9 SPEICHER

Etwas aufzuschreiben (vor ganz langer Zeit in Bildern, später dann eben mit Zeichen) bedeutet, etwas zu speichern. Weil es naheliegend und einfach ist, reden wir im Folgenden nur über Zeichen.

Wenn man über Speicher reden will, muss man über Zeit reden. Speichern bedeutet, etwas zu einem Zeitpunkt zu schreiben und zu einem späteren Zeitpunkt zu lesen.

Die ersten Sätze dieser Vorlesungseinheit über Speicher haben Sie vermutlich von oben nach unten Ziele für Zeile und in jeder Zeile von links nach rechts gelesen. Für diese Reihenfolge ist der Text auch gedacht. Man könnte aber auch relativ einfach nur jede dritte Zeile lesen. Mit ein bisschen mehr Mühe könnte man auch nachlesen, welches das zweiundvierzigste Zeichen in der dreizehnten Zeile ist. Und wenn Sie wissen wollen, was auf Seite 33 in Zeile 22 Zeichen Nummer 11 ist, dann bekommen Sie auch das heraus.

Man spricht von *wahlfreiem Zugriff*. *wahlfreier Zugriff*

Auch Rechner haben bekanntlich Speicher, üblicherweise welche aus Halbleitermateralien und welche aus magnetischen Materialien. Auch für diese Speicher können Sie sich vorstellen, man habe wahlfreien Zugriff. Die gespeicherten Werte sind Bits bzw. Bytes. Diejenigen „Gebilde“, die man benutzt, um eines von mehreren gespeicherten „Objekten“ auszuwählen, nennt man *Adressen*.

9.1 BIT UND BYTE

Das Wort „Bit“ hat verschiedene Bedeutungen. Eine zweite werden Sie in der Vorlesung „Theoretische Grundlagen der Informatik“ kennen lernen. Die erste ist: Ein *Bit* ist ein Zeichen des Alphabetes $\{0, 1\}$. *Bit*

Unter einem *Byte* wird heute üblicherweise ein Wort aus acht Bits verstanden (früher war das anders und unterschiedlich). Deswegen wird in manchen Bereichen das deutlichere Wort *Octet* statt Byte bevorzugt. Das gilt nicht nur für den frankophonen Sprachraum, sondern zum Beispiel auch in den in früheren Kapiteln schon erwähnten *Requests for Comments* der *Internet Engineering Task Force* (siehe `http://www.ietf.org/rfc.html`, 13.11.13). *Byte* *Octet* *RFC* *IETF*

So wie man die Einheiten Meter und Sekunde mit m bzw. s abkürzt, möchte man manchmal auch Byte und Bit abkürzen. Leider gibt es da Unklarheiten:

- Für Bit wird manchmal die Abkürzung „b“ benutzt. Allerdings ist das „b“ schon die Abkürzung für eine Flächeneinheit, das „barn“ (wovon Sie vermutlich noch nie gehört haben). Deswegen schreibt man manchmal auch „bit“.

- Für Bytes wird oft die Abkürzung „B“ benutzt, obwohl auch „B“ schon die Abkürzung für eine andere Einheit ist (das Bel; Sie haben vielleicht schon von deziBel gehört).
- Für Octets wird die Abkürzung „o“ benutzt.

9.2 BINÄRE UND DEZIMALE GRÖSSENPRÄFIXE

Früher waren Speicher klein (z. B. ein paar Hundert Bits), heute sind sie groß: Wenn zum Beispiel die Adressen für einen Speicherriegel aus 32 Bits bestehen, dann kann man $2^{32} = 4\,294\,967\,296$ Bytes speichern. Festplatten sind noch größer; man bekommt heute im Laden problemlos welche, von denen der Hersteller sagt, sie fassten 1 Terabyte. Was der Hersteller damit (vermutlich) meint sind $1\,000\,000\,000\,000$ Bytes.

Solche Zahlen sind nur noch schlecht zu lesen. Wie bei sehr großen (und sehr kleinen) Längen-, Zeit- und anderen Angaben auch, benutzt man daher Präfixe, um zu kompakteren übersichtlicheren Darstellungen zu kommen (Kilometer (km), Mikrosekunde (µs), usw.)

10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}	10^{-18}
1000^{-1}	1000^{-2}	1000^{-3}	1000^{-4}	1000^{-5}	1000^{-6}
milli	mikro	nano	pico	femto	atto
m	μ	n	p	f	a
10^{3}	10^{6}	10^{9}	10^{12}	10^{15}	10^{18}
1000^{1}	1000^{2}	1000^{3}	1000^{4}	1000^{5}	1000^{6}
kilo	mega	giga	tera	peta	exa
k	M	G	T	P	E

Im Jahre 1999 hat die *International Electrotechnical Commission* „binäre Präfixe“ analog zu kilo, mega, usw. eingeführt, die aber nicht für Potenzen von 1000, sondern für Potenzen von 1024 stehen. Motiviert durch die Kunstworte „kilobinary“, „megabinary”, usw. heißen die Präfixe *kibi*, *mebi*, usw., abgekürzt Ki, Mi, usw.

