RS-Übung 1

Arne Beer (MN 6489196),
Rafael Epplee (MN 6269560),
Julian Polatynski (MN 6424884)

December 3, 2012

1 Begriffsbildung

1.1 Interpreter

Ein Interpreter interpretiert Befehle einer jeweils höheren Ebene, indem er die zu Befehlen einer Ebene n zugeordneten Befehle einer Ebene n-1 ausfuehrt.

1.2 Compiler

Erzeugt ein Programm, in dem die urspruenglichen Befehle einer höheren Ebene in die Befehle einer niedrigeren Ebene uebersetzt werden (meist in die Befehle der Assembler-Ebene). Dafuer wird ein Befehl in Sprache n meist zu mehreren Befehlen in Ebene $n - x$ (wobei $n, x \in \mathbb{N}$) umgewandelt. Das uebersetzte Programm wird danach gespeichert und ausfuehrbar gemacht.

Sowohl Interpreter als auch Compiler lesen die zu uebersetzenden Befehle aus dem Speicher.

1.3 Virtuelle Maschine

Eine Virtuelle Maschine stellt eine Umgebung fuer Programme bereit, in der sie nichts mehr ueber die Befehle von niedrigeren Ebenen wissen muss. Die auf der Virtuellen Maschine laufenden Programme sind in einer Sprache einer höheren Ebene geschrieben. Somit entspricht ein Interpreter dem Konzept der Virtuellen Maschine.

2 Ebenen eines Digitalrechners

Die Zeit t_0, die zum Ausführen eines Befehls auf Ebene 0 benötigt wird, wird mit $t_0 = k$ bestimmt.

Da n die Anzahl der Befehle die auf Ebene 0 für die Ausführung eines Befehls auf Ebene 1 benötigt werden repräsentiert, wäre die Formel für Ebene 1 wie folgt:

$$t_0 \cdot n = t_1$$
$$\Leftrightarrow k \cdot n$$

Ähnlich wäre sie für Ebene 2:

$$t_1 \cdot n = t_2$$
$$\Leftrightarrow t_0 \cdot n$$
$$\Leftrightarrow k \cdot n^2$$

Da sich dieses Muster wiederholt, kann man eine allgemeine Formel ableiten:

$$k \cdot n^i = t$$

3 Von-Neumann-Architektur

Die Speicherung von Daten und Programmen in der selben Komponente bringt Vorteile und Nachteile mit sich. Vorteile sind:

- Flexibilität (Programme können schnell ausgetauscht / verändert werden)
- Programme können sich reparieren
- Minimaler Hardware- und Speicheraufwand führt zu Kostenminderung.
- Die Zugriffsweise auf Daten und Programme ist diesselbe, was die Anzahl der Schnittstellen vermindert.

Nachteile sind:

- Geringere Performance aufgrund der Verzögerung beim Zugriff auf den Speicher
- Programme können zur Laufzeit modifiziert werden, was gefährlich sein kann (z.B. Selbstzerstörung)
- Es gibt keine spezielle Schutzkomponente zwischen Daten und Programm (Außer eventuell Software).

4 Optimierung von Rechenabläufen

4.1 Horner-Schema

Ohne Horner-Schema ergibt sich folgende Rechnung:

$$y = (a \cdot x^6 + b \cdot x^5 + c \cdot x^4 + d \cdot x^3 + e \cdot x^2 + f \cdot x + g)$$

$$\Leftrightarrow t = 6 \cdot 6ns + 1ns + 5 \cdot 6ns + 1ns + 4 \cdot 6ns + 1ns + 3 \cdot 6ns + 1ns + 2 \cdot 6ns + 1ns + 6ns + 1$$

$$\Leftrightarrow t = 37 + 31 + 25 + 19 + 13 + 7 = 68 + 44 + 20$$

$$\Leftrightarrow t = 112 + 20 = 132ns$$

Mit dem Horner-Schema hingegen sieht das ganze etwas anders aus:

$$y = (g + x \cdot (f + x \cdot (e + x \cdot (d + x \cdot (c + x \cdot (b + x \cdot a))))))$$

