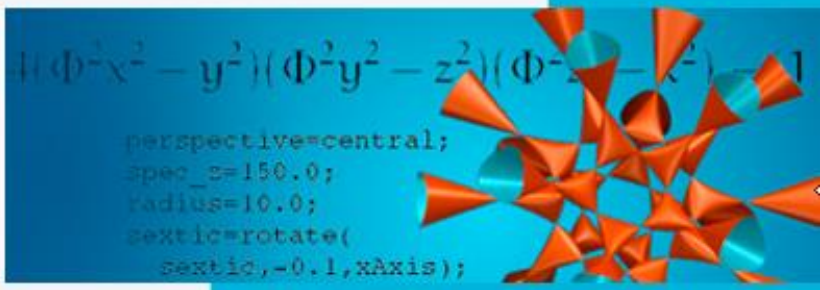
Philipps Universität Marburg

Technische Informatik I

Teil 11, Kapitel 1:

Prozessor-Architekturen



Kapitel Inhalt

- RISC- und CISC Architekturen
- RISC-Prozessoren
- Pipelining

RISC-Prozessoren

- Zu Beginn der 80er Jahre wurde der Begriff **RISC** geprägt.
- Diese Abkürzung steht für ein CPU-Konzept mit einem **reduzierten** Befehlssatz.
  - RISC = Reduced Instruction Set Computer.
- Für alle bis dahin verwendeten CPU-Konzepte wurde der Begriff **CISC** geprägt
  - CISC = Complex Instruction Set Computer.
- Das Ziel der RISC-Philosophie war es, die CPU-Architektur an neuere Entwicklungen der Hardware-Technik anzupassen.
- Die Grundidee war, einen Maschinenbefehl nicht durch ein Mikroprogramm zu implementieren, sondern ihn direkt durch einen einzigen Mikrobefehl ausführen zu lassen.
- Anfang der 80er Jahre entstanden Prototypen zur RISC-Technologie
  - Stanford MIPS, Berkeley RISC und IBM 801.
- Die ersten darauf aufbauenden kommerziellen Produkte waren nicht besonders erfolgreich.
  - Beispiel: IBM R/6000 Serie.

RISC-Prozessoren

- Ende der 80er Jahre begann dann die Blütezeit der RISC-Technologie.
- In den folgenden Jahren gab es bzw. gibt es mehrere kommerziell und technisch mehr oder weniger erfolgreiche Produktreihen, die auf RISC-Prozessoren aufbauen:
  - IBM RS/6000, PowerPC,
  - DEC α,
  - SUN Sparc, Ultra Sparc,
  - SGI Iris Indigo, Crimson, Indy 2, O2, Challenger, Origin,
  - HP PA.
- Die ARM Architecture ist in Stückzahlen gemessen wohl die erfolgreichste RISC-Familie, sie findet sich in vielen Systemen, bei denen es um hohe Leistung, geringen Stromverbrauch und niedrige Kosten geht (Typisch 100–500 MHz im Jahr 2008).
- Die ARM Ltd., die diese Systeme konstruiert, baut allerdings selbst keine Prozessoren, sondern verkauft lediglich Lizenzen für das Design an ihre Kunden. Mittlerweile sollen 10 Milliarden ARM-CPUs im Umlauf sein, die unter anderem in folgenden Systemen zum Einsatz kommen:
  - Apple: iPods (ARM7TDMI SoC),
  - Apple: iPhone (Samsung ARM1176JZF, Apple A4)

Quelle (des Textes über ARM): Wikipedia

CISC Prozessoren

- Als besonders markante Vertreter von Computerfamilien, die auf CISC-CPUs aufbauen, gelten die Familien:
  - IBM /360, /370, /390, ES9000, System z
  - DEC VAX
- Aber auch die Mikroprozessoren der x86-Serie von Intel und der 680x0-Serie von Motorola müssen diesem CPU-Konzept zugeordnet werden, auch wenn die Hersteller Wert darauf legen, bei neueren Modellen weitgehend RISC-Konzepte zu berücksichtigen.
- Dies trifft z.B. insbesondere zu auf :
  - Intel Pentium, Core Duo, i7, …

Motivation für CISC

- Die den CISC-Prozessoren zugrunde liegenden Ideen charakterisieren die Situation Anfang der 60er Jahre und gehen aus von:
  - einer relativ schnellen Arbeitsweise der CPU,
  - einem relativ langsamen Arbeitsspeicher,
  - einem sehr kleinen Arbeitsspeicher,
  - einem sehr teuren Arbeitsspeicher.
- Während die ersten beiden Punkte sich bis heute nicht wesentlich verändert haben, aber im Gegensatz zu früher durch Cache-Speicher im Wesentlichen kompensiert werden können, treffen die beiden letzten Punkte heute nicht mehr zu.
- Aus damaliger Sicht war es jedoch erstrebenswert, möglichst wenige, dafür komplexe Maschinenbefehle zu verwenden, mit dem Ziel, Programme zu verkürzen und die Zahl der Speicherzugriffe zum Laden von Instruktionen zu minimieren.

