technische universität
dortmund

fakultät für informatik
informatik 12

Non-Von Neumann-Maschinen

Peter Marwedel
Informatik 12
TU Dortmund

2012/04/19

2.7 ASIPs

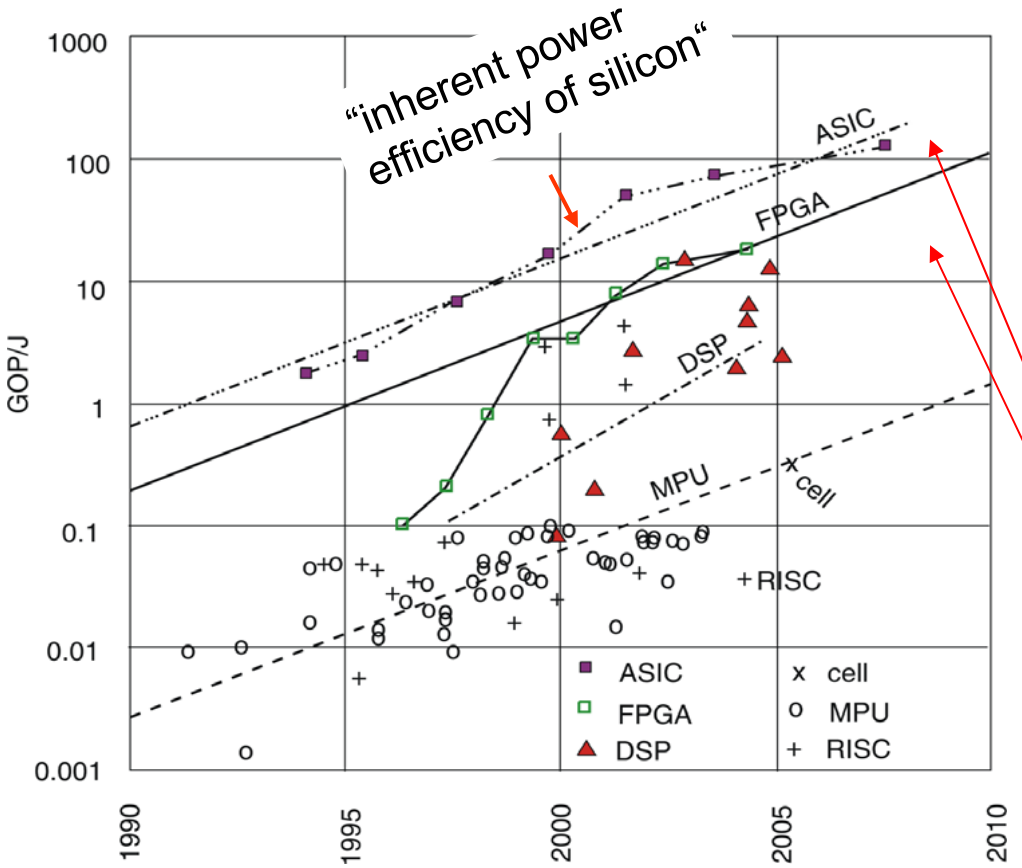
Application-specific instruction set processor (ASIP):
Befehlssatz aufgrund der Anwendung festgelegt.

Einsatz:
U.a. für effiziente eingebettete Prozessoren

Frühe Beispiele:

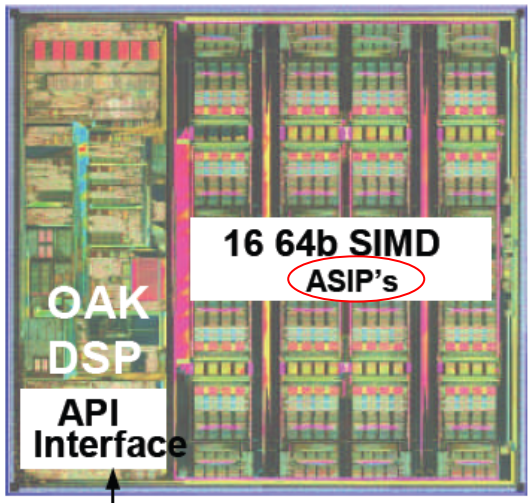
- Yasuura et al. (Fukuoka, Japan): Generische Architektur, u.a. werden die Datenwortbreiten anwendungsspezifisch festgelegt.
- I.-J. Huang: Weglassen unbenötigter Befehle beim 6509.

