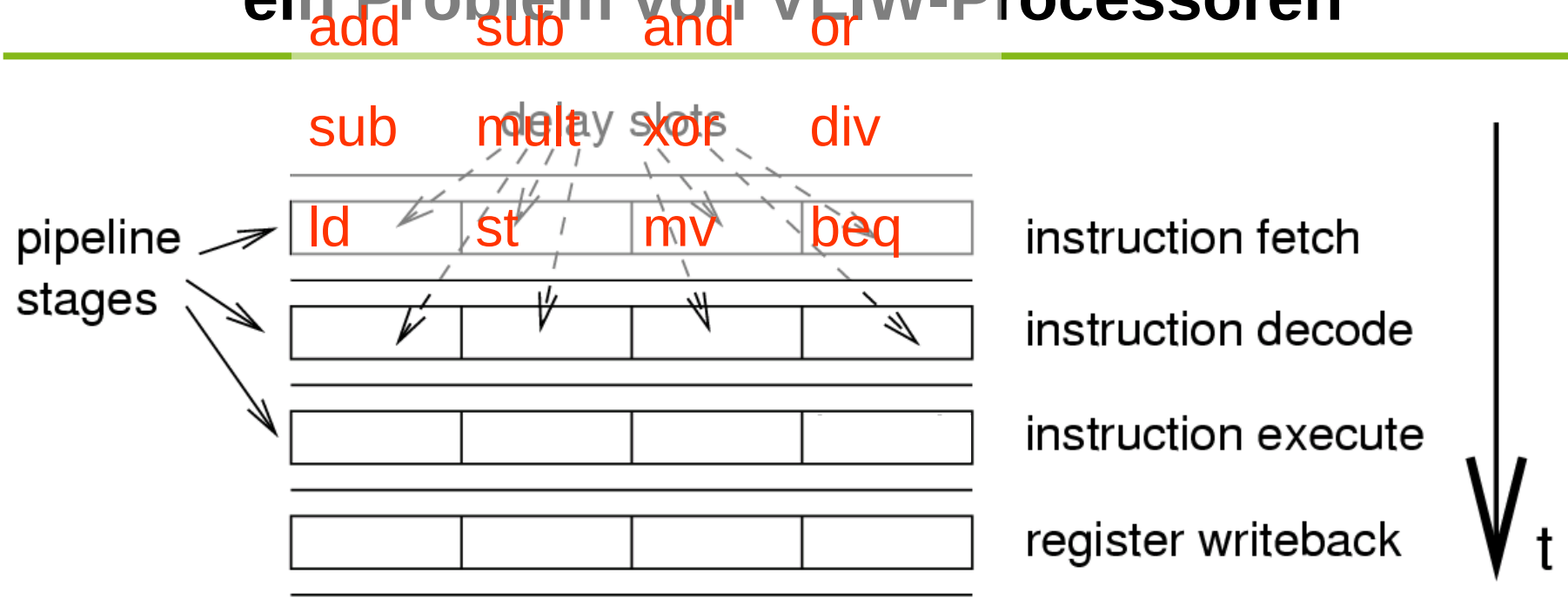
Im Allg. (falls n mod 4 != 0) zusätzlicher Code erforderlich

Parallelisierbarkeit von bedingten Anweisungen?

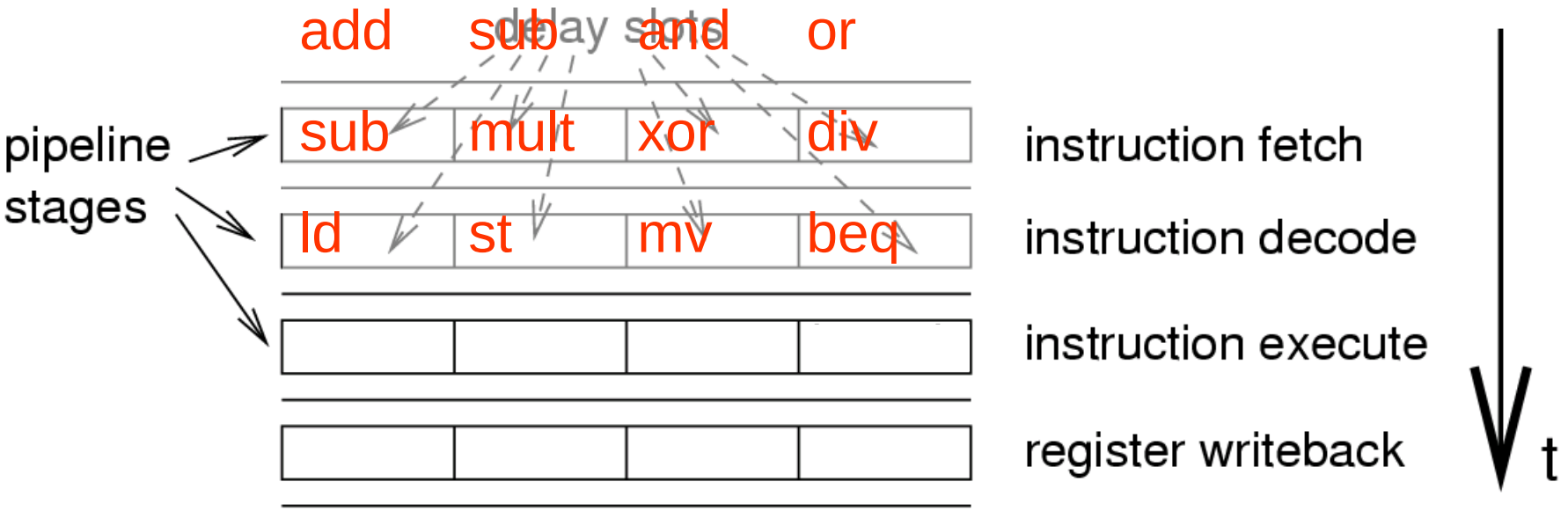
Kontrollfluss innerhalb von Programmen häufig nicht linear!

- Beispiel:
```
for (i = 0; i < n; i++)
    x[i] += a;
```
- Benutzung von bedingten Sprüngen?

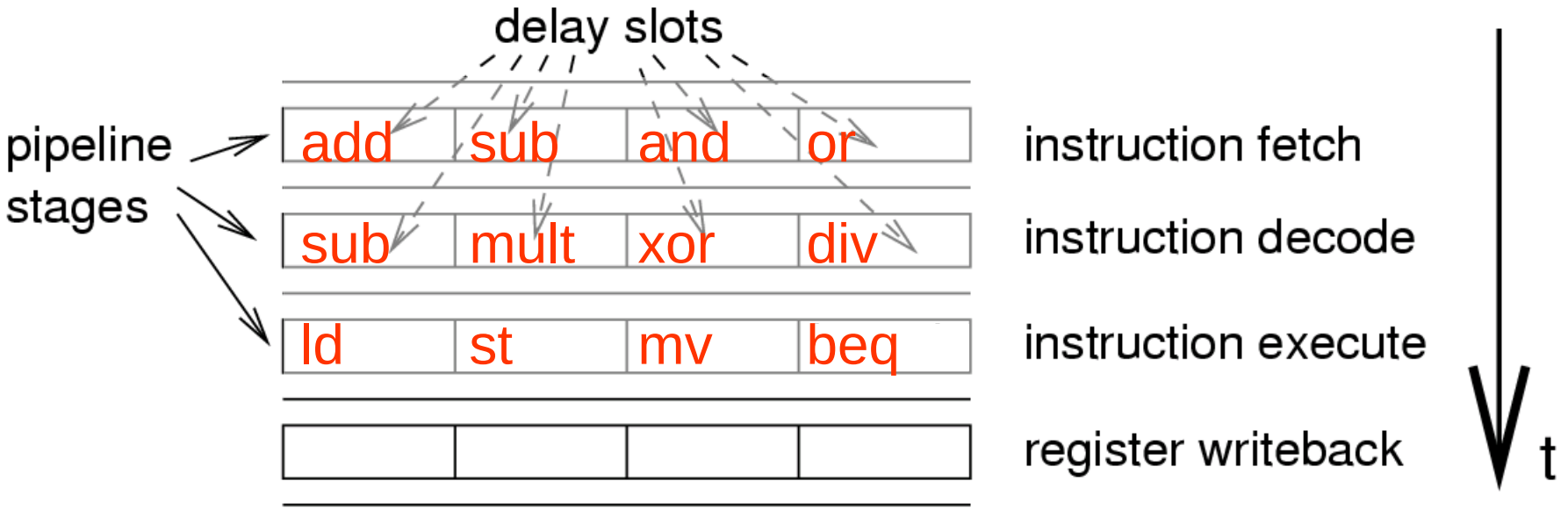
Große Zahl von *delay slots*, ein Problem von VLIW-Processoren