CISC und Mikroprogramme

- Ermöglicht werden komplexe Maschinenbefehle durch die **Mikroprogrammtechnik**.
- Nur durch die Einführung dieser zusätzlichen Abstraktionsschicht erhält man die Chance, eine große Zahl komplexer Befehle fehlerfrei in einer CPU zu realisieren.
- Die komplexen Maschinenbefehle werden als Einsprungpunkte in ein Mikroprogramm aufgefasst und von dem dort anzutreffenden Mikroprogramm gesteuert.
- Sie werden durch eine relativ einfache CPU-Logik ausgeführt.
- Das Mikroprogramm kann vor der Konstruktion der CPU entworfen werden und mit Hilfe von Simulationsprogrammen ausgetestet werden.
- Eine weitere Motivation für die Verwendung von Mikroprogrammen war die Implementierung von Prozessor-Familien:
  - Definiert wird eine Hard- bzw. Software-Schnittstelle in Form einer definierten Maschinenarchitektur.
  - Unterschiedlichen Mikroprogramme realisieren diese Architektur auf mehreren Modellen der Prozessorfamilie:
    - Einfache und damit billige Modelle
    - Mittlere Modelle
    - Aufwändige und damit leistungsfähige aber teure Modelle

CISC Eigenschaften

- Typische Eigenschaften von CPUs in CISC-Architektur:
  - Relativ wenige Register.
  - Die meisten Register sind keine Mehrzweckregister sondern haben Spezialfunktionen.
  - Die Maschinenbefehle haben meist sehr viele Varianten, die mit Tricks wie z.B. Modifier-Bits, Prefix-Bytes, etc. aufwändig codiert werden.
  - Insgesamt gibt es viele und teilweise sehr komplexe Maschinenbefehle.
  - Es gibt viele Typen von Operanden:
    - direkte Operanden (also explizite Werte)
    - Registeroperanden
    - Speicheroperanden (also Werte aus dem Speicher mit vielen Adressierungsarten)
- Die Speicheradressierung und die große Menge komplexer Instruktionen werden häufig damit begründet, dass auf Basis eines solchen Designs die Generierung von Maschinencode durch einen Compiler einfacher wird.
- Die obigen Punkte treffen offensichtlich auf die x86-Familie von Intel zu!

Von CISC zu RISC

- Einer der Ausgangspunkte des Übergangs zu RISC-Architekturen war die Untersuchung gängiger Compiler.
- Es wurden Statistiken bekannt, denen zufolge gängige Compiler einen großen Teil der komplexen Instruktionen überhaupt nicht verwendeten.
- Und auch dort, wo sie verwendet wurden, trugen die komplexen Instruktionen nur ca. 20 % zur Laufzeit des generierten Codes bei.
- Die meisten verwendeten Instruktionen sind dagegen so einfach, dass sie auch zur RISC Philosophie passen.
- Hinzu kam die Beobachtung, dass immer mehr Speicher innerhalb der CPU und im Arbeitsspeicher zur Verfügung stehen, so dass keine Notwendigkeit besteht, Instruktionen zusammenzustauchen.
- Cache-Speicher verringern den zusätzlichen Zeitaufwand zum Laden von **mehreren** Instruktionen.
- Eine der Maßnahmen zur Reduzierung des Platzbedarfs von CISC-Instruktionen war die Definition zahlloser Befehlsformate: So kann die Befehlslänge bei der x86-Familie von 1 bis 32 Byte variieren, wobei fast jeder Zwischenschritt möglich ist.
- Bei dem weitgehend unbekannten Prozessor iAPX 432 aus dem Jahr 1982 variierten Befehlsanfang und Befehlslänge nicht nur auf Byte-, sondern sogar auf Bitebene.

