Universität Konstanz
Fachbereich Informatik & Informationswissenschaft
Stephan Heidinger

Rechnersysteme
Wintersemester 2010/2011
28. Januar 2012

Find any errors? Please send them back, I want to keep them!

Vorlesung 0 - Hardware, Einstieg

- viel blabla...
- **Z3** erste elektrische Rechenmaschine 1941, Representationen von Zahlen im Binärsystem (Relais)
- **Additionssysteme** Einzelne Symbole haben einen Wert, die Werte werden einfach addiert, etwa *Römische Zahlen* oder *Unärsystem*. Vorteile: Addition sehr einfach, Nachteile: andere arithmetische Operationen schwer, unübersichtlich
- **Stellenwertsysteme** Jedes Symbol hat einen Wert, jede Stelle hat einen Wert, der Wert der gesamten Zahl setzt sich zusammen aus der Summe der Werte jedes Symboles multipliziert mit dem Wert der Stelle (Basis hoch Stellennummer) Vorteile: Einfache arithmetische Operationen möglich.
- **Systeme zur Darstellung des Binärsystems:** Binärsystem schnell unübersichtlich
  Oktal: Basis $8 = 2^3$, eher veraltet, früher gebräuchlich, heute noch für etwa Rechte in UNIX-Systemen
  Dezimal: Basis 10, Darstellung in menschengewohnter Form
  Hexadezimal: Basis $16 = 2^4$, kompakte Darstellung, 2 Stellen = 1 Byte
- Umrechnungen:

Quell	Ziel	Algorithmus
bin	oct	3er Gruppierung
bin	hex	4er Gruppierung
oct	hex	erst zu bin, dann wie bin→hex
hex	oct	erst zu bin, dann wie bin→oct
dec	base n	Teilen mit Rest durch n bis Rest 0
base n	dec	Hornerschema
base m	base n	Erst zu dec, dann wie dec→base n

- **Zahlensysteme:**
  metrisch Basis 10 (spezielle Symbole für Basis 2), SI-Einheiten, konsistent, Umrechnung, basiert auf Vielfachen von Basiseinheiten
  imperial Basiert auf (traditionellen) Alltagswerten (Fuß, Elle, Stein,...), schwer umzurechnen, wegen variierender Basis
- powers of 2

2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536

Vorlesung 1 - Hardware

- Geschwindigkeitssteigerung

  processor speed: mehr Speed im Prozessor ⇒ mehr Gesamtspeed

  component size: kleinere Komponenten ⇒ kürzere Wege ⇒ schneller

  memory size: memory speed nimmt kaum zu

  optimization of structure: pipelining (fetch, decode, execute gleichzeitig), speculative execution, parallel/redundant units (>1 apu, multiple cores, multithreading,...)

- **Moore's Law** *The number of transistors on given chip area doubles every 18 months.*

- **Computer Architecture** Von-Neumann-Rechner:

  CPU Steuer- und Rechenwerk, Steuerung der Befehlsabfolge + Rechenoperationen

  Speicher Speichert Daten und Instruktionen (Text)

  I/O in-/output + interfaces

  Nachteil: Von-Neumann-Flaschenhals: Daten und Text über einen Bus, CPU schneller als Bus

- verschiedenes:

  Halbleiter: auf Silizium basierend, abhängig von Temperatur Leiter oder Isolator

  Transistor: Schalten und Verstärken elektrischer Signale

  Kondensator: speichert elektrische Energie

  Integrierter Schaltkreis: elektronische Schaltung auf einem Halbleiter

- Speicherhirarchie
  - register
  - cache
  - main memory
  - solid state drive
  - electronic disk
  - magnetic disk
  - optical disk
  - magnetic tape

- **Betriebssystem:** Management von Prozessen, Abstraktion von Hardware (Speichermanagement, Interruptmanagement, Dateisystemmanagement, Gerätetreiber, System Calls als Abstraktion, ...)

- **Virtuelle Maschine:** ein aus Software bestehender Rechner, als Betriebssystem simuliert sie Hardware, als Laufzeitumgebung virtuellen Prozessor

- **Kernel:**

  kernel space: Speicherbereich des Kernels, Prozesse in diesem Bereich haben volle Sichtbarkeit und vollen Zugriff auf Hardware

  user space: restlicher Speicherbereich, kein direkter Zugriff auf Resourcen

  monolithischer Kernel: alle Komponenten als ein Prozess im kernel space, Volle Sichtbarkeit, Fehler führt zu system failure (Linux/UNIX, DOS, Windows (ohne NT), OS/2)

Mkcrokernel: Nur wichtigste Prozesse (Prozesskommunikation, Speicher-/Prozessmanagement) im kernel space, Fehler lässt nur einzelne Komponenten abstürzen, hoher Aufwand für Synchronisation (minix, Mach, Hurd, QNX)