Große Zahl von *delay slots*, ein Problem von VLIW-Processoren



Große Zahl von *delay slots*, ein Problem von VLIW-Prozessoren



Die Ausführung von vielen Befehlen wurde gestartet bevor erkannt wurde, dass ein Sprung ausgeführt werden soll.
Diese Befehle für ungültig zu erklären würde Rechenleistung verschwenden.

- ☞ Ausführung der Befehle wird als *Feature* erklärt (statt als *bug*)
- ☞ Wie soll man all die *delay slots* füllen?
- ☞ Sprünge möglichst vermeiden!

Predicated execution: Sprungfreie Realisierung von *if-Statements*

Predicated execution “[c] I“ besteht aus:

- Bedingung c
- Befehl I

c = true ☞ I ausgeführt
c = false ☞ NOP

Predicated execution: Sprungfreie Realisierung von *if-Statements* (TI C6x)

```
if (c)
{ a = x + y;
  b = x + z;
}
else
{ a = x - y;
  b = x - z;
}
```

Conditional branch

```
      [c] B L1
          NOP 5
          B L2
          NOP 4
          SUB x,y,a
      ||  SUB x,z,b
L1:       ADD x,y,a
      ||  ADD x,z,b
L2:
```

Max. 12 Zyklen

Predicated execution

```
      [c]  ADD x,y,a
|| [c]  ADD x,z,b
|| [!c] SUB  x,y,a
|| [!c] SUB  x,z,b
```

1 Zyklus

VLIW-Beispiel: Transmeta Crusoe

Zielanwendung: mobile Geräte (z.B. Laptops)

☞ geringer Stromverbrauch

x86-Befehle werden vom Prozessor in VLIW-Befehle übersetzt

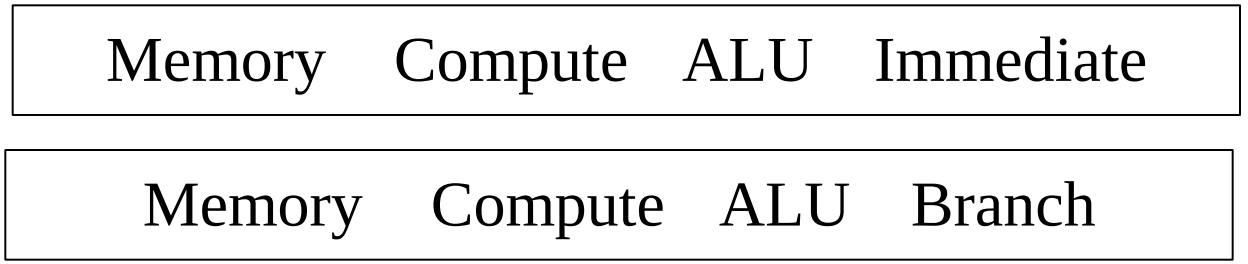
Befehlscodierung:

- 2 Befehlsgrößen (64 u. 128 bit) mit 2 (bis zu 4) Ops
- 5 Typen von Operationsfeldern:
  - ALU-Operationen
  - Compute
  - Memory:
  - Branch
  - Immediate: 32 bit Konstante für andere Op. im Befehl

VLIW-Beispiel: Transmeta Crusoe (2)

... Befehlscodierung:

- 2 Befehlsformate (128 bit Befehlsbreite)



Befehlsübersetzung:

Crusoe-Befehl (VLIW) kann mehreren x86 Op. entsprechen

- Interpretation pro x86 Instruktion
- Bei wiederholten Codeblöcken => Caching des Crusoe-Codes