2^{10}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}
1024^{1}	1024^{2}	1024^{3}	1024^{4}	1024^{5}	1024^{6}
kibi	mebi	gibi	tebi	pebi	exbi
Ki	Mi	Gi	Ti	Pi	Ei

9.3 SPEICHER ALS TABELLEN UND ABBILDUNGEN

Um den aktuellen Gesamtzustand eines Speichers vollständig zu beschreiben muss man für jede Adresse, zu der etwas gespeichert ist, angeben, welcher Wert unter dieser Adresse abgelegt ist. Das kann man sich zum Beispiel vorstellen als eine große Tabelle mit zwei Spalten: in der linken sind alle Adressen aufgeführt und in der rechten die zugehörigen Werte (siehe Abbildung 9.1).

Adresse 1	Wert 1
Adresse 2	Wert 2
Adresse 3	Wert 3
Adresse 4	Wert 4
⋮	⋮
Adresse n	Wert n

(a) allgemein

000	10110101
001	10101101
010	10011101
011	01110110
100	00111110
101	10101101
110	00101011
111	10101001

(b) Halbleiterspeicher

Abbildung 9.1: Speicher als Tabelle

Mathematisch kann man eine solche Tabelle zum Beispiel auffassen als eine Abbildung, deren Definitionsbereich die Menge aller Adressen und deren Wertebereich die Menge aller möglichen speicherbaren Werte ist:

$$m : \text{Adr} \to \text{Val}$$

9.3.1 Hauptspeicher

Betrachten wir als erstes einen handelsüblichen Rechner mit einem Prozessor, der 4 GiB adressieren kann und mit einem Hauptspeicher dieser Größe ausgerüstet ist. Bei Hauptspeicher ist die Menge der Adressen fest, nämlich alle Kombinationen von 32 Bits, und was man unter einer Adresse zugreifen kann, ist ein Byte. Jeder Speicherinhalt kann also beschrieben werden als Abbildung

$$m : \{0,1\}^{32} \to \{0,1\}^{8}$$

Der in einem Speicher im Zustand m an einer Adresse $a \in \text{Adr}$ gespeicherte Wert ist dann gerade $m(a)$.

Die Menge der Adressen ist hier fest, und bezeichnet im wesentlichen einen physikalischen Ort auf dem Chip, an dem ein bestimmtes Byte abgelegt ist. Das entspricht einer Angabe wie

Am Fasanengarten 5
76131 Karlsruhe

auf einem Brief.

Nicht nur den Zustand eines Speichers, sondern auch das Auslesen eines Wertes kann man als Abbildung formalisieren. Argumente für eine solche Funktion Auslesefunktion memread sind der gesamte Speicherinhalt m des Speichers und die Adresse a aus der ausgelesen wird, und Resultat ist der in m an Adresse a gespeicherte Wert. Also:

$$\begin{aligned} \text{memread} : \text{Mem} \times \text{Adr} &\to \text{Val} \\ (m, a) &\mapsto m(a) \end{aligned}$$

Dabei sei Mem die Menge aller möglichen Speicherzustände, also die Menge aller Abbildungen von Adr nach Val.

Hier sind nun zwei allgemeine Bemerkungen angezeigt:

1. Man lasse sich nicht dadurch verwirren, dass die Funktion memread eine (andere) Funktion m als Argument bekommt. Das Beispiel soll gerade klar machen, dass daran nichts mystisch ist.
2. Wir werden noch öfter die *Menge aller Abbildungen* der Form $f : A \to B$ von einer Menge A in eine Menge B notieren wollen. Hierfür benutzen wir die Schreibweise B^A. (Anlass für diese Schreibweise war vielleicht die Tatsache, dass für endliche Mengen A und B gilt: $|B^A| = |B|^{|A|}$.)

