Universität Konstanz

Rechnersysteme und -netze
Kapitel 0 - Einführung

Bastian Goldlücke
Universität Konstanz

Inhalt

Rechnersysteme

Beschreibung und Interaktion der verschiedenen Schichten in der Hierarchie vom Anwendungsprogramm (Programmcode in einer Hochsprache) zur Hardware (Transistor).

Rechnernetze

Beschreibung und Interaktion der verschiedenen Schichten in der Hierarchie von der Anwendungskommunikation (Browser/Webserver) zur Hardware (Netzwerkkabel/Funkverbindung).

Die beiden Vorlesungsteile sind relativ unabhängig voneinander, der Abschnitt über Rechnernetze recht kurz (Start erst nächstes Jahr).

Rechnersysteme: Einführung

Rechnersysteme

Beschreibung und Interaktion der verschiedenen Schichten in der Hierarchie vom Anwendungsprogramm (Programmcode in einer Hochsprache) zur Hardware (Transistor).

Wir starten mit einem sehr kurzen geschichtlichen Abriß zur Einstimmung,
sowie einem Überblick über den Inhalt der Vorlesung.

Am Anfang war die Zahl ...

Primitive Zählhilfen - bereits vor 20’000 Jahren in der Cro-Magnon-Zeit.

Knochen mit Kerben, erkennbare Fünferbündelung.

Wolfsknochen, Alter 25 – 30’000 Jahre, älteste bekannte Zahlendarstellung.

Kerbholz (bis weit ins 19. Jhdt verbreitet) als kaum fälschbarer “Schuldschein”.

Zählzeichen

1	10	100	1.000	10.000	100.000	1.000.000
Einfacher Strich	Rinds-gespann	Seilschlinge	Wasserlilie	Finger	Kaulquappe oder Frosch	Heh (altägyptischer Gott der Unendlichkeit)

Ägyptische Zahlschrift, bezeugt ab ca. 3000 v. Chr.

I = 1 X = 10 C = 100

V = 5 L = 50 D = 500

M = 1000 MMLI = 2051

XVII = 17 XLIX = 49

Römische Zahlschrift

https://meinstein.ch/math/geschichte-der-mathematik/

Die ersten Taschenrechner

Römischer Handabakus

Chinesischer Suanpan

- Benutzen zumindest implizit die Idee des Stellenwertsystems
- Man kann mit Methoden des schriftlichen Rechnens neben Addieren und Subtrahieren auch Multiplizieren, Dividieren, und sogar Wurzeln ziehen.

Die mechanische Ära

Einfache mechanische Rechengehilfen:

- “Rechenuhr” zum Addieren und Subtrahieren [Wilhelm Schickard 1623]
- Addition und Subtraktion durch Räderwerk [Blaise Pascal 1642]
- Addition, Subtraktion, Multiplikation, Division [Gottfried Wilhelm Leibniz 1685]

www.hnf.de

www.crossingselves.ch

Die mechanische Ära

Die ersten programmierbaren Rechenwerke

- Difference Engine: polynomiale Funktionen [Charles Babbage 1822]
- Analytical Engine: freie Programmierbarkeit [Charles Babbage 1837]
- Erstes Programm: Berechnung der Bernoulli-Zahlen mit der Analytical Engine [Augusta Ada Byron King, Countess of Lovelace 1843]

Teil der (nie ganz gebauten) Analytical Engine

Lochkarte zur Programmierung

Jaquard-Webstuhl (“programmierbar”)

Elektromechanische Ära

Rechner basierend auf Relais und elektrischen Zählrädern:

- BOMBE: Entschlüsselung von ENIGMA-Codes [Alan Turing 1939]
- Complex Number Computer [George Stibitz & Samuel Williams 1940]
- Digitalrechner Z3: freie Programmierbarkeit [Konrad Ernst Otto Zuse 1941]
- Mark I/Automatic Sequence Controlled Calculator [Howard H. Aiken 1944]

www.pinterest.com

Elektrische Ära 1

Rechner basierend auf Vakuumröhren:

- ABC: Lösung linearer Gleichungssysteme [John Atanasoff & Clifford Berry 1939]
- COLOSSUS: Entschlüsselung der Lorenz-Chiffre [Tommy Flowers et al. 1943]
- ENIAC (Electronic Numerical Integrator and Computer)
  [J. Presper Eckert & John W. Mauchly 1946]
- EDSAC (Electronic Delay Storage Automatic Calculator): speicherprogrammierbar
  [John von Neumann 1945, Maurice V. Wilkes *et al.* 1949]

www.wikipedia.org

www.hnf.de

Elektrische Ära 2

Rechner basierend auf Transistoren:

- Moderne Rechner basieren auf Transistoren [William B. Shockley *et al.* 1947]
- TRansistorized Airborne DIgital Computer (TRADIC) [Bell Labs 1955]
- Transistorized eXperimental computer TX-0/TX-2 [MIT Laboratory 1955/1958]
- Programmed Data Processor 1 (PDP-1, erster Minicomputer) [DEC 1959]
  Erstes Computerspiel: *Spacewar!* [MIT, 1962]
- Transistor-Transistor-Logik (TTL) [James E. Buie 1961]
- PDP-8: OMNIBUS, erster breiter verkaufter Rechner [DEC 1965]
- PDP-11: 16-Bit, Interrupts, Massenproduktion [DEC 1970]

www.hnf.de

www.pdp-11.nl

Elektrische Ära 3

Integrierte Schaltungen (integrated circuits, ICs):

- Elektronische Transistorschaltungen auf dünnen Plättchen aus Halbleitermaterial
- Halbleiterverstärker (5 Transistoren) [Werner Jacobi 1949]
- Flipflop, US-Patent für den ersten integrierten Schaltkreis [Jack Kilby 1958]
- Photolithographisches Verfahren [Robert Noyce/Fairchild Semiconductor 1959]
- Personalcomputer/Mikrocomputer [ab ca. 1976]
  Hochintegrierte Schaltungen (Very Large Scale Integration, VLSI) [ab 1980]

www.computerhistory.org

www.macworld.com

www.oldcomputers.net

- Moderne Rechner besitzen Milliarden von Transistoren
  auf Siliziumchips von wenigen hundert Quadratmillimetern Größe.

Zukunft: quantenmechanische Ära?

Today we published the results of this quantum supremacy experiment in the Nature article, "Quantum Supremacy Using a Programmable Superconducting Processor". We developed a new 54-qubit processor, named "Sycamore", that is comprised of fast, high-fidelity quantum logic gates in order to perform the benchmark testing. Our machine performed the target computation in 200 seconds, and from measurements in our experiment we determined that it would take the world's fastest supercomputer 10,000 years to produce a similar output.

Left: Artist's rendition of the Sycamore processor mounted in the cryostat. (Forest Stearns, Google AI Quantum Artist in Residence) **Right:** Photograph of the Sycamore processor. (Erik Lucero, Research Scientist and Lead Production Quantum Hardware)

`https://ai.googleblog.com/2019/10/quantum-supremacy-using-programmable.html`

Rechner: Anwendungsprogramme und Spiele

Was passiert eigentlich in einem Rechner?

- Rechner (Computer) sind heutzutage sehr bequem zu bedienen, weil sie über eine erstaunliche Rechenleistung verfügen und es sehr viele Programme für eine Vielzahl von Anwendungen gibt.

- Das ist sehr angenehm, hat aber den Nachteil, daß nur noch wenige Experten verstehen, was in einem Rechner eigentlich genau vor sich geht.

- Wir werden uns anschauen, wie ein Rechner aufgebaut ist und wie er Programme ausführt: **Rechnersysteme**, und wie mehrere Rechner miteinander kommunizieren: **Rechnernetze**.

Als Beispiel wir beginnen mit einer Idee für ein sehr einfaches Spiel:

Ein Spieler kann einen Schläger (bat) horizontal bewegen und muß verhindern, daß ein sich bewegender Ball den Bildschirm nach unten verläßt. Jedes Treffen des Balls gibt einen Punkt.

Anwendungs-/Systementwurf

- Ausgangspunkt ist eine Idee für ein Anwendungsprogramm oder ein Spiel.
- Zur Umsetzung dieser Idee muß man nicht auf die unterste Ebene der Rechnerarchitektur hinabsteigen, sondern setzt an auf der Ebene der
  - **höheren Programmiersprachen** (higher-level programming languages)
    Programmiersprachen mit höherer Abstraktion und Komplexität, die es Menschen leichter machen, Programme zu schreiben.
  - **Betriebssysteme** (operating systems)
    Sammlung von Programmen, die die Systemressourcen eines Rechners verwalten, wie z.B. Arbeitsspeicher, Festplatten, Ein-/Ausgabegeräte etc.