Existenznachweis der Energieeffizienz



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

VIP for car mirrors
Infineon



Hd Compiler ← VPL C

200MHz , 0.76 Watt
100Gops @ 8b
25Gops @ 32b

Close to power efficiency of silicon

Literatur

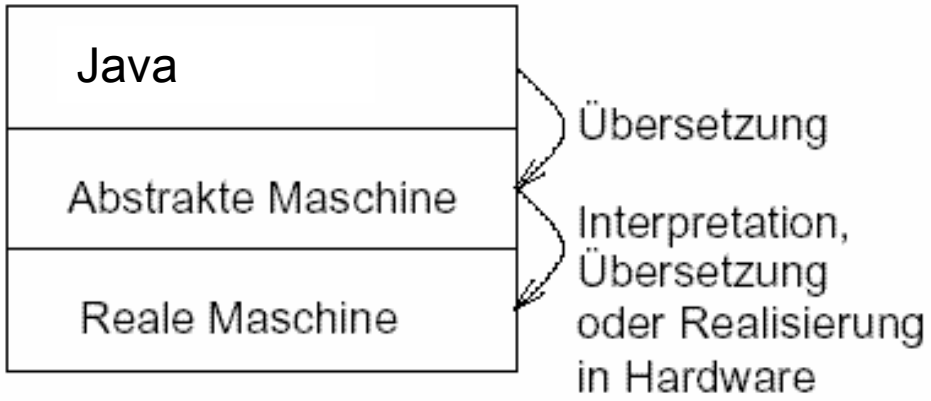
- M. K. Jain, M. Balakrishnan, Anshul Kumar: *ASIP Design Methodologies* : Survey and Issues, VLSI Design, 2001
- M. Gries, K. Keutzer (Hrg.): *Building ASIPs: The Mescal Methodology*, Springer, 2005
- P. Ienne, R. Leupers (Hrg.): *Customizable Embedded Processors,* Morgan Kaufmann, 2006
- O. Schliebusch, H. Meyr, R. Leupers: *Optimized ASIP Synthesis from Architecture Description Language Models*, Springer, 2007
- T. Shiro, M. Abe, K. Sakanushi, Y. Takeuchi, M. Imai: *A Processor Generation Method from Instruction Behavior Description Based on Specification of Pipeline Stages and Functional Units*, ASP-DAC, 2007
- C. Wolinski, K. Kuchcinski: *Automatic selection of application-specific reconfigurable processor extensions*, DATE, 2008
- Laura Pozzi, Kubilay Atasu, Paolo Ienne: *Exact and approximate algorithms for the extension of embedded processor instruction sets*. IEEE Transactions on CAD, 2006.
- Tensilica Inc.: The xpres compiler: Triple-threat solution to code performance challenges. Tensilica Inc Whitepaper, 2005.

2.8 Abstrakte Maschinen

Maschinen, die jeweils einen abstrakten Befehlssatz interpretieren.

Programme werden in Befehle der abstrakten Befehlssätze übersetzt, nicht direkt in Maschinenbefehle realer Maschinen.

Beispiele: abstrakte Befehlssätze zur Realisierung von Java, (UCSD-) Pascal, PROLOG, LISP, FORTH, Smalltalk usw..



Lediglich der Interpreter der abstrakten Befehlssätze muss für verschiedene Maschinen jeweils neu erzeugt werden.
Nachteil: niedrigere Ausführungsgeschwindigkeit.

Java Virtual Machine

Java:

- Objektorientierte Programmiersprache
- Datentypen: *byte, short, int, long, float, double, char*
- Unterstützt Netzwerk-Programmierung
- Im Hinblick auf Sicherheit entworfen:
  - keine Pointer-Manipulation wie in C, C++.
  - beschränkte Möglichkeit, Informationen über die momentane Umgebung zu erfahren
- Automatische Freispeicherverwaltung
- *Multithreading*
- Über Netze auf alle relevanten Maschinen zu laden.
- Java-Programme können dort ausgeführt werden, wo die JVM realisiert ist.