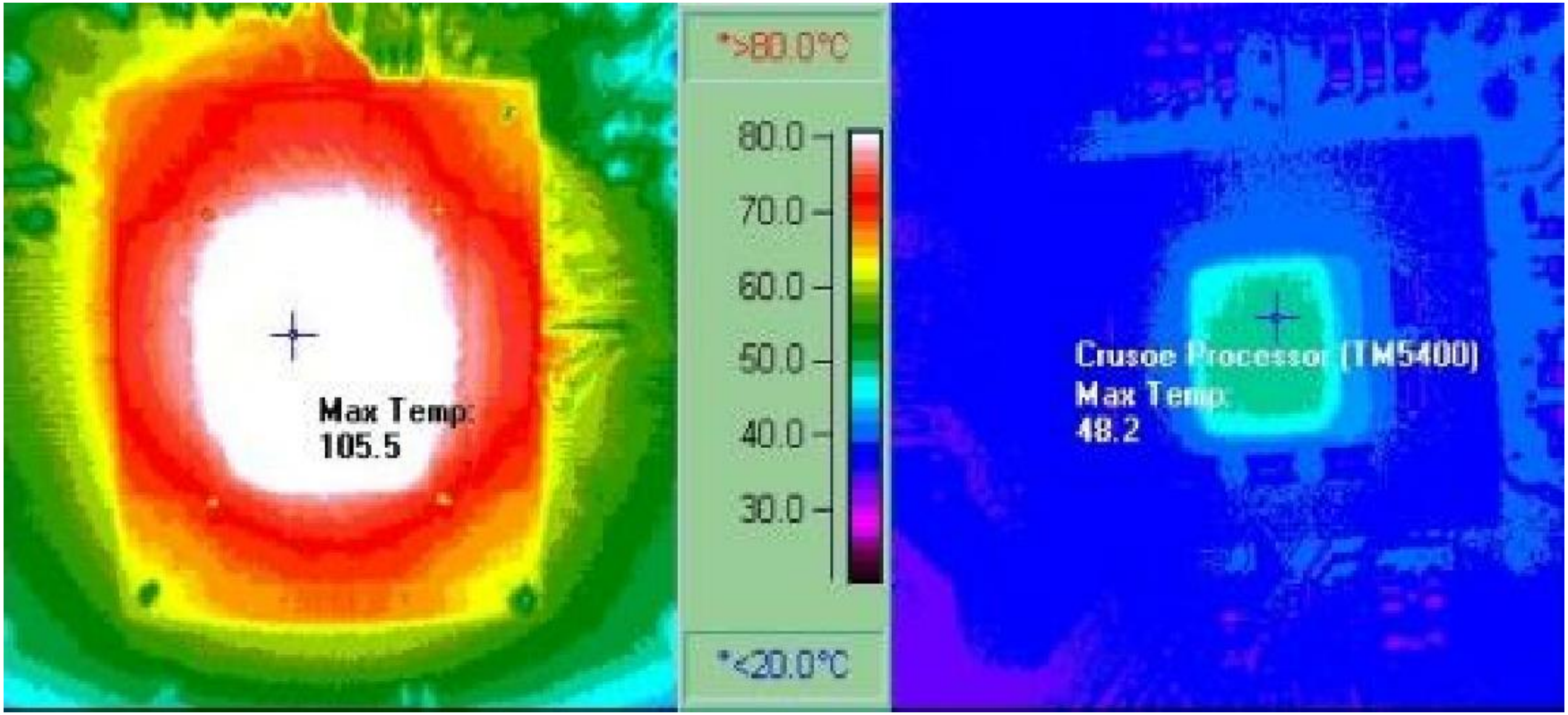
Problem: Ausnahmebehandlung

- Crusoe verwendet „Schattenregister”
- Primäre Register können bei Ausnahme restauriert werden

Temperaturvergleich

Pentium

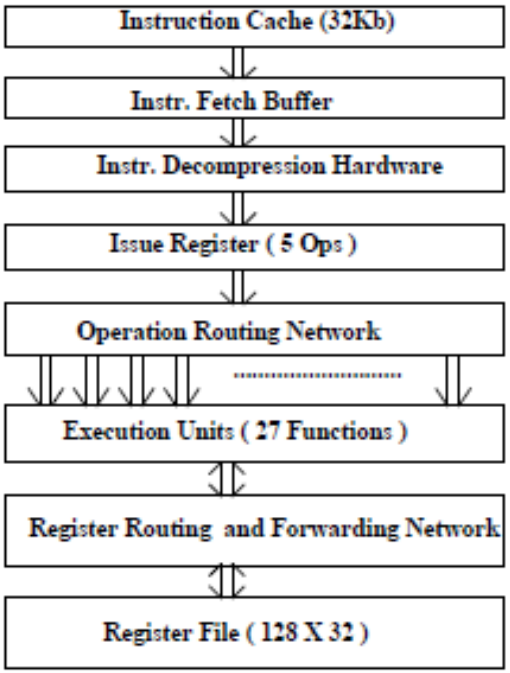
Crusoe



Bei Ausführung derselben Anwendung. Gemäß Publikation von Transmeta [www.transmeta.com] (~2003, nicht mehr verfügbar)

TriMedia

TM-1 VLIW engine



- 5 RISC operations/cycle
- 32 kByte Icache (compressed)
- dual port 16 kByte Dcache
- conditional (guarded) operations
  $R_g : R_{dest} = imul\ R_{src1}, R_{src2}$
- multimedia operation set

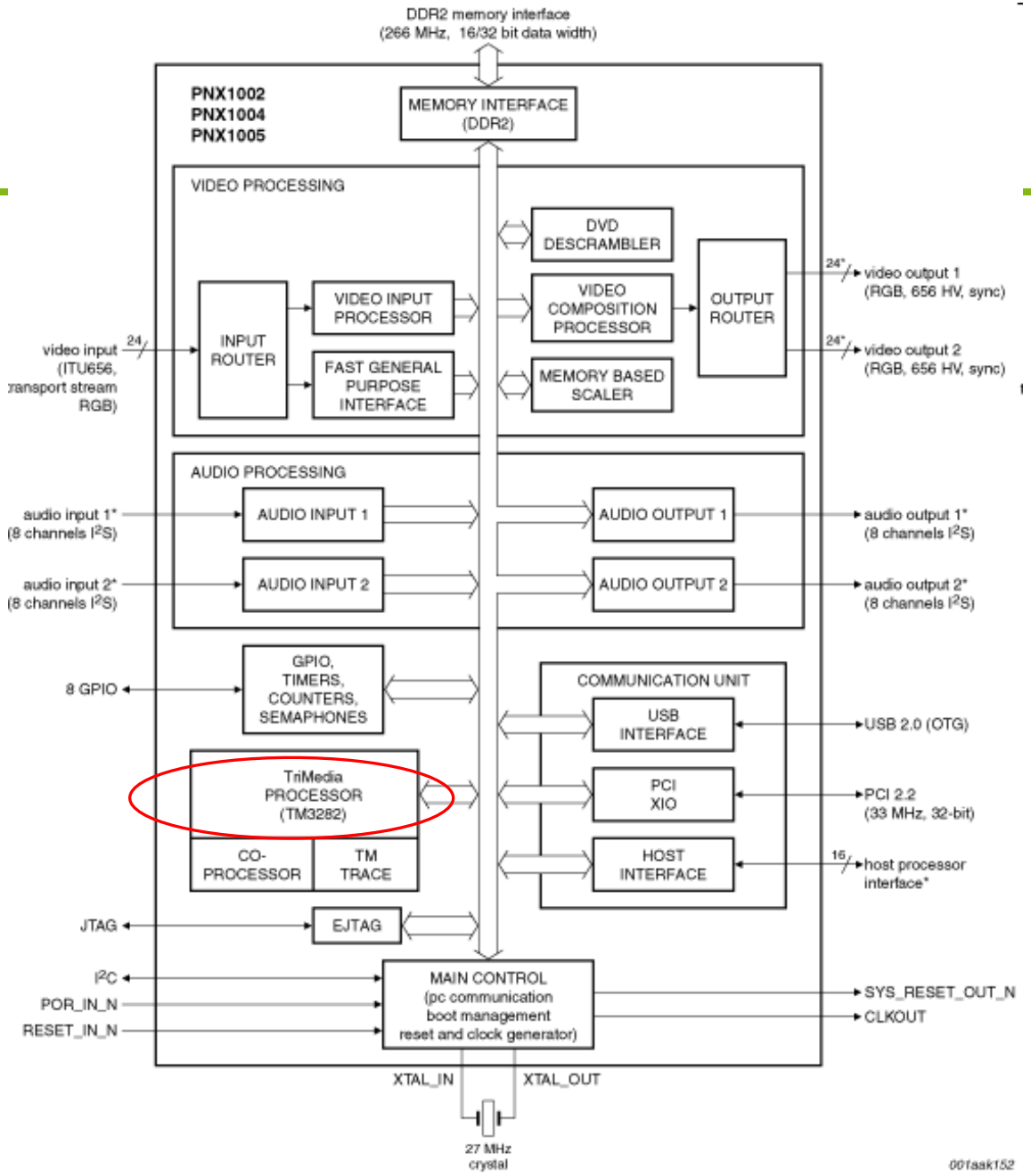
Func Type	Qty	Latency	Recovery
Const	5	1	1
ALU	5	1	1
Memory	2	3	1
Shift	2	1	1
DspAlu	2	2	1
DspMul	2	3	1
Branch	3	3	1
Falu	2	3	1
Ifmul	2	3	1
Fcomp	1	1	1
Fdiv/Fsqrt	1	17	16

2009: 45nm-Version, Bestandteil vieler Systems on a Chip (SoCs)

[http://www.hotchips.org/archives/hc8/3_Tue/HC8.S6/HC8.6.1.pdf]
[http://ce.et.tudelft.nl/publicationfiles/1228_587_thesis_JAN_WILLEM.pdf]
[http://en.wikipedia.org/wiki/TriMedia_%28Mediaprocessor%29]