Hochsprachen und Betriebssysteme

Höhere Programmiersprache (Jack)

Beispiel des sehr einfachen Spiels:

```
class Bat {                          /** A graphic bat for a pong game */
    field int x, y;
    field int width, height;

    method void draw (boolean color) {
        do Screen.setColor(color);
        do Screen.drawRectangle(x,y,x+width,y+height);
        return;
    }
}
```

- Höhere Programmiersprachen sind für Menschen recht gut verständlich.
  Sie verstecken die Komplexität der eigentlichen Programmausführung.
- Um Systemresourcen zu nutzen, werden Betriebssystemfunktionen aufgerufen.
  Hier: Funktionen zum Zeichnen auf dem Bildschirm (`Screen.`...).

Betriebssystemebene (JackOS)

Typische Betriebssystemfunktion: Bildschirmdarstellung

```
class Screen {
    static boolean currentColor;            /** the current color */
    function void drawRectangle (int x0, int y0, int x1, int y1) {
        var int x, y;
        let x = x0;
        while (x < x1) {
            let y = y0;
            while (y < y1) {
                do Screen.drawPixel(x,y);
                let y = y+1;
            }
            let x = x+1;
        }
    }
}
```

- Die Betriebssystemfunktionen sind (zum größten Teil) ebenfalls in einer Hochsprache geschrieben.

Virtuelle Maschinen und Übersetzer

- Problem: Ein Rechner kann Programme, die in einer höheren Programmiersprache geschrieben sind, nicht direkt ausführen.

  Ein Rechner “versteht” höhere Programmiersprachen nicht direkt.

- Höhere Programmiersprachen abstrahieren von Sprachen tieferer Ebenen. Sie müssen daher in Sprachen tieferer Ebenen übersetzt werden. Dies geschieht durch spezielle Programme: **Übersetzer** (compiler).

Höhere Programmiersprache (Jack)

Beispiel des sehr einfachen Spiels:

```
/** Moves the bat one step (4 pixels) to the right. */
    method void moveR() {
        do draw(false);             /** erase bat at current location */
        let x = x + 4;              /** change bat's X-location */
        /** but don't go beyond the screen's right border */
        if ((x + width) > 511) {
            let x = 511 - width;
        }
        do draw(true);              /** re-draw bat in new location */
        return;
    }
```

- Wir betrachten beispielhaft die Übersetzung des rot gezeigten Ausdrucks, der aus einer Addition und einem Vergleich besteht.
- Ziel der Übersetzung ist die Sprache einer **virtuellen Maschine**.

Übersetzung in Sprache einer virtuellen Maschine

Beispiel des sehr einfachen Spiels:

- Problem: Ein Rechner kann Programme, die der Sprache der virtuellen Maschine geschrieben sind, nicht direkt ausführen.

- Auch die virtuelle Maschine abstrahiert noch von Sprachen tieferer Ebenen. Sie muß daher ebenfalls noch in Sprachen tieferer Ebenen übersetzt werden. Vorteil: **Maschinenunabhängigkeit, leichtere Übersetzbarkeit**

Stapelbasierte Virtuelle Maschine

```
// Jack source code
if ((x + width) > 511) {
   let x = 511 - width;
}
```

```
// VM code
   push x                // s1
   push width            // s2
   add                   // s3
   push 511              // s4
   gt                    // s5
   if-goto L1            // s6
   goto L2               // s7
label L1
   push 511              // s8
   push width            // s9
   sub                   // s10
   pop x                 // s11
label L2
...
```

Speicher (vorher)

	...
x	485
	...
width	40
	...

Speicher (nachher)

	...
x	471
	...
width	40
	...

Stapel (stack; sp: stack pointer)

Vorteil virtueller Maschinen

Assembler und Maschinensprache

- Die **Assembler-Sprache** ist einer VM-Sprache ähnlich, aber funktioniert für einen konkreten Rechner anstatt nur die virtuelle Maschine.
- Sie ist die tiefste Sprachebene, auf der noch von Menschen programmiert wird (aber nur dann, wenn es um sehr laufzeitkritische Funktionen geht).