Eigenschaften der JVM

- Die JVM ist eine Kellermaschine,
- Operationscodes in einem Byte kodiert,
- Typische Befehle:
  - lade integer auf den Keller,
  - addiere die obersten Kellerelemente,
  - starte nebenläufige Ausführung,
  - synchronisiere nebenläufige Ausführung,
  - Befehle zur Realisierung der Objektorientierung
- Byte-Code ist kompakt (≈ 1/3 der Größe von RISC-Code). Interessant wenn Code zusammen mit den Prozessor auf einem Chip gespeichert werden muss.
- JVM verzichtet weitgehend auf *Alignment*-Beschränkungen

3 Methoden der Realisierung von JVMs:

1. Durch **Interpretation** von JVM-Befehlen in Software,

2. Durch Übersetzung in den Maschinencode der aufrufenden Maschine unmittelbar vor der Ausführung; *just-in-time compilation*.

3. Durch Realisierung einer JVM als „echte” Maschine.

„Echte“ Java-Maschinen

- PicoJava (I) (Sun, 1997): Spezifiziert, aber nie realisiert
  - Oberste Kellerelemente im Prozessor in schnellem Speicher
  - Häufigste Befehle in Hardware ausgeführt, CPI=1.
  - Komplexere Befehle: Mikroprogramm.
  - Ganz komplexe Befehle: Interrupt + Software
- PicoJava II (Sun, 1999): Spec frei verfügbar
- AJile JEMCore: für Realzeit-Anwendungen
- CJip-Prozessor: 16-Bit CISC-Architektur
- Jazelle-Erweiterung des ARM-Prozessors
- Versionen in FPGAs

[W. Puffitsch, M. Schoeberl: picoJava II in an FPGA, *JTRES '07 Proceedings of the 5th International workshop on Java technologies for real-time and embedded systems*, 2007]

Nicht-Von-Neumann-Maschinen

Wir geben die sequentielle Ausführung von Programmcode auf ….

2.9 Funktionale Programmierung und Reduktionsmaschinen

- Maschinen unterstützen die Auswertung funktionaler Programme direkt.
- Reduktionsmaschinen akzeptieren Ausdrücke einer funktionalen Sprache.
- Auswertung mittels **Baumtransformationen**, bis ein **Wert** erhalten wird.
- Die Bedeutung eines Programms = abgelieferter Wert.
- Einfachere Verifikation
- **Vereinfachte Parallelverarbeitung**

Auswertungsstrategien

1. ***string reduction***

   Jede Operation, die auf eine bestimmte Definition (auf einen bestimmten Teilbaum) zugreift, erhält und bearbeitet eine Kopie des Teilbaums. ➔erleichterte Realisierung, schnelle Bearbeitung skalarer Werte, ineffiziente Behandlung gemeinsamer Teilausdrücke.

2. ***graph reduction***

   Jede Operation, die auf eine bestimmte Definition zugreift, arbeitet über Verweise auf der Original-Definition.
   ➔ schwieriger zu realisieren, falls parallel bearbeitet wird; effizient auch für strukturierte Werte und gemeinsame Teilausdrücke; komplizierte Haldenverwaltung (*garbage collection*).

Steuerung der Berechnungs-Reihenfolge

1. **Die „*outermost*”- oder „*demand driven*”-Strategie:**

   Operationen werden selektiert, wenn der Wert, den sie produzieren, von einer bereits selektierten Operation benötigt wird.

   Erlaubt *lazy evaluation*.

   Erlaubt konzeptuell unendliche Listen, solange nur endl. Teilmengen referenziert werden (vgl. Streams).

2. **Die „*innermost*”- oder „*data-driven*” - Strategie:**

   Operationen werden selektiert, wenn alle Argumente verfügbar sind.

Vorteile

- Programmierung auf höherer Ebene
- kompakte Programme
- keine Seiteneffekte
- kein *Aliasing*, keine unerwartete Speichermodifikation
- keine Unterscheidung zwischen *call-by-name*, *call-by-value* und *call-by-reference* notwendig
- einfachere Verifikation, da nur Funktionen benutzt werden
- das von-Neumann-Modell von Speicherzellen und Programmzählern ist überflüssig
- beliebige Berechnungsreihenfolge für Argumente (außer bei nicht-terminierenden Berechnungen)
- Debugging einfach: *Trace* des aktuellen Ausdrucks.
- Kommerziell bislang nicht erfolgreich