TriMedia-Processor

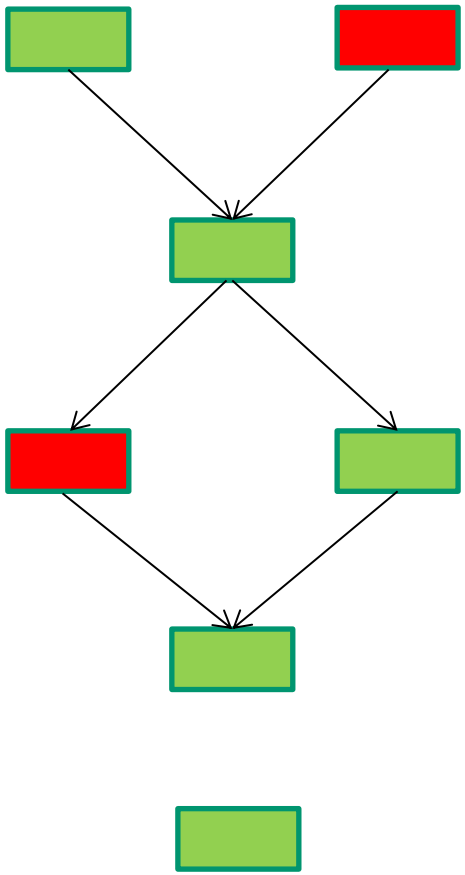
Heute 5-8 Operationen pro Zyklus startbar



http://www.tridentmicro.com/producttree/stb/media-processor/pnx100x/

Meilenstein(e): VLIW nach Fisher

- Joseph A. Fisher: **Trace Scheduling**: A Technique for Global Microcode Compaction, IEEE Trans. Computers, 1981, S. 478ff; Überwindet Grenzen des Schedulings für einzelne Basisblöcke, indem es ganze (häufig ausgeführte) Ausführungspfade einplant und ggf. Kompensationscode einführt.
- Joseph A. Fisher, Paolo Faraboschi, Cliff Young: *Embedded Computing - A VLIW Approach to Architecture, Compilers and Tools*, Morgan-Kaufmann, 2004: Architekturen, mit tlw. wenig kompakter Kodierung, wenig embedded

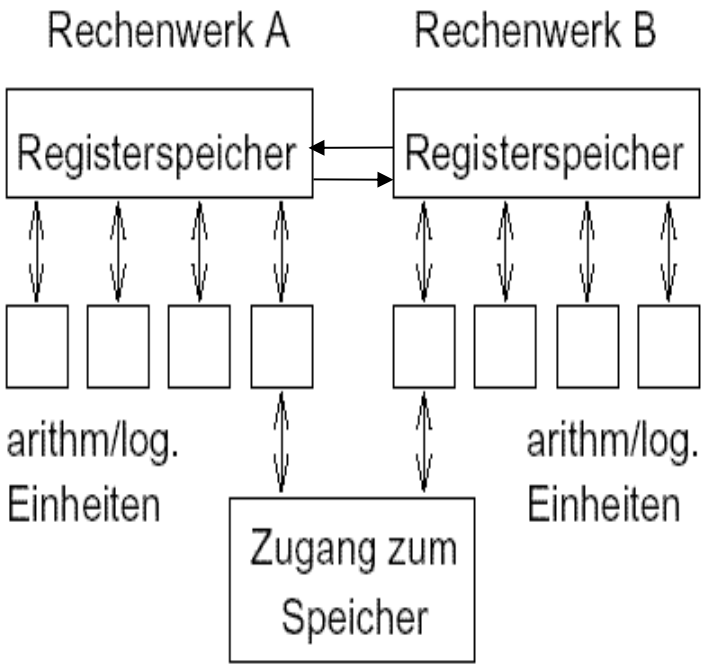


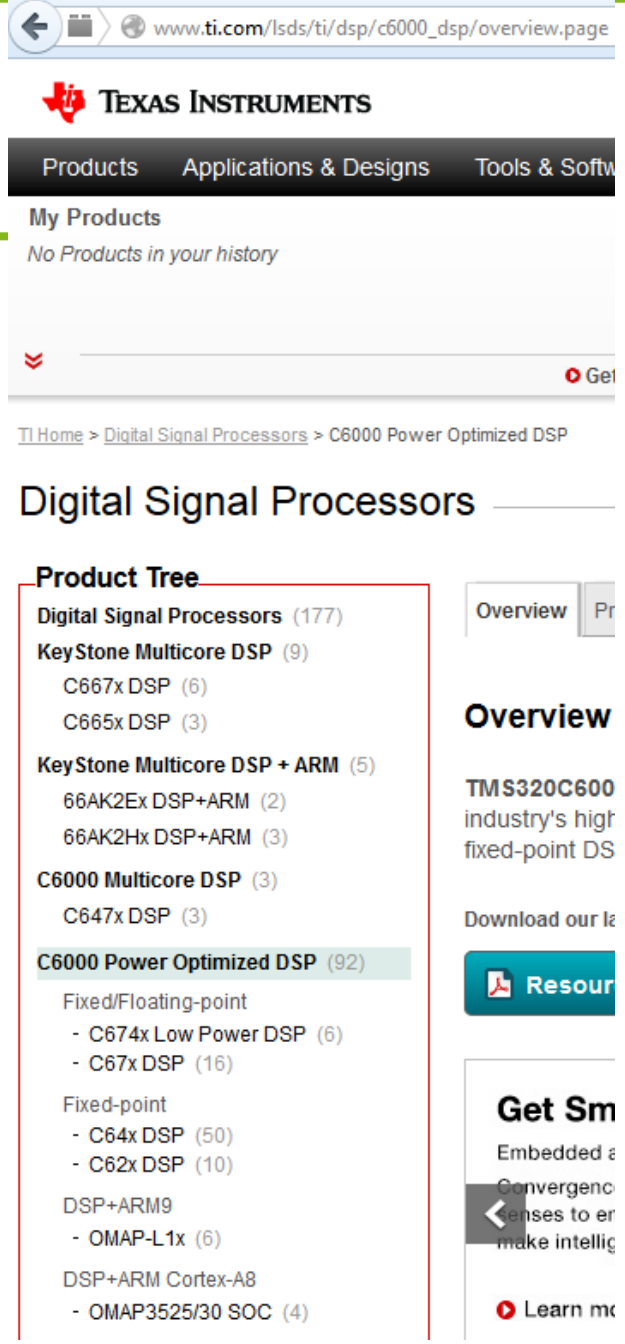
Häufiger Pfad = Kandidaten für VLSI-Befehle

VLIW/EPIC-Beispiel: TMS320C6205 Prozessor

Viele Speicherports sind teuer:

- ☞ Zwei Rechenwerke
- Begrenzte Anzahl von Verbindungen





VLIW/EPIC-Beispiel: TMS320C6205 Prozessor (2)

- Bei 200 MHz bis zu 8 32-Bit-Befehle pro Takt (5 ns) zu starten (max. 1,6 GIPS)
- Jeder 32-Bit-Befehl spricht funktionelle Einheit an; Compiler nimmt Zuordnung zur *Compile*-Zeit vor.
  1 Bit in jedem Befehl: Nächster Befehl noch im selben Takt?
  Max. 256 Befehlsbits (*instruction packet*) pro Zyklus.

Ausführung:

Schritt 1	Befehle A, B, C, D, E
Schritt 2	Befehl F

De facto variable Befehlslänge von 32 bis 256 Bit.