Assembler und Maschinensprache

Die Maschinensprache besteht nicht mehr aus Symbolen, sondern aus Binärzahlen. Sie ist daher von einem Menschen nur mit großen Schwierigkeiten zu lesen. Deshalb abstrahieren höhere Sprachen von ihr.

Virtuelle Maschine

```
...
   push x
   push width
   add
   push 511
   gt
   if-goto L1
   goto L2
L1:
   push 511
   push width
   sub
   pop x
L2:
...
```

VM-Übersetzer

Assemblersprache

```
// push 511
@511
D = A    // D = 511
@SP
A = M    // A = SP
M = D    // *SP = D
@SP
M = M+1  // SP++
```

Assembler

Maschinensprache

```
0000000000000000
1111110111001000
```

- Die **Maschinensprache** kann von einem Rechner direkt ausgeführt werden!

Semantik der Maschinensprache

- Die **Maschinensprache** ist die Schnittstelle von der Software (Programme) zur Hardware (Geräte). Durch sie wird der Rechner gesteuert.

Semantik der Maschinensprache

- Benötigt: **Hardware-Plattform**, die diese Semantik realisiert.
- Die Hardware-Plattform sollte in der Lage sein,
  - die Befehle zu interpretieren (gemäß obiger Semantik)
  - und auszuführen (entsprechende Operationen durchführen).

Hack-Architektur

- Rechnerarchitektur aus [Noam Nisan & Shimon Schocken 2008] für die Hack-Plattform (pc: program counter, d.h. Befehlszähler).

Hack-Architektur: CPU

- Nur **Daten- und Adreßpfade** sind gezeigt (d.h., Verbindungen, die Daten und Adressen transportieren), nicht jedoch die **Steuerlogik**, mit Ausnahme der Ein- und Ausgaben von Steuerbits, die durch (c) markiert sind.

Implementierung der Hardware-Plattform

Implementierung der Hardware-Plattform

- Eine Hardware-Plattform wird implementiert
  durch eine Menge zusammengeschalteter Bausteine (Chips).

- Bausteine (Chips) bestehen aus einfacheren Bausteinen/Schaltungselementen,
  bis hinunter auf die Ebene einzelner (Logik-)Gatter.

- Ein **(Logik-)Gatter** ist ein elektronischer Baustein,
  der eine bestimmte Boolesche (logische) Funktion implementiert.

- (Logik-)Gatter bilden die Grundlage der **Gatter-** oder **Schaltungslogik**.
  Diese zerfällt in zwei Kernbereiche:

  - **Kombinatorische Logik** (zeit-/zustandsunabhängig)

    Z.B. zur Implementierung des Rechenwerks eines Prozessors
    (arithmetisch-logische Einheit, arithmetic logic unit, ALU).

  - **Sequentielle Logik** (zeit-/zustandsabhängig)

    Z.B. zur Implementierung der Prozessorregister (CPU registers)
    und des Speichers (random access memory, RAM).

Gatter-/Schaltungslogik

Schnittstelle (interface)

a	b	$a \oplus b$
0	0	0
1	0	1
0	1	1
1	1	0

Implementierung der Schnittstelle

$$a \oplus b \equiv (a \wedge \neg b) \vee (\neg a \wedge b)$$

- Jeder Baustein (Chip) und jedes Gatter hat eine abstrakte Schnittstelle, die angibt, **was** er/es tut, und eine Implementierung, die angibt, **wie** er/es dies tut.
- Hier gezeigt: Schnittstelle eines XOR-Gatters (Wahrheitstafel) und seine Implementierung durch zusammengeschaltete andere Gatter.

Hardware-Beschreibungssprache

Implementierung

HDL-Beschreibung

```
CHIP Xor {
   IN a, b;
   OUT out;
   PARTS:
   Not(in=a,out=Nota);
   Not(in=b,out=Notb);
   And(a=a,b=Notb,out=w1);
   And(a=Nota,b=b,out=w2);
   Or(a=w1,b=w2,out=out);
}
```

- Die Implementierung eines Bausteins (Chips) oder eines Gatters durch andere Gatter wird in einer **Hardware-Beschreibungssprache** (hardware description language, HDL) angegeben.
  Dies ermöglicht u.a. eine automatische Prüfung einer Implementierung.
- Bestimmte Gatter (oft NAND, NOR und NOT) sind vorgegeben.