Ansätze zur Nutzung von funktionaler Programmierung bei Multi-Core-Prozessoren

- Verschiedene funktionale Sprachen
- **MapReduce** (Google): Definition zweier Funktionen:
  - *map*: (key,value) → (intermediate key,intermed. value)*
  - *reduce*: (intermed. key,value*) → (intermed. key,value)

  Beispiel:
  - *map*: (*buchtitel, inhalt*) → (*wort*, 1)*
  - *reduce* erhält nach *wort* sortierte Tupel, zählt Worte und liefert Häufigkeit für ein bestimmtes Wort

  map und *reduce* sind **Funktionen**
- **CLOJURE:** Lisp-Dialekt zur parallelen Programmierung auf der Basis der JVM

2.10 Maschinen zur Realisierung von Logischen Programmiersprachen (1)

Direkte Realisierung von logischen Programmiersprachen

Beispiel:

PROLOG-Maschinen

Klassen von PROLOG-Maschinen:

- Sequentielle Maschinen mit strenger Wahrung der PROLOG-Semantik
  Leiden meist unter Performanz-Problemen
- Parallele Maschinen mit unterschiedlicher Semantik

Maschinen zur Realisierung von Logischen Programmiersprachen (2)

Realisierung:

PROLOG-Maschinen basieren meist auf der Übersetzung von PROLOG in *Warren Abstract Machine*- (WAM) Code.

WAM-Realisierung:

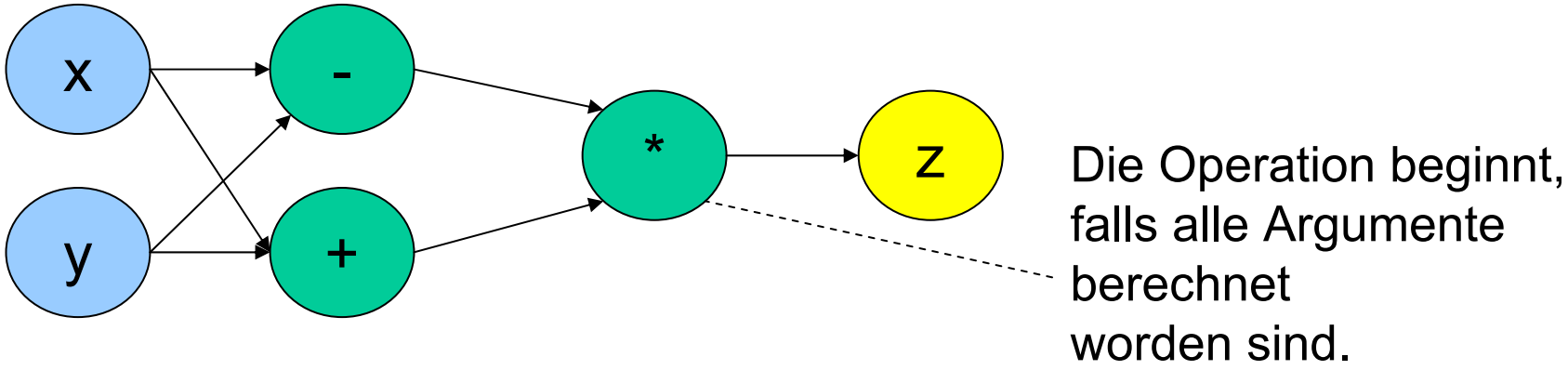
- WAM von Maschinenprogrammen interpretiert,
- WAM-Befehle in Maschinencode übersetzt
- WAM in Hardware

Beispiele von Maschinen:

- Entwürfe des *5th Generation Projekts*.
- PRIPs = Entwurf der Projektgruppe PRIPs.

2.11 Datenflussmaschinen

Verfügbarkeit von Daten veranlasst Ausführung von Operationen: Beispiel: **z = (x + y) * (x - y);**
x, y, z: Benennungen für Werte.



Modellierung mit Marken: Gültige Daten werden als Marken dargestellt. Eine Operation kann ausgeführt werden, falls alle ihre Argumente markiert sind. Potenziell viele Operationen gleichzeitig!