Sprünge „mitten in ein Befehls-paket" hinein.

Die TRIMARAN-Compiler-Infrastruktur

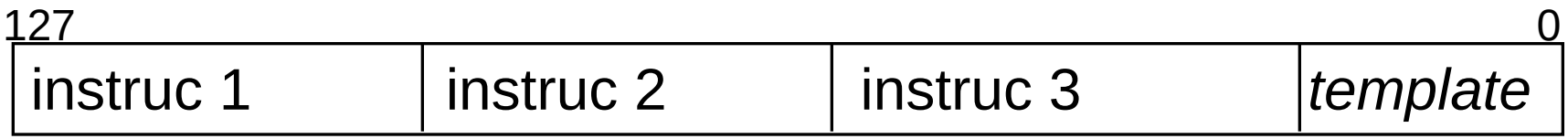
"*An Infrastructure for Research in Backend Compilation and Architecture Exploration*

Trimaran is an integrated compilation and performance monitoring infrastructure. The architecture space that Trimaran covers is characterized by HPL-PD, a parameterized processor architecture supporting novel features such as predication, control and data speculation and compiler controlled management of the memory hierarchy. Trimaran also consists of a full suite of analysis and optimization modules, … Trimaran is intended for researchers and educators interested in the following topics: ….."

[www.trimaran.org]

Mehr Kodierungsfreiheit mit dem IA-64 Befehlssatz

3 Instruktionen pro ***bundle*:**



Es gibt 5 Befehlstypen:

- A: allgemeine ALU Befehle
- I: speziellere integer Befehle (z.B. *shifts*)
- M: Speicher-Befehle
- F: Gleitkomma-Befehle
- B: Sprünge

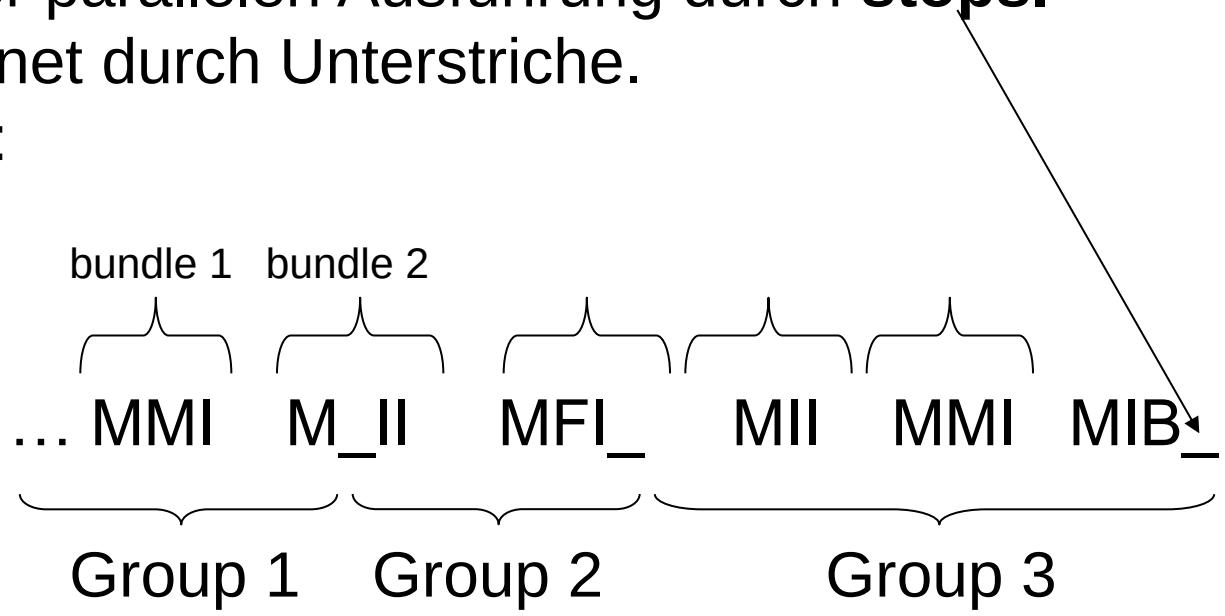
Instruction grouping information

Die folgenden Kombinationen können in *Templates* kodiert werden:

- MII, MMI, MFI, MIB, MMB, MFB, MMF, MBB, BBB, MLX mit LX = *move 64-bit immediate* kodiert in 2 *slots*

Stops

Ende der parallelen Ausführung durch **stops.**
Bezeichnet durch Unterstriche.
Beispiel:



Sehr beschränkte Platzierung von Unterstrichen in einem Bündle.
Parallele Ausführung kann sich über mehrere Bündel erstrecken.

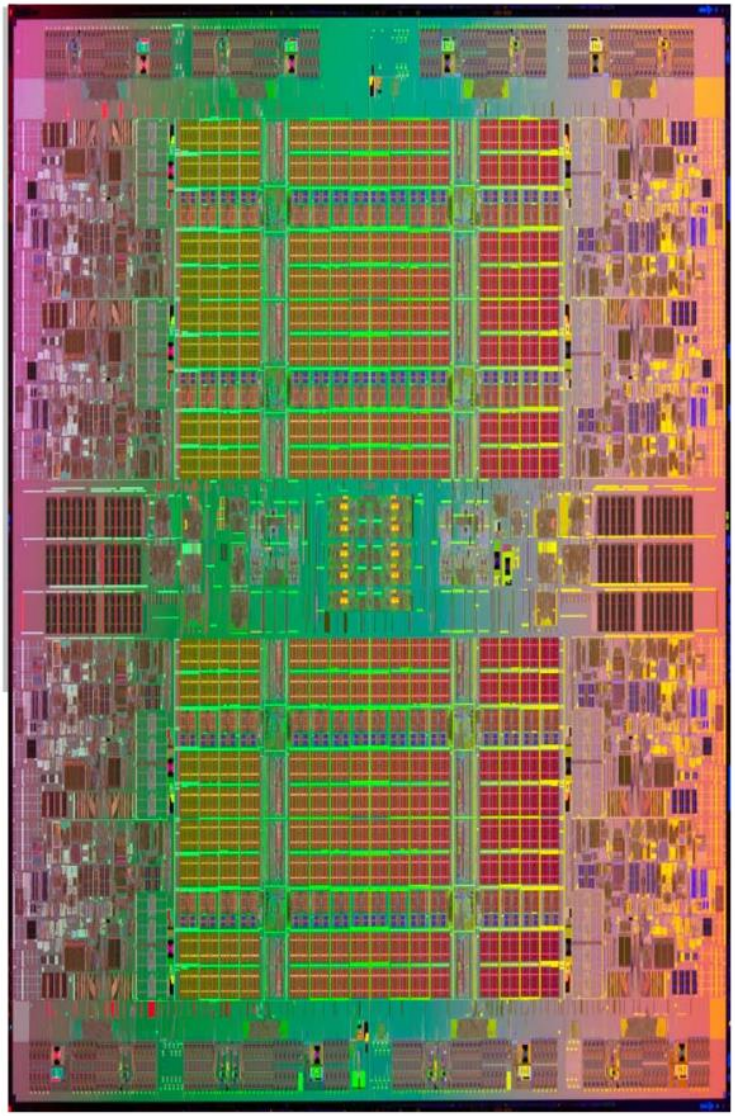
Intel IA-64 Vertreter: Itanium

- Implementierung der IA-64 Architektur
- Prozessorkern erlaubt bis zu 6 Befehlsausführungen pro Taktzyklus (inkl. bis zu 3x *Branch*, 2x Speichertransfer)
- Speicherhierarchie mit 3 Cache-Ebenen