Implementierung der (Logik-)Gatter

Implementierung von Gattern durch Transistoren

a	b	$a \mid b$
0	0	1
1	0	1
0	1	1
1	1	0

- NAND-Gatter als Schaltzeichen mit Wahrheitstabelle:
  Ein NAND-Gatter berechnet die negierte Konjunktion.
- Rechts: eine mögliche Implementierung eines NAND-Gatters durch 4 Feldeffekttransistoren (MOSFETs) in CMOS-Technik.

Rechnersysteme: Vollständige Hierarchie

Rechnersysteme: Plan der Vorlesung

- Wir werden uns in dieser Hierarchie von unten nach oben vorarbeiten, d.h., sozusagen einen Rechner von Grund auf aufbauen.

Rechnernetze: Einführung

Rechnernetze

Beschreibung und Interaktion der verschiedenen Schichten in der Hierarchie von der Anwendungskommunikation (Browser/Webserver) zur Hardware (Netzwerkkabel/Funk-verbindung).

Rechnernetze: Stark vereinfachte Einführung

Die Sendung mit der Maus - Wie funktioniert das Internet

https://www.youtube.com/watch?v=Xs7T1QAe-8E

Rechnernetze: Webseitenaufruf

Einfaches Beispiel: Wir wollen die Webseite `www.uni-konstanz.de` aufrufen.

(Achtung: im folgenden stark vereinfachtes Modell!)

HTML: HyperText Markup Language (Beschreibungssprache hauptsächlich für Webseiten)

Rechnernetze: Webseitenaufruf

Einfaches Beispiel: Wir wollen die Webseite `www.uni-konstanz.de` aufrufen.

(Achtung: im folgenden stark vereinfachtes Modell!)

HTTP: HyperText Transfer Protocol (Übertragungsprotokoll hauptsächlich für Webseiten)

Rechnernetze: Webseitenaufruf

- Die Anfrage an den Universitäts-Server wird in mehreren Schritten bearbeitet und durch mehrere Abstraktionsschichten hindurchgeleitet.
- Jede dieser Schichten hat bestimmte, eng begrenzte Aufgaben, und auf jeder Schicht gibt es Netzwerkprotokolle mit klaren Schnittstellen.

2.bp.blogspot.com

Rechnernetze: Webseitenaufruf

```
To:      ???
www.uni-konstanz.de
```

- Der Internetbrowser Mozilla Firefox ist unser HTTP-Client.
  Dies ist die **Anwendungsschicht** (application layer).
- Die HTTP-Anfrage wird z.B. durch Eintippen von `www.uni-konstanz.de` in die Adreßzeile des Firefox-Browsers gestellt.
- Der Firefox-Browser kapselt die HTTP-Anfrage („steckt sie in einen Briefumschlag“) und gibt sie an die **Transportschicht** (transport layer) weiter.
- Aber: Die Transportschicht versteht die Adresse `www.uni-konstanz.de` nicht.

  Adressen auf der Transportschicht sind **IP-Adressen** (IP: internet protocol).
  Diese bestehen aus vier (IPv4) oder acht (IPv6) Zahlen, z.B. `134.34.240.80`.

Rechnernetze: Domain Name System (DNS)

- Der HTTP-Client fragt einen DNS-Server (DNS: Domain Name System, gewissermaßen ein „Telefonbuch“ oder ein „Adreßverzeichnis“) nach dem **Domain-Namen** `www.uni-konstanz.de`.
- Der DNS-Server übersetzt `www.uni-konstanz.de` in eine **IP-Adresse**, hier: `134.24.240.80` (IPv4) (IPv6 nicht verfügbar).
- An diese Adresse schickt nun der HTTP-Client (hier: Firefox-Browser) die Anfrage nach der Webseite der Universität Konstanz.
- An welchem „Briefkasten“ oder „Postfach“ (port) wird die Anfrage zugestellt?

Rechnernetze: Ports

To: 134.34.240.80:80

⟨ip-address⟩ : ⟨port⟩

- **Ports** sind wie „Briefkästen“ oder „Postfächer“ für bestimmte Dienste (services).