Vorteile der Datenflussrechner

Vorteile

- eingebaute Parallelität, eingebaute Synchronisation; vereinfachte Parallelisierung
- Adressen sind nach außen nicht sichtbar
- vorteilhaft bei gemeinsamen Teilausdrücken
- beliebige Reihenfolge der Abarbeitung bereiter Befehle

Historie der Datenflussrechner

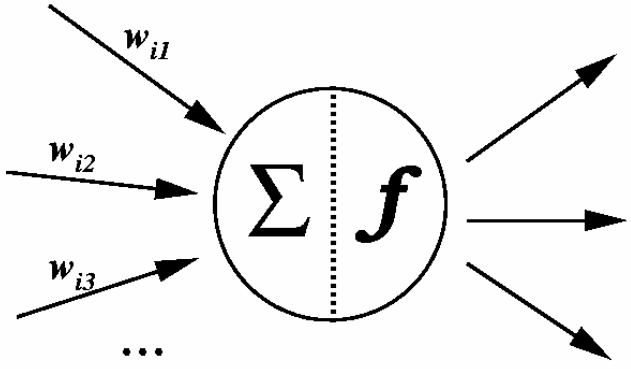
- Euphorie Ende der 70er Jahre
- Stille Ende der 80er Jahre
- Prinzipien später teilweise in andere Systeme integriert
  - Datenflussrechner können leicht asynchron realisiert werden, benötigen so wenig Energie (z.B. Entwicklung . Sharp/U.Kochi/U.Osaka für Videokameras)
  - Von-Neumann-Rechner mit dynamischen *Scheduling* (*Scoreboarding*, Tomasulo-Algorithmus) [Kap.3] haben das Datenflussprinzip intern übernommen.
  - Datenflusssprachen: LabView, Simulink, ….

☞ „Datenflussrechner“ sind Realität, aber nicht in der ursprünglich angedachten Form

Weitere Nicht-Von-Neumann-Maschinen: Neuronale Netze

- Neuronale Netze emulieren Netze von Neuronen in Lebewesen
- Einsatz zur Klassifikation von Mustern und als nicht-lineare adaptive Filter
- Neuronale Netze erfordern eine Anlernphase zum Einstellen der Parameter
- Sind geeignet, wenn sonst wenig Erfahrung zur Lösung des Problems vorliegt.

$$y_i = f\left(\sum_j w_{ij} y_j\right)$$



Weitere Nicht-Von-Neumann-Maschinen: DNA-Rechner

- Im DNA-Molekül erfolgt eine Kodierung von Informationen mit 4 verschiedenen Basen.
- In einem Liter Flüssigkeit mit 6 g DNA ließen sich theoretisch ca. 3000 Exabyte an Informationen speichern.
- Die Rechenleistung läge bei ca. 10^{15} Operationen/s.
- Durch die Parallelverarbeitung könnten theoretisch kryptographische Schlüssel ermittelt werden.
- Angeblich wurde ein *traveling salesmen* –Problem gelöst – nur das Auslesen dauert lange.
- Referenz: K.H. Zimmermann, Z. Ignatova, I. Martinez-Perez: *DNA Computing Models*, Springer, 2008

Weitere Nicht-Von-Neumann-Maschinen: Quantenrechner

- Nullen und Einsen verschiedener Lösungen können sich in einem Register überlagern. Dadurch parallele Berechungen möglich.
  - Durch die Form der Parallelarbeit verändern sich die Komplexitäten von klassischen Algorithmen:
    - Algorithmus von Shor zur nicht-exponentiellen Faktorisierung auf einem Quantenrechner
    - Die Faktorisierung von 15 ist praktisch gelungen
- Quantenrechner werden Spezialaufgaben vorbehalten bleiben, sie bilden keinen Ersatz für klassische Rechner.

Weitere Nicht-Von-Neumann-Maschinen: Quantenkryptographie

- Nutzung quantenmechanischer Effekte, um kryptographische Probleme zu lösen oder um kryptographische Systeme zu überwinden*. Beispiele:
  - Nutzung von Quantenrechnern, um kryptographische Codes zu knacken
  - Quantenmechanische Kommunikation: Nutzt das quanten-mechanische Phänomen, dass allein die Beobachtung eines Zustandes eine Selektion unter Zuständen bewirkt. Tauschen Partner Schlüssel aus, so bewirkt schon die Beobachtung der Kommunikation durch einen Dritten eine Veränderung.