  (Daten- und Instruktions-Cache getrennt auf 1. Ebene)
- 9 funktionale Einheiten:
  2x I-*unit* (ALU+*shift*), 2x M-*unit* (ALU+Load/Store),
  3x B-*unit* (*Branches* etc.), 2x F-*unit* (Gleitkomma)
- Dekodierungs-“Fenster” umfasst 2 *bundles*
- Allozierung von Instruktionen auf funktionale Einheiten
  - Falls erforderliche Einheit bereits belegt ☞ Bündel teilen
  - ∃ spezielle Restriktionen (z.B. M-M-F bundles nur isoliert)

Itanium® 9500

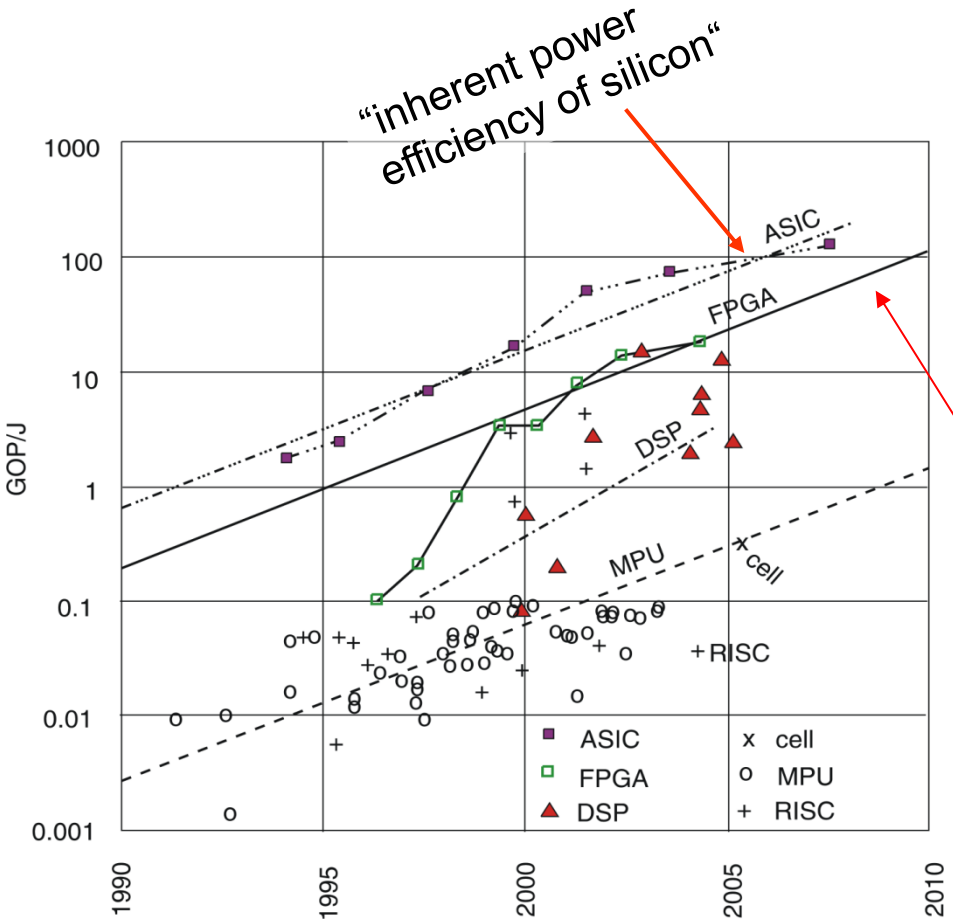
- 3.1 G Transistoren
- 8 Kerne
- 2-fach Hyper-threading/Kern
- Bis zu 2 TB Speicher
- Bis zu 2.53 Ghz bei 170 W
- Im Desktop-Bereich nicht (mehr) vorhanden, zielt derzeit v.a. auf den Server-Bereich

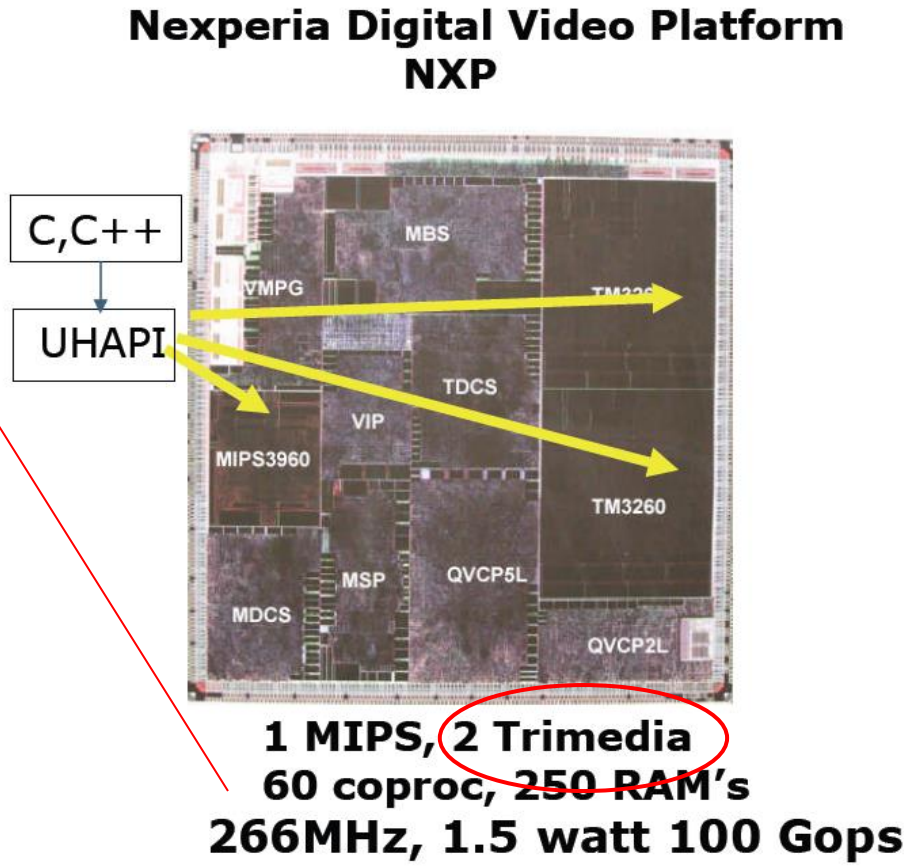


Intel Itanium processor 9500 series die.
8 cores, 32MB Last Level Cache, and over 3.1 Billion transtors.

http://www.intel.com/content/www/us/en/processors/itanium/itanium-9500-brief.html

Wie energieeffizient können solche Prozessoren werden?