Wir hätten oben also auch schreiben können:

$$\begin{aligned} \text{memread} : \text{Val}^{\text{Adr}} \times \text{Adr} &\to \text{Val} \\ (m, a) &\mapsto m(a) \end{aligned}$$

Das Speichern eines neuen Wertes v an eine Adresse a in einem Speicher m kann man natürlich auch als Funktion notieren. Es sieht dann ein wenig komplizierter aus als beim Lesen:

$$\begin{aligned} \text{memwrite} : \text{Val}^{\text{Adr}} \times \text{Adr} \times \text{Val} &\to \text{Val}^{\text{Adr}} \\ (m, a, v) &\mapsto m' \end{aligned}$$

Dabei ist m' dadurch festgelegt, dass für alle $a' \in \text{Adr}$ gilt:

$$m'(a') = \begin{cases} v & \text{falls } a' = a \\ m(a') & \text{falls } a' \neq a \end{cases}$$

Was ist „das wesentliche“ an Speicher? Der allerwichtigste Aspekt überhaupt ist sicherlich dieser:

- Wenn man einen Wert v an Adresse a in einen Speicher m hineinschreibt und danach den Wert an Adresse a im durch das Schreiben entstandenen Speicher liest, dann ist das wieder der Wert v. Für alle $m \in \text{Mem}$, $a \in \text{Adr}$ und $v \in \text{Val}$ gilt:

$$\text{memread}(\text{memwrite}(m, a, v), a) = v \tag{9.1}$$

So wie wir oben memread und memwrite definiert haben, gilt diese Gleichung tatsächlich.

Allerdings man kann sich „Implementierungen“ von Speicher vorstellen, die zwar diese Eigenschaft haben, aber trotzdem nicht unseren Vorstellungen entsprechen. Eine zweite Eigenschaft, die bei erstem Hinsehen plausibel erscheint, ist diese:

- Was man beim Lesen einer Speicherstelle als Ergebnis erhält, ist nicht davon abhängig, was vorher an einer anderen Adresse gespeichert wurde. Für alle $m \in \text{Mem}$, $a \in \text{Adr}$, $a' \in \text{Adr}$ mit $a' \neq a$ und $v' \in \text{Val}$ gilt:

$$\text{memread}(\text{memwrite}(m, a', v'), a) = \text{memread}(m, a) \tag{9.2}$$

So wie wir oben memread und memwrite definiert haben, gilt diese Gleichung tatsächlich.

Es gibt aber Speicher, bei denen diese zunächst plausibel aussehende Bedingung *nicht* erfüllt ist, z. B. sogenannte Caches (bzw. allgemeiner Assoziativspeicher). Für eine adäquate Formalisierung des Schreibens in einen solchen Speicher müsste man also die Defintion von memwrite ändern!

Wir können hier nur kurz andeuten, was es z. B. mit Caches auf sich hat. Spinnen wir die Analogie zu Adressen auf Briefen noch etwas weiter: Typischerweise ist auch der Name des Empfängers angegeben. So etwas wie „Thomas Worsch“ ist aber nicht eine weitere Spezifizierung des physikalischen Ortes innerhalb von „Fasanengarten 5, 76131 Karlsruhe“, an den der Brief transportiert werden soll, sondern eine „inhaltliche“ Beschreibung. Ähnliches gibt es auch bei Speichern. Auch sie haben eine beschränkte Größe, aber im Extremfall wird für die Adressierung der gespeicherten Wörter sogar überhaupt keine Angabe über den physikalischen

Ort gemacht, sondern nur eine Angabe über den Inhalt. Und dann kann Speichern eines Wertes dazu führen, dass ein anderer Wert aus dem Speicher entfernt wird.

Zum Schluss noch kurz eine Anmerkung zur Frage, wozu Gleichungen wie 9.1 und 9.2 gut sein können. Diese Gleichungen sagen ja etwas darüber aus, „wie sich Speicher verhalten soll“. Und daher kann man sie als Teile der *Spezifikation* dessen auffassen, was Speicher überhaupt sein soll. Das wäre ein einfaches Beispiel für etwas, was Sie unter Umständen später in Ihrem Studium als algebraische Spezifikation von Datentypen kennen lernen werden.

Wenn Sie nicht der Spezifizierer sondern der Implementierer von Speicher sind, dann liefern Ihnen Gleichungen wie oben Beispiele für Testfälle. Testen kann nicht beweisen, dass eine Implementierung korrekt ist, sondern „nur“, dass sie falsch ist. Aber das ist auch schon mehr als hilfreich.

9.4 AUSBLICK

- In Vorlesungen über Technische Informatik werden Sie lernen, wie z. B. Caches aufgebaut sind.
- In Vorlesungen über Systemarchitektur und Prozessorarchitektur werden Sie lernen wozu und wie Caches und Assoziativspeicher in modernen Prozessoren eingesetzt werden. Dazu gehört dann auch die Frage, was man tut, wenn mehrere Prozessoren auf einem Chip integriert sind und gemeinsam einen Cache nutzen sollen.
- Der Aufsatz *What Every Programmer Should Know About Memory* von Ulrich Drepper enthält auf 114 Seiten viele Informationen, unter anderem einen 24-seitigen Abschnitt über Caches. Die Arbeit steht online unter der URL `http://people.redhat.com/drepper/cpumemory.pdf` (16.11.15) zur Verfügung.