**Stephan Brumme, SST, 3.FS, Matrikelnr. 70 25 44**
mail@stephan-brumme.com

## Aufgabe 1

Ich erkläre die Schichten, indem ich mit der untersten beginne und mich zu den oberen und komplexeren vorarbeite:

Die Rechnerarchitektur ist der komplette Hardware-Teil eines EDV-Anlage. Sie definiert sowohl alle mechanischen Schnittstellen zur Umwelt, als auch das gesamte Zusammenspiel der Hardware-Komponenten im System, wie Speicher, Rechenwerk usw..

Das Betriebssystem stellt eine abstrahierte Schnittstelle zur Rechnerarchitektur her. Damit erlaubt man den auf dem Rechner laufenden Programmen eine einheitliche Umgebung, die nicht an Details und Spezifika einzelner Komponenten gebunden ist. Idealerweise laufen alle Hardwarezugriffe über Betriebssystemschnittstellen.

Die Basissysteme sind die erste logische Schicht in diesem Schema, sie sind vollkommen unabhängig von der Hardware. Sie stellen die organisierte und strukturierte Benutzung von Computersystemen durch Menschen (meist Programmierer und Administratoren) sicher. Allerdings sind sie noch keine Programme zur Problemlösung, sondern eher Hilfsmittel. Ich würde sie mit der Verwaltung in einer Institution vergleichen.

Der End-Benutzer kommt in der Regel nur mit der Anwendungssoftware in Kontakt. Sie ist einzig und allein für die Lösung von Anwendungsproblemen erstellt worden. Sie ist vollkommen unabhängig von Details der darunter liegenden Schichten.

Jede Schicht sollte idealerweise alle übergeordneten Schichten von allen untergeordneten Schichten durch Abstraktion trennen.

## Aufgabe 2

Die Idee des von-Neumann-Computers besteht darin, den Rechner unabhängig von der Aufgabenstellung (die das Programm darstellt) zu gestalten. Dies erfordert eine Unterteilung in Steuerwerk, Rechenwerk, Speicher und I/O-Peripherie. Diese Universalität ist das Erfolgsgeheimnis der Architektur.

Die Programme werden linear abgearbeitet, dabei wird nur-lesend (Programmcode) und lesend/schreibend (Daten) auf den Speicher zugegriffen. Der Speicher beinhaltet beide Arten (Code und Daten), die theoretisch gemischt sein können, in der Praxis versucht man eine Trennung auf Quellcodeebene durchzusetzen.

Das Steuerwerk muss dazu den Programmcode auswerten und Speicherzugriffe ausführen bzw. Aufträge an das Rechenwerk delegieren. Der Speicher wird anfänglich vom Eingabewerk gefüllt, die Ausgaben werden an das Ausgabewerk weitergeleitet. Das Steuerwerk kann diese beiden ebenso beeinflussen (z.B. Bildschirmdialoge).

Sämtliche Kommunikation erfolgt über einen zentralen Bus, der vom Steuerwerk kontrolliert wird. Dieser unterteilt sich in Steuer-, Adress- und Datenleitungen. Das Steuerwerk fordert über die Steuerleitungen die Peripherie auf, die an den Adressleitungen angelegte Adresse zu lesen oder zu schreiben und über die Datenleitungen Daten zu lesen bzw. zu schreiben. Somit sind Adress- und Datenleitungen bidirektional, die Steuerleitungen unidirektional (Ausnahme: moderne Busmaster-Peripherie) mit dem Steuerwerk als Sender und der Peripherie als Empfänger.

Unzureichende Busbreiten bremsen die Einzelbestandteile der Architektur und verhindern eine Ausschöpfung der theoretischen Möglichkeiten des Systems. Die alleinige Kontrolle des Steuerwerks verbietet eine Nutzung des Busses, wenn er frei, das Steuerwerk aber gerade beschäftigt ist. Eine Aufhebung der strikten Unidirektionalität durch busmaster-fähige Peripherie umgeht diesen Schwachpunkt der von-Neumann-Architektur.

Speicher ist im Zugriff um ein Vielfaches langsamer als das Steuerwerk/Rechenwerk. Geschickte Cache-Strategien erlauben eine Minimierung der Speicherzugriffe und tragen erheblich zur Beschleunigung des Systems bei. Die Effektivität der Caches steigt, wenn man eine Trennung von Code und Daten erzielt.

**Aufgabe 3**

a) Sowohl RISC als auch CISC sind Akronyme:
- RISC: Reduced Instruction Set Computer
- CISC: Complex Instruction Set Computer

Die Idee von CISC-Systeme beruht auf der Annahme, die Architektur soll enorm viele Befehle kennen. Durch ihre „feste Verdrahtung" erhoffte man sich eine hohe Geschwindigkeit bei der Ausführung dieser komplexen Befehle. Problematisch ist der hohe Speicherbedarf eines Einzelbefehls. Architekturen mit variablen Befehlslängen umschiffen zwar diesen Schwachpunkt, erhöhen aber im Gegenzug die Komplexität des Befehlsdekoders.