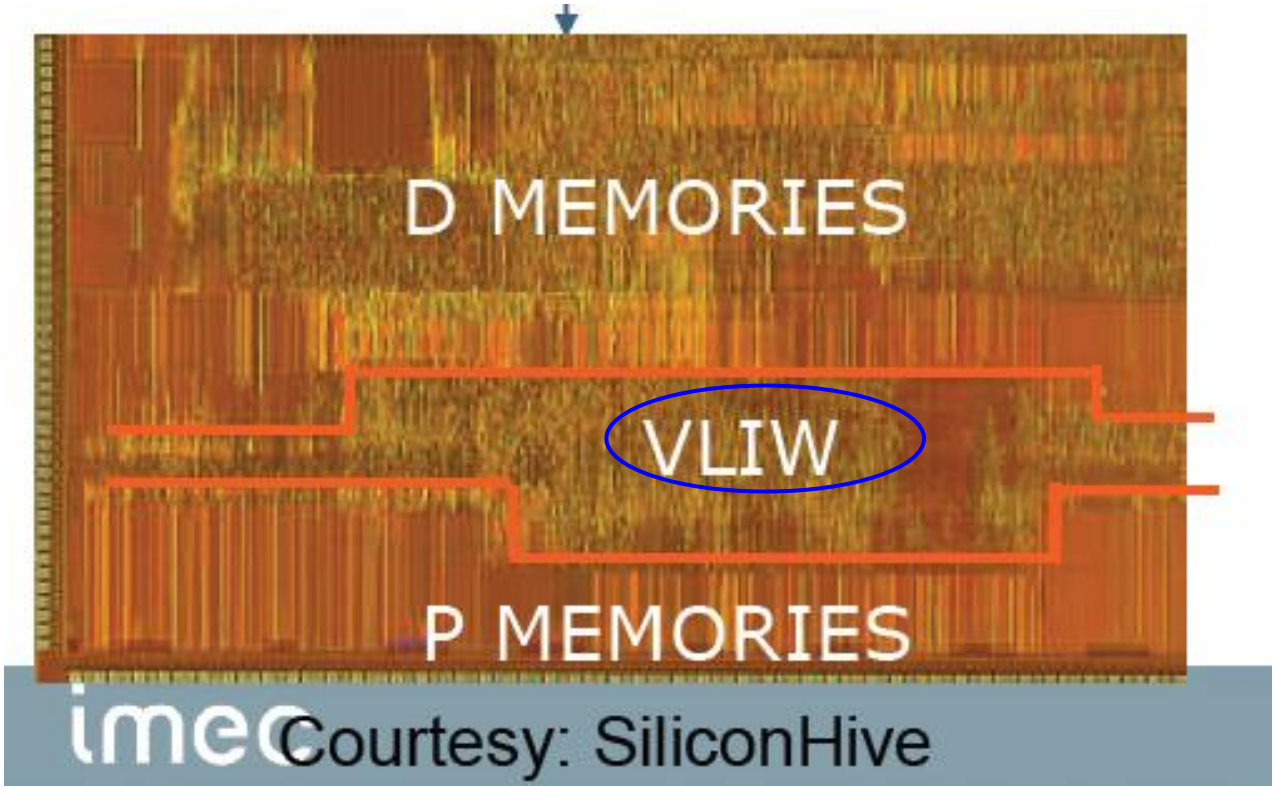




~1/2 inherent power effiency of silicon

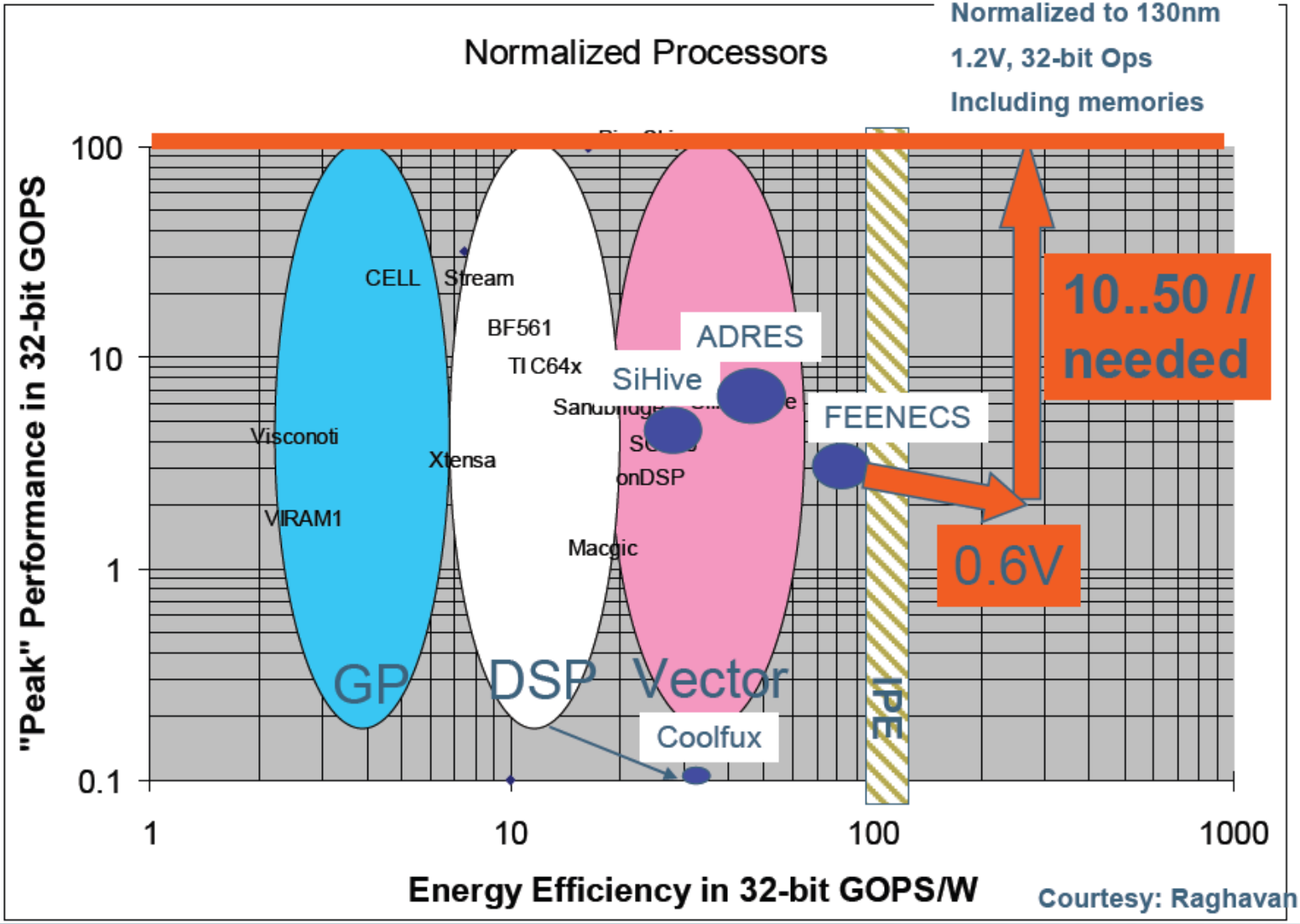
© Hugo De Man: From the Heaven of Software to the Hell of Nanoscale Physics: An Industry in Transition, *Keynote Slides*, ACACES, 2007

41 *Issue VLIW for SDR*



130 nm, 1,2 V; 6.5 mm²; 16 bit; 150 MHz; 30 operations/cycle (OFDM); 190 mW (incl. SRAMs); 24 GOPs/W;
~1/5 inherent power effiency of silicon