  Analog hat z.B. der Lehrstuhl Bioinformatik und Information Mining das Postfach 712, um ihn innerhalb der Universität Konstanz zu identifizieren.
- Insgesamt gibt es $2^{16} = 65536$ Ports, von denen die unteren $2^{10} = 1024$ (also die Ports 0 bis 1023) für standardisierte (System-)Dienste vorgesehen sind, z.B.:

File Transfer Protocol (FTP)	20 (Daten) & 21 (Steuerung)
Secure SHell (SSH)	22
Simple Mail Transfer Protocol (SMTP)	25
Domain Name System (DNS)	53
HyperText Transfer Protocol (HTTP)	80

Rechnernetze: Transmission Control Protocol (TCP)

```
From: 192.168.1.101:3482
To:   134.34.240.80:80
```

- Der HTTP-Client (Firefox) kapselt die HTTP-Anfrage und gibt sie an die **Transportschicht** weiter, die von einem **TCP-Client** verwaltet wird.
- Der TCP-Client baut die Verbindung zum Universitäts-Server auf (hier: zu Port `80` der IP-Adresse `134.34.240.80`).
- Der TCP-Client segmentiert und numeriert die zu sendenden Daten, fügt Absender- und Empfängerinformationen und einige weitere Dinge hinzu.

Rechnernetze: Internet Protocol (IP)

- Der TCP-Client übergibt die erzeugten Segmente/Pakete an das **IP-Modul** und beauftragt es, sie an die IP-Adresse `134.34.240.80` zu liefern.
- Auf dieser **Netzwerkschicht** ist eine direkte Auslieferung der Segmente/Pakete nur an Zieladressen **im gleichen Netzwerk** möglich.
- `www.uni-konstanz.de` ist nicht im gleichen Netzwerk wie Bastians Laptop. (Bastians Laptop ist in einem lokalen Netzwerk, local area network, LAN.)
- Das IP-Modul erzeugt daher ein **IP-Datagramm/IP-Paket** und sendet es an den **Standard-Netzübergang** (default gateway).

Rechnernetze: IP-Adressen/IP-Netzwerk

trival.co

- Ein **IP-Netzwerk** kann in logische Teilnetze unterteilt werden.
- Die **Teilnetzmaske** (subnet mask) wird benutzt, um festzustellen, ob die Zieladresse im gleichen Teilnetzwerk ist.
- Kein Unterschied der Adressen unter der Maske ⇒ gleiches Teilnetz.

Rechnernetze: Internet Protocol (IP)

- Der TCP-Client übergibt die erzeugten Segmente/Pakete an das **IP-Modul** und beauftragt es, sie an die IP-Adresse `134.34.240.80` zu liefern.
- Das IP-Modul erzeugt daher ein **IP-Datagramm/IP-Paket** und sendet es an den **Standard-Netzübergang** (default gateway).

TCP segment

```
(Christian)
From: 192.168.1.101:3482
To:   134.34.240.80:80
      (www.uni-konstanz.de)
```

```
(Christian)
From: 192.168.1.101
To:   134.34.240.80
      (www.uni-konstanz.de)
```

IP datagram

Rechnernetze: Internet Protocol (IP)

- Um es an den **Standard-Netzübergang** (default gateway) zu senden, muß das **IP-Datagramm** in einen **Ethernet-Datenrahmen** (Ethernet (data) frame) gepackt werden. Dies ist die **Verbindungsschicht**.
- Der Ethernet-Treiber versteht weder IP-Adressen noch Domain-Namen, sondern nur **MAC-Adressen** (MAC: Media Access Control).

IP datagram

```
      (Christian)
From: 192.168.1.101
To:   134.34.240.80
      (www.uni-konstanz.de)
```

→

```
      (Christian)
From: 00:00:5E:00:01:07
To:   00:00:8F:AE:01:EE
      (Default Gateway)
```

Ethernet frame

Rechnernetze: Ethernet

Die MAC-Adressen werden eigentlich schon vorher übertragen, nämlich beim Einrichten der Netzwerkverbindung, z.B. über DHCP (Dynamic Host Configuration Protocol).

- Der Ethernet-Treiber versteht weder IP-Adressen noch Domain-Namen, sondern nur **MAC-Adressen** (Media Access Control addresses).
- Die Übersetzung wird über sogenannte **ARP-Tabellen** vorgenommen (Address Resolution Protocol), die z.B. im Rechner oder Router abgelegt sind.
- Der **Ethernet-Datenrahmen** (Ethernet frame) gelangt nun an den Router / Netzwerkübergang.

Rechnernetze: Router

- Der Router entpackt den **Ethernet-Datenrahmen** (ethernet frame) ...