Im Gegensatz dazu setzt RISC auf die Benutzung eines nur kleinen Befehlsumfanges, da sich in der Praxis gezeigt hat, dass nur ein geringer Teil von Befehlen in Programmen auch tatsächlich verwendet wird. Diese werden dann hochoptimiert auf dem Chip umgesetzt. Somit vereinfacht sich u.a. auch der Befehlsdekoder. Nachteilig ist, dass die komplexen Befehle, die im Gegensatz zu CISC fehlen, durch Befehlsfolgen ersetzt werden müssen. Ein RISC-Programm wird deshalb meist mehr Befehle zur Umsetzung eines Algorithmus' benötigen als ein CISC-System.

b) Die MIPS der Beispielrechnerarchitektur liegt bei ($CPI$ bedeutet cycles per instruction, $t$ ist die Dauer eines Taktes):

$$MIPS = \frac{10^6}{CPI \cdot t}$$

$$t = 10^{-9} s$$

$$CPI = \frac{30{,}4\% \cdot 1{,}5 + 10\% \cdot 1 + 3{,}8\% \cdot 10 + 9{,}5\% \cdot 7 + 6{,}5\% \cdot 15 + 3\% \cdot 1 + 20\% \cdot 1{,}5 + 10\% \cdot 2 + 6{,}8\% \cdot 2}{100\%}$$

$$MIPS = \frac{10^3}{3{,}242} \approx 308$$

(1) Das Akronym MIPS steht für „Million Instructions Per Second". Es wird jedoch nicht exakt definiert, was eine Instruktion ist, ihr Komplexitätsgrad bleibt offen. Hierbei können große Differenzen zwischen Rechner mit verschiedenen Befehlssätzen auftreten. So sind i.a. RISC-Rechner den CISC-Rechnern bei gleichem Prozessortakt hinsichtlich des MIPS-Wertes deutlich überlegen, da ihr CPI-Wert niedriger ist. Vergleicht man dann die Ausführungszeiten von Algorithmen oder Programmen, so sind beide eventuell gleich schnell. Die Begründung für dieses Phänomenen lieferte ich bereits unter Punkt a) dieser Aufgabe.

(2) Die Berechnungsgrundlage des MIPS-Wertes ist ein Befehlsmix. Dabei orientiert man sich an einer statistischen Verteilung der verschiedenen Befehle. Je nachdem, welche Programme man vorher zur Erstellung dieser Statistik analysierte, ergeben sich unterschiedliche Verteilungen. Während eine Tabellenkalkulation eher Integer- und Floating-Point-lastig ist, weist eine Datenbank mehr Speicherzugriffe und logische Operationen auf.
Diese Problematik kann man durch eine breite Basis an zugrundeliegenden Programmen entschärfen (z.B. BapCo-Suite für Windows-Systeme).

(3) Die MIPS-Zahl lässt sich auf zwei Arten erhöhen:
- Erhöhung der Taktfrequenz
- Optimierung der Architektur

Der erste Punkt ist nicht immer einfach umzusetzen, man stößt schnell an technische (und mittlerweile physikalische) Grenzen. Der zweite Weg kann im Idealfall den CPI-Wert auf 1 drücken, Techniken wie Parallelisierung berücksichtige ich nicht. Da man einige Befehle nicht derart optimieren kann, muss man sie aus der Architektur entfernen. Der Compiler hat dann die Aufgabe, die vormals hardware-implementierten Algorithmen in Software nachzubilden. Diese Vorgehensweise ergibt einen höheren MIPS-Wert auf Kosten der Programmlänge. Das Programm wird dadurch nicht schneller, sondern sogar eventuell effektiv langsamer.

## Aufgabe 4

Man muss zwischen zwei Arten von Sprungbefehlen unterscheiden: bedingte und unbedingte. Letztere sind absolut unproblematisch, da sie deterministisch sind. Es sind zuverlässig ermittelbar, wo die Befehlsausführung nach dem Sprung fortgesetzt wird, daher können die nachfolgenden Befehle ohne Nebeneffekte geladen und vorverarbeitet werden. Vorsicht ist bei bedingten Sprüngen geboten: solange das Ergebnis der Bedingung nicht vorliegt, ist nicht klar, welcher Befehl auf den Sprungbefehl folgt.

Diese Problematik versucht man auf verschiedenem Wegen zu lösen. Am gebräuchlichsten sind Sprungvorhersagen, die auf vorherigen Schleifendurchläufen desselben Codefragments beruhen. Der Prozessor lädt dann den in den bisherigen Durchläufen wahrscheinlicheren der beiden möglichen Folgebefehle. Sollte die Vorhersage fehlschlagen, müssen diese falsch vorhergesagten Folgebefehle aus der Pipeline entfernt werden, was viel Zeit kosten. Je mehr Stufen die Pipeline umfasst, desto höher ist der Performance-Verlust, beim Pentium 4 sind dies ca. 20 Takte, beim Athlon nur etwa 10. Allerdings liegt die Effizienz der Vorhersage bei weit über 90%, so dass dieser Fall weitgehend vermieden werden kann.

Ein anderer Weg sind Out-of-Order-Execution-Techniken. Dabei bedient man sich der Tatsache, dass man die Sprungbedingung durch Befehlsumgruppierungen schon vor dem Sprung auswerten kann.

Neben den Sprungbefehlen gibt es noch andere Befehlsabhängigkeiten. Von-Neumann-Rechner folgen der SISD-Architektur (Singe Instruction, Single Data). Sie erlauben die Abhängigkeit eines Befehles von einem Vorgängerbefehl. Als Beispiel verwende ich folgende Pseudo-Befehlssequenz:

```
a = 5
b = 2a
```

Der zweite Befehl muss warten, bis das Ergebnis des ersten vorliegt. Somit kann die Operand-Fetch-Phase des zweiten Befehls erst *nach* der Write-Result-Back-Phase des ersten erfolgen. Der Prozessor hat dies zu erkennen und muss selbstständig einen Wartezyklus einfügen. Tut er dies nicht, sind Inkonsistenzen und unvorhersagbares Verhalten die Folge.