"Estimated" State of the Art Processors



Neue Entwürfe? Ja, z.B. Kalray

Core architecture

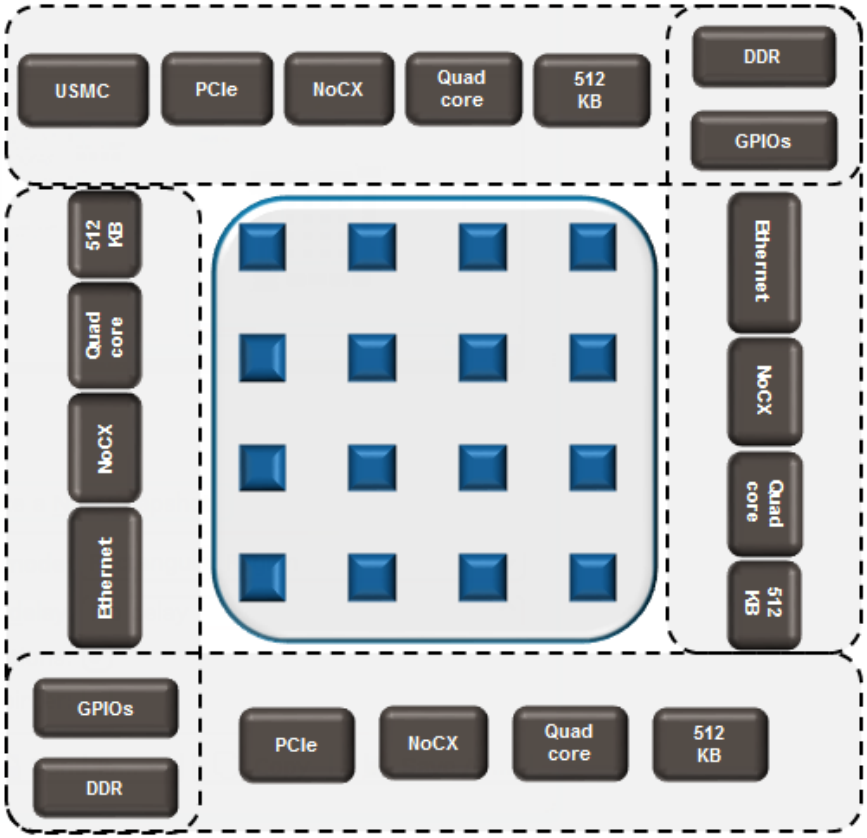
MPPA® core: **32-bit VLIW processor**:

- One Branch/Control Unit
- 2 ALUs
- One Load/Store Unit incl. simplified ALU
- 1 MAC / FPU incl. simplified ALU
- Standard IEEE 754-2008 FPU+extens.
- 1 MMU
- **≤ 5 x 32bit RISC like integer ops/cycle**

Compute Cluster

- 16 cores, private FPU, MMU
- DVFS + DPM support
- 1 system core, private FPU, MMU
- 1 instruction/data L1-cache per core
- 1 DMA, 1 shared memory

16 Compute Clusters



2013: 256 cores, **75 GOPs/W**

© www.kalray.eu

VLIW- und EPIC-Befehlssätze

Befehlssätze, die eine parallele Ausführung unterstützen:

1. **VLIW = *very long instruction word machine***
   Bildung eines Befehlspaketes konstanter Länge;
   alle Befehle im Paket sind parallel auszuführen;
   Compiler bestimmt, was parallel auszuführen ist.
   ☞ Sehr lange Befehlswörter (64, 128 Bit oder mehr)
2. **EPIC = *Explicitly parallel instruction computing***
   Mögliche parallele Ausführungen werden ebenfalls zur Compilezeit erkannt und im Code kodiert;
   Parallele Ausführung kann aber flexibler ausgedrückt werden, nicht notwendigerweise durch Bildung eines Befehlspaketes.

VLIW: Diskussion (1)

- **Vergrößerung der Codegröße**
  - Gründe
    - Wegen des Abrollens von Schleifen
      (um mehr Parallelisierungspotential zu schaffen)
    - Unbenutzte funktionale Einheiten
      ☞ unbenutzte Teile des VLIW-Befehlsworts (NOPs)
  - Mögliche Gegenmaßnahmen:
    - Nur ein großes Feld für Direktoperanden (für beliebige Operation im Befehlswort) ☞ z.B. Crusoe
    - Komprimierung des Binärcodes
    - Kein festes VLIW, sondern Kodierung parallel auszuführender Befehle mit variabler Codelänge ☞ EPIC / IA-64

VLIW: Diskussion (2)

- **Binärkompatibilität:**
  Generierung der Befehlscodierung macht expliziten Gebrauch von Wissen über **interne Architektur** des Prozessors (insbes. Anzahl funktionaler Einheiten, aber auch zum *Pipelining*)
  ☞ Code ggf. **nicht** auf veränderter interner Architektur lauffähig
  Hier: Parallelisierung durch Compiler (vs. Hardware)
  Widerspricht eigentlich der Idee einer externen Architektur (Befehlssatz) als Abstraktion von Realisierung und Schnittstelle zum Programmierer

VLIW: Diskussion (3)

- Erzeugung ausreichender Parallelität auf Instruktionsebene (ILP = *instruction level parallelism*)
  - Abrollen von Schleifen
  - *Trace Scheduling*
  - *Predicated execution*

Parallele Funktionseinheiten können ggf. trotzdem ausreichend genutzt werden.

Problem jeder Parallelverarbeitung, auch wenn Befehle auf Funktionseinheiten [dynamisch] verteilt werden ☞ Kap. 3

EPIC-Befehlssätze

EPIC = *Explicitly Parallel Instruction Computing*

Oberklasse der VLIW-Maschinen

- Explizite Angabe der Parallelität
- Aber: Keine 1:1 Beziehung zwischen Worten im Binärcode und paralleler Ausführung
- Auch: Mikroarchitektur übernimmt Aufgaben bei der Steuerung der Parallelverarbeitung
  (bei “reinem” VLIW: Kein Eingreifen der Hardware nötig)

Beispielarchitektur: Intel IA-64

Zusammenfassung

- Gründe für die Abkehr vom Prinzip immer schnellerer superskalarer Prozessoren
  - Fehlende Energieeffizienz
  - Zunehmende Komplexität der Hardware
- VLIW/EPIC
  - VLIW: Festes Befehlsformat: nicht kompakt, Binärkompatibilität?
  - EPIC: Größere Flexibilität der Kodierung, Hardware darf in begrenztem Umfang über Parallelität entscheiden
  - Zurückhaltende kommerzielle Nutzung, dennoch in einigen Anwendungen im Einsatz

☞SIMD
☞Multi-cores

} Später

Quellen

Zu den Wurzeln:

- Joseph A. Fisher, Paolo Faraboschi, Cliff Young: *Embedded Computing - A VLIW Approach to Architecture, Compilers and Tools*, Morgan-Kaufmann, 2004 (kein Fokus auf Eingebettete Systeme}
- Peter Marwedel: The Design of a Subprocessor with Dynamic Microprogramming with MIMOLA, Informatik-Fachberichte, 1980, Vol. 27, S. 164-177 (auf der Basis der Ideen von G.Zimmermann)

Beispielhafte jüngere Quellen

- Kyo, S.; Okazaki, S.; Koga, T.; Hidano, F., A 100 GOPS in-vehicle vision processor for pre-crash safety systems based on a ring connected 128 4-Way VLIW processing elements," *VLSI Circuits, 2008 IEEE Symposium on* , vol., no., pp.28,29, 18-20 June 2008
- J. L. Ayala, M. Lopez-Vallejo, D. Atienza, P. Raghavan, F. Catthoor, D. Verkest: Energy-aware compilation and hardware design for VLIW embedded systems, Intern. Journ. of Embedded Systems, 1-2/2007