    Der Rekord der Übertragung per Glasfaserkabel liegt bei 184 km°

---

* Wikipedia, „Quantum computing“, Abfrage 16.4.2011

° P. A. Hiskett, D. Rosenberg, C. G. Peterson, R. J. Hughes, S. Nam, A. E. Lita, A. J. Miller, J. E. Nordholt: *Long-distance quantum key distribution in optical fibre.* In: *New Journal of Physics.* 8, Nr. 9, 2006, S. 193

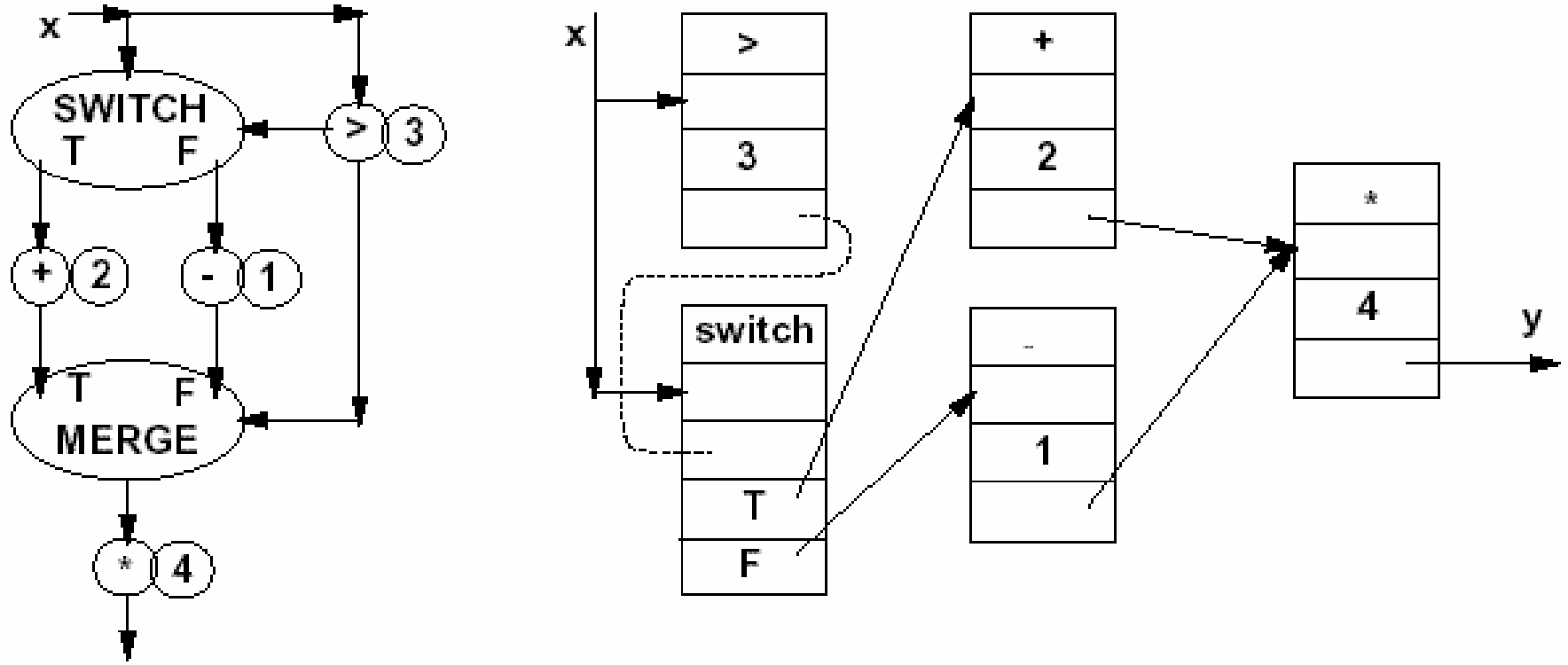
Zusammenfassung

- *Application Specific Instruction Set Processors*
- Abstrakte Maschinen
- Funktionale Programmierung und Reduktionsmaschinen
  - *graph*- und *string reduction*
- Datenflussmaschinen
- Realisierung logischer Programmiersprachen
- DNA-*Computing*
- *Quantum Computing*
  - Quantenrechner
  - Quantenkryptographie

RESERVE

Komplizierterer Fall

y := (IF x > 3 THEN x+2 ELSE x-1) * 4



Darstellung der Befehle in der Maschine (Dennis)

Als Tupel (Opcode, Plätze für Argumente, Ziel-Liste).
Die Ziel-Liste ist die Liste der Tupel, die das Ergebnis als Argument benötigen.