RISC-Eigenschaften

- RISC-Prozessoren sind gekennzeichnet durch:
  - wenige einfache Befehle, die möglichst in einem Maschinentakt ausgeführt werden,
  - wenige Befehlsformate, möglichst mit nur einer festen Befehlslänge,
  - viele Mehrzweckregister,
  - Speicherzugriffe nur über Load- bzw. Store-Befehle.
- Letzteres bedeutet, dass Quelle und Ziel von Operationen nur Register, nie Hauptspeicher sein können.
- Werden Operanden aus dem Speicher benötigt, so müssen sie vorher durch einen gesonderten Load-Befehl in einem Register bereitgestellt werden.
- Ein typisches Beispiel hat der MIPS-Prozessor R3000
  - er hat nur 64 Maschinenbefehle,
  - der Operationscode wird mit 6 Bit codiert,
  - es gibt drei Befehlsformate,
  - alle Befehle haben die gleiche Länge,
  - es gibt 32 Mehrzweckregister.

RISC-Prozessoren: Registerzahl

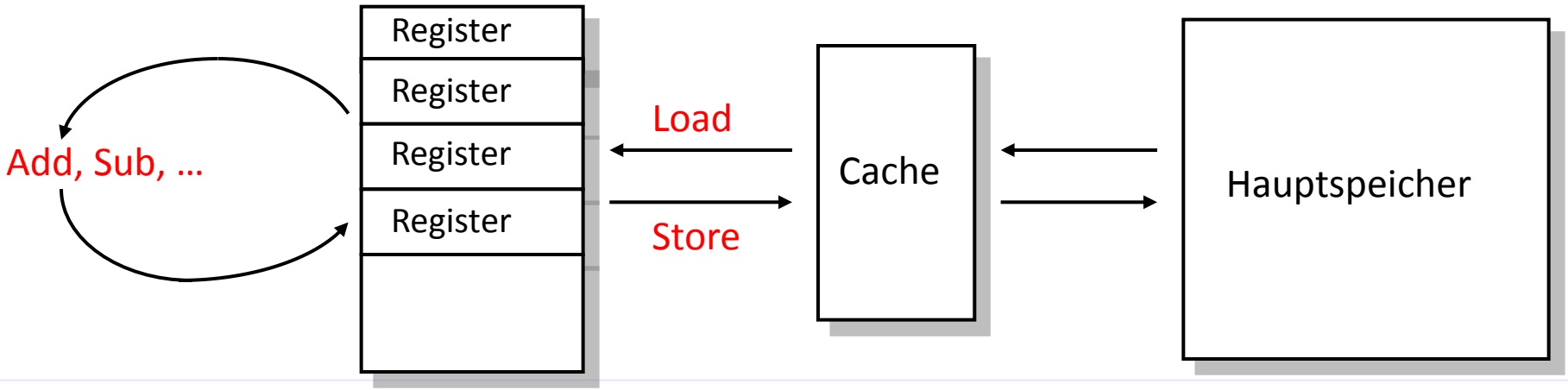
- Die Anzahl der Register war bei den ersten Prototypen der RISC-Architekturen sehr verschieden:
  - Der Stanford MIPS (Vorläufer der MIPS R Serien) hatte nur 16 Register.
  - Der Berkeley RISC (Vorläufer der SPARC-Prozessoren) hatte 138 Register.
  - Der IBM 801 (Vorläufer der IBM-POWER-Prozessoren) hatte 32 Register.
- Heute gilt die Zahl von 32 Registern als guter Kompromiss.
- Die Effekte, die man mit einer größeren Anzahl von Registern erzielen wollte, insbesondere die Verringerung von Speicherzugriffen, erreicht man heute besser mit einem On-Chip-Cache.
- Strittig ist noch die Frage, wie viele Befehle ein RISC-Prozessor haben soll.
  - Die von 64 Befehlen des MIPS R3000 erscheint aus heutiger Sicht sehr einengend.
  - Möglicherweise ist eine Codierung des Operationscodes durch 8 Bits sinnvoller.
  - Diese würde bis zu 256 Befehle zulassen und damit eigentlich der ursprünglichen RISC-Philosophie widersprechen.

RISC-Prozessoren: Motivation kleiner Befehlssätze

- Viele ursprüngliche RISC-Entwürfe gingen von einer Aufteilung der Funktionen auf mehrere Chips aus:
  - Diese sollten den Hauptprozessor und mehrere Coprozessoren realisieren.
  - Die Coprozessoren sollten über eigene Befehlssätze verfügen.
  - Daher glaubte man, für jeden einzelnen Prozessor mit wenigen Befehlen auskommen zu können.
- Mit der heutigen Integrationsdichte erscheint diese Vorgehensweise nicht mehr zeitgemäß.
- Es ist sinnvoller, Gleitpunktoperationen durch eigene Maschinenbefehle anzusprechen und nicht über eine Coprozessorschnittstelle abzuwickeln.
- Dies trifft auch auf alle anderen "ausgelagerten Befehlssätze" zu.