Ethernet frame

(Christian)
From: 00:00:5E:00:01:07

To: 00:00:8F:AE:01:EE
(Default Gateway)

(Christian)
From: 192.168.1.101

To: 134.34.240.80
(www.uni-konstanz.de)

IP datagram

- ... und das **IP-Datagramm** bis zum **TCP-Segment**.

IP datagram

(Christian)
From: 192.168.1.101

To: 134.34.240.80
(www.uni-konstanz.de)

(Christian)
From: 192.168.1.101:3482

To: 134.34.240.80:80
(www.uni-konstanz.de)

TCP segment

Rechnernetze: Router

- Die Ziel-IP-Adresse des IP-Datagramms wird vom Router benutzt, um das nächste Zwischenziel auf dem Weg zum Web-Server zu bestimmen.
- Hierbei helfen **Wegetabellen** (routing table, routing information base). Wegetabellen enthalten Angaben zu möglichen Wegen, zu “optimalen” Wegen, zum Status, zur Metrik (z.B. Anzahl “Hops”), und zum Alter.
- Durch Wegetabellen werden Ziel-IP-Adressen mit Richtungsangaben in Form des Folgerouters und der zu verwendenden Schnittstelle verknüpft.

  Wegeinformationen können statisch sein oder dynamisch gelernt werden.

Rechnernetze: Network Address Translation (NAT)

- Im TCP-Segment wird die **Quell-IP-Adresse** ersetzt durch die **Router-IP-Adresse** und einen spezifischen **Port**.

- In die **NAT-Tabelle** (NAT: Network Address Translation) des Routers wird die entsprechende Zuordnung eingetragen:

$$192.168.1.101:3482 \quad \Leftrightarrow \quad 65.96.14.76:5621$$

Rechnernetze: Network Address Translation (NAT)

NAT Translation Table

	Local IP Address	Source Port #		Internet IP Address	Source Port #
process X, Host A →	192.168.1.101	54,847	=	65.96.14.76	1
Host B →	192.168.1.103	24,123	=	65.96.14.76	2
process Y, Host A →	192.168.1.101	42,156	=	65.96.14.76	3
Host C →	192.168.1.102	33,543	=	65.96.14.76	4

www.microchipdeveloper.com

Rechnernetze: Network Address Translation (NAT)

- Das modifizierte **TCP-Segment** wird in ein **IP-Datagramm** umgewandelt:

- Das **IP-Datagramm** wird in einen **Ethernet-Datenrahmen** umgewandelt:

Rechnernetze: Network Address Translation (NAT)

- Am nächsten Router wiederholt sich der Ablauf:
  - Der Ethernet-Datenrahmen wird zum IP-Datagramm und dieses zum TCP-Segment ausgepackt.
  - Das TCP-Segment bekommt eine neue Quelladresse für Antworten.
  - Das modifizierte TCP-Segment wird wieder zum IP-Datagramm und dieses zum Ethernet-Datenrahmen eingepackt.
  - Der Ethernet-Datenrahmen wird zum nächsten Router geschickt...

  ... bis schließlich der Zielrechner (hier: Universitätsserver) erreicht wird.

Rechnernetze: Server

- Der Server entpackt den **Ethernet-Datenrahmen** (ethernet frame) ...

- ... und das **IP-Datagramm** bis zum **TCP-Segment**.

Datenpaketübertragung im Internet

- Die einzelnen TCP-Segmente, in die die zu übertragenden Daten aufgeteilt werden, können auf verschiedenen Wegen übertragen werden.

Rechnernetze: Server

- Die einzelnen **TCP-Segmente** werden wieder zusammengesetzt:

- Dem **HTTP-Server** auf dem Universiätsserver liegt nun die Anfrage nach der (Haupt-)Webseite der Universität Konstanz vor.
- Die Daten dieser Webseite werden nun auf gleiche Weise zurückgeschickt:

  Aufteilung in TCP-Segmente etc.

Rechnernetze: Zusammenfassung

- Der Bereich der Rechnernetze beschäftigt sich mit der abstrakten Beschreibung der Kommunikation zwischen Rechnern.
- Die Kommunikation wird üblicherweise durch ein Schichtenmodell beschrieben, z.B. das TCP/IP-5-Schichten-Modell oder das ISO/OSI-7-Schichten-Modell.