Die obige Rechnung enthält 6 Additionen und 6 Subtraktionen. Daraus ergibt sich

$$t = 6 \cdot 1ns + 6 \cdot 6ns = 42ns$$

4.2 Effizientes Berechnen von Potenzen

Hier errechnen wir zunächst einige Zwischenwerte, indem wir Potenzen potenzieren:

$$z = (x + 1)$$

$$z^2 = a$$

$$z^4 = a^2 = b$$

$$z^8 = b^2 = c$$

$$z^16 = c^2 = d$$

Die Berechnung dieser Potenzen benötigt folgende Anzahl an Nanosekunden:

$a = z^2 = (x + 1)^2$: Eine Addition und eine Multiplikation ergeben $7ns$.

$b = a^2$: Da a ein eindeutiger, bereits berechneter Wert ist, brauchen wir hier nur eine Multiplikation ($= 6ns$).

$c = b^2$: Auch hier wieder nur eine Multiplikation, also $6ns$.

$d = c^2$: Eine Multiplikation, $6ns$.

Wir benötigen also um a, b, c und d zu berechnen 25 Nanosekunden.

Der wichtigste Teil besteht aus dem Multiplizieren der verschiedenen Zwischenergebnisse, um letztendlich z^{23} herauszubekommen:

$$z^{16} \cdot z^4 \cdot z^2 \cdot z = z^{23}$$

$$\Leftrightarrow d \cdot b \cdot a \cdot z = z^{23}$$

Für diesen Teil benötigen wir noch einmal 3 Multiplikationen, also $3 \cdot 6ns$. $25ns + 18ns$ ergibt $43ns$, was die Gesamtzahl der benötigten Nanosekunden ist.

5 "Life-Log" und "Moore's Law"

5.1 Datenmengen

Bei 5MB/sec würden bei $60 \cdot 60 \cdot 24 = 86.400$ Sekunden am Tag $86.400 \cdot 5 = 432.000$ MB pro Tag anfallen.

Im Jahr gibt es (mit Schaltjahren) 365,25 Tage, was bedeutet, dass die MB pro Jahr wie folgt berechnet werden:

$$365,25 \cdot 432.000 = 157.788.000$$

Mit 80 multipliziert ergibt sich die Datenmenge eines 80jährigen Lebens.

$$157.788.000 \cdot 80 = 12.623.040.000$$

Ein 80jähriger Mensch würde also in seinem Leben Videomaterial im Umfang von ca. 12.6 Petabyte aufzeichnen.

5.2 Festplattenwachstum

Wir rechnen zuerst 2 Tibibyte in Megabyte um, um mit unserer Zahl aus 5.1 rechnen zu können:

$$\frac{2 \cdot 2^{40}}{10^6} \simeq 2199023$$

In der folgenden Gleichung ist t die Anzahl der Jahre, in denen Die Festplatten eine Größe von ca. 12.6 Petabyte erreicht haben.

$$\frac{2 \cdot 2^{40}}{10^6} \cdot 1,45^t = 12.623.040.000$$

$$log_{1,45}(\frac{12.623.040.000}{\frac{2 \cdot 2^{40}}{10^6}}) = t \simeq 23.2942$$

Nach ca. 23 Jahren passt ein ganzes Leben auf einer einzigen Festplatte.

5.3 SD-Cards

Gleiches Vorgehen wie bei der Festplatte. Umrechnung von 32 Gibibyte in Megabyte:

$$\frac{32 \cdot 2^{30}}{10^6}$$

Wachstumsgleichung:

$$1,52^t \cdot \frac{32 \cdot 2^{30}}{10^6} = 12.623.040.000$$

$$\Leftrightarrow t = log_{1.52}(\frac{12.623.040.000}{\frac{32 \cdot 2^{30}}{10^6}}) \simeq 14.1062$$

Nach ca. 14 Jahren ist eine SD-Karte groß genug, um ein gesamtes Leben aufzunehmen.

5.4 Magnetbänder

Folgende einfache Formel löst das Problem:

$$\frac{12.623.040.000}{140} = 90164571\frac{3}{7}$$