RISC-Prozessoren: Load- Store-Architektur

- Nach wie vor unumstritten ist die Reduktion des Speicherzugriffs auf Load- und Store- Befehle.
- Daten können nur manipuliert werden, wenn sie sich in Registern befinden.
  - Das vereinfacht das Design von CPUs und Cache's erheblich.
  - … und macht sie schneller.
  - Das passt auch sehr gut zu optimierenden Compilern:
    - Häufig verwendete Variablen können für längere Abschnitte fest einem Register zugeordnet werden.



RISC-Prozessoren: Verzögerte Ausführung

- Häufig vorkommende Befehle wie Load, Store, Add, Sub etc. werden möglichst schnell ausgeführt, d.h. in einem Maschinentakt und ohne Hilfe eines Mikroprogramms.
- Einer besonderen Optimierung bedarf es auch bei Sprungbefehlen:
  - Diese kommen sehr häufig vor.
  - Wegen eines eventuell notwendigen Speicherzugriffs zum Laden der Zieladresse können sie meist nicht in einem Takt erledigt werden.
  - Häufig findet man daher das Konzept des verzögerten Sprungs:
    - Ein Sprungbefehl wird in einem Takt abgearbeitet, springt aber erst einen Takt später.
    - Der Programmierer/Compiler hat Gelegenheit dies zu nutzen, indem er die Befehle so umsortiert, dass unmittelbar nach dem Sprungbefehl noch ein Befehl abgearbeitet wird, der logisch gesehen vor dem Sprung ausgeführt werden soll und die Sprungbedingung nicht beeinflusst.
    - Falls ein solcher Befehl nicht gefunden werden kann, muss ein NoOp-Befehl (No Operation) eingefügt werden.
- Dieses Konzept der verzögerten Wirkung kann auch auf Load- und Store-Befehle angewendet werden, falls sich die Taktzeit auf diese Weise weiter reduzieren lässt.

RISC-Prozessoren: Mikroprogramme

- Die ursprüngliche RISC-Philosophie forderte, auf Mikroprogramme ganz zu verzichten.
- Solange nur ganz wenige Befehle mehrere Takte benötigen (z.B. die Multiplikation und vor allem die Division), ließ sich das auch durchhalten.
- Heute werden wegen der hohen Integrationsdichte wieder zunehmend komplexere Befehle in der CPU ausgeführt, z.B.
  - Gleitpunktbefehle,
  - Bitblockbefehle,
  - Multimediabefehle,
- Heute fordert man daher lediglich
  - dass nur wenige Befehle mithilfe von Mikroprogrammen abgewickelt werden.

Transistorfunktionen und Leistungssteigerung

- Heute werden vor allem folgende Konzepte zur Leistungssteigerung angewendet:
  - Anwendung der Fließbandtechnik (Pipelining),
  - Parallelisierung (Superskalar-Technik).
- Diese Techniken steigern die Leistung der Prozessoren und nutzen die ungeheure Zahl von Transistorfunktionen, die in einem heutigen Chip potentiell zur Verfügung stehen.
- Im Jahr 2008 hatten die höchstintegrierten Chips (4 GBit DRAMs)
  - ca. 6.400.000.000 Transistorfunktionen.
- Die meisten bekannten CPU-Chips verwenden bisher erheblich weniger Transistorfunktionen:
  - Der Intel Core Duo E8600 hat ca. 450.000.000 Transistorfunktionen (Jahr 2008)
  - Der Intel Core i7 980X hat ca. 1.700.000.000 Transistorfunktionen (Jahr 2010)
  - Der Intel Itanium 9350 hat ca. 2.046.000.000 Transistorfunktionen (Jahr 2010)
- Relativ wenige der Transistorfunktionen werden für die CPU Logik benötigt.
- Ansätze zur Nutzung zusätzlicher Transistorfunktionen:
  - Mehrere CPU Kerne in einem Prozessorchip: 2, 4, 6 oder sogar 8 Kerne.
  - Integrierte L1-, L2 und L3 Caches.
    - Beim i7-980X: 6 Kerne; je Kern 64 kB L1 und 256 kB L2-Cache; ein L3 Cache von 12 MB für alle Kerne.

Pipelining

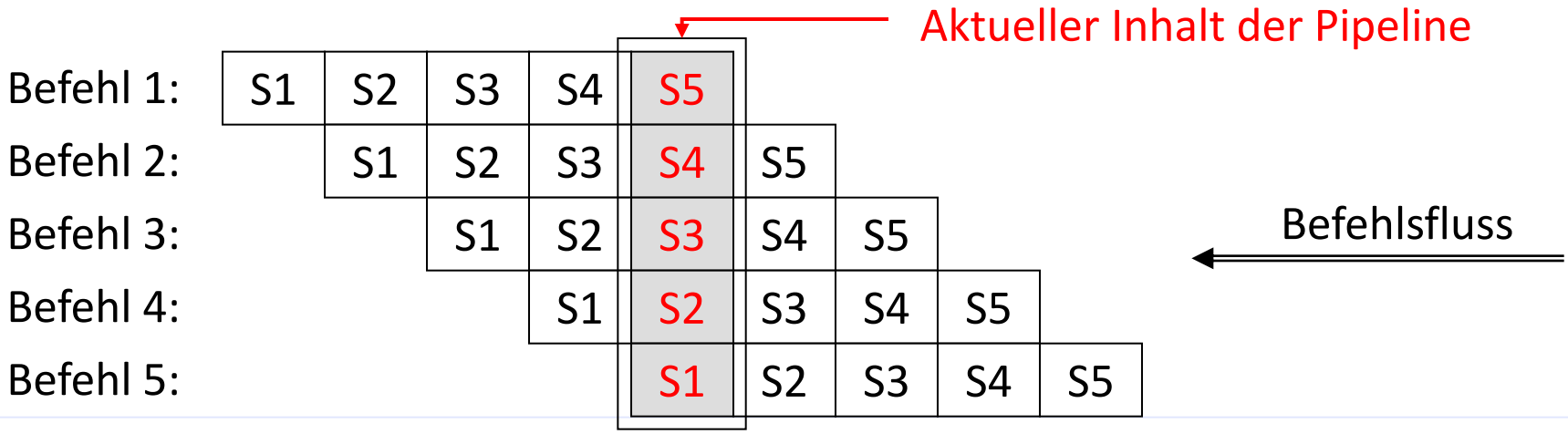
- Eine **Pipeline** ist eine Warteschlange, in der sich die als Nächstes abzuarbeitenden Befehle befinden.
- Jeder Befehl besteht aus einer Reihe von Phasen. Während noch die letzten Phasen der vorderen Befehle in der Pipeline abgearbeitet werden, kann bereits mit den ersten Phasen der nächsten Befehle begonnen werden.
- Mithilfe der Pipeline-Technik lassen sich die Taktzeiten einer CPU weiter reduzieren, wobei angestrebt wird, dass die durchschnittliche Ausführungszeit eines Befehls nahe bei einem Takt liegt.
- Der Befehl wird in mehrere Phasen aufgeteilt, die nacheinander, aber gleichzeitig mit anderen Phasen anderer Befehle, in einer Pipeline ausgeführt werden.
- Während eine Phase eines Befehls bearbeitet wird, erledigt die Pipeline schon andere Phasen weiterer Befehle.
  - Heute sind 5- bis 35-stufige Pipelines üblich.
- Bei einer 5-stufigen Pipeline könnte die Phasen-Aufteilung für einen Register/Register-Befehl etwa folgendermaßen aussehen:

Pipelining

- Bei einer 5-stufigen Pipeline könnte die Phasen-Aufteilung für einen Register/Register-Befehl etwa folgendermaßen aussehen:

S1:	**Befehlsbereitstellung**
S2:	**Dekodieren des Befehls**
S3:	**Lesen der beteiligten Register**
S4:	**ALU-Operation**
S5:	**Schreiben in das Ziel-Register**

- Dabei werden bis zu fünf Befehle gleichzeitig überlappend bearbeitet.
- Während die Ergebnisse des 1. Befehls noch in ein Register übertragen werden, wird bereits der 5. Befehl bereitgestellt, der 4. Befehl dekodiert usw.



Pipelining

- Die Verwendung einer Pipeline setzt voraus, dass zwischen den 5 beteiligten Befehlen keine störenden Zwischenbeziehungen existieren.

- Beispiel:

  - **Da der 2. Befehl seine Register bereits gelesen hat, dürfen diese nicht mit dem Register übereinstimmen, das der 1. Befehl noch schreiben will.**

- Es gibt Techniken, solche Zwischenbeziehungen auf Hardwareebene zu entdecken.
- In einem solchen Fall muss die Pipeline zwischen den beteiligten Befehlen so lange angehalten werden bis Konsistenz vorliegt.

Gibt es Fragen?

Anregungen oder Verbesserungsvorschläge können auch gerne per E-mail an mich gesendet werden: Kontakt