Hybridkernel: Mischung aus monolitisch und mikro (Windows NT, XNU/Darwin/OSx)

Vorlesung 2 - File System, Operating System, Process Management

- **File System:** Abstrahiert I/O Geräte, Dateien als Geräte unabhängige Entitäten, organisiert Daten in Dateien und Vereichnissen, sicher Integrität und Zugriff, managed Speicher

- **File System**
  Main Memory/RAM (random access memory), CPU can access directly, volatile
  Secondary Storage, non-volatile, large, slower

k	kilo	10^3	1 000	Ki	kibi	2^{10}	1 024
M	mega	10^6	1 000 000	Mi	mebi	2^{20}	1 048 576
G	giga	10^9	1 000 000 000	Gi	gibi	2^{30}	1 073 741 824
T	tera	10^{12}	1 000 000 000 000	Ti	tebi	2^{40}	1 099 511 627 776

- **read/write-delay:** seek time (move heads to track, wait for head to settle), rotational delay (wait for sector to move under head), transfer time (read/write time)
  access time = seek time + rotational delay + transfer time
  $\text{transfer time} = \frac{\text{transferrate}}{\text{blocksize}}$
  capacity = cylinders · heads · sectors · sector size
  $\text{cylinders} = \frac{\text{tracks}}{\text{side}}$

- **File System Requirements:** Block allocation/management/caching, Name/file mapping (list, open, close), File offset/block mapping (read, write, seek), Access Control

- **SSD/Solid State Drive:** Halbleiterspeicher, keine mechanischen Teile ⇒ leiser, schneller, robuster, energieeffizienter, ABER: teuer, weniger Kapazität, schwierige Herstellung

- **MBR/Master Boot Record:** 1 Block (512B) Bootloader und Partitionstabelle

- **DOS/FAT:** (file allocation table) Drive Letters, Special Devices, Sequential Files
  Gleich große Cluster, FAT hat für jeden Eintrag einen Cluster Disk Layout:
  - boot block
  - file allocation tables
  - root directory area
  - data area (+directory hirarchy)

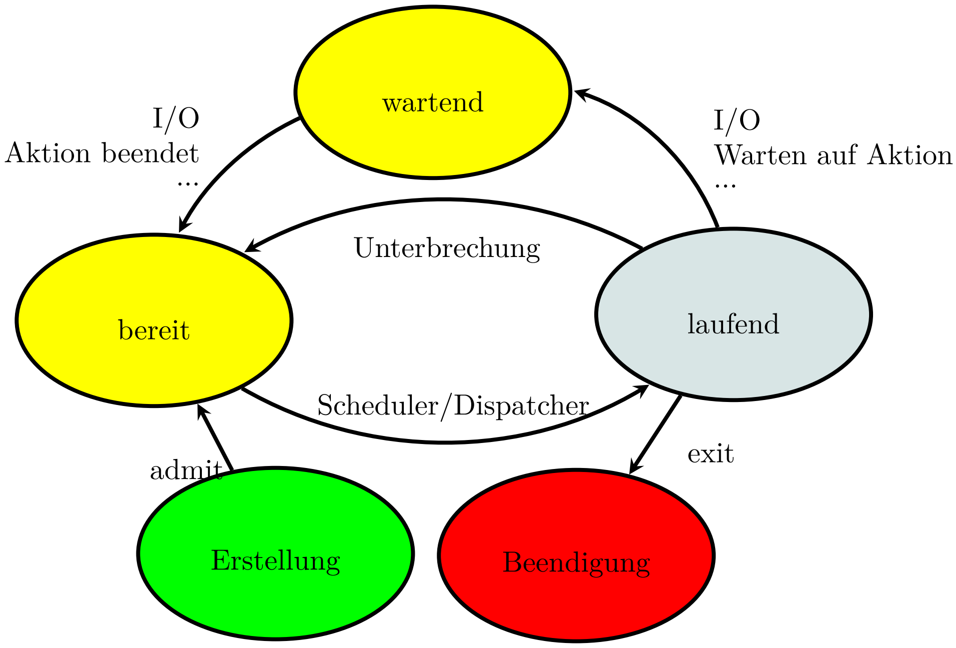
  VFAT: Long File Names

- **Fragmentierung:** Verstreutes Speichern zusammengehörender Daten, Verhindern durch: Defragmentieren, grössere Blöcke, preallokation von Blöcken, spätes Festlegen von Blöcken

- **Inodes**: UNIX-Idee, Baumstruktur

- **Betriebssystem:** wimp: <u>w</u>indow, <u>i</u>con, <u>m</u>enu, <u>p</u>ointer

- **Processes:** programm vs process



PCB (process control block), contains necessary information to: interrupt/continue, remember I/Os and file usage of process, remember privileges (Eintrag in Prozesstabelle, Status, program counter, stack pointer, singals, scheduling info, . . .
forking of processes

- **Scheduler:** allow processes to use cpu, while other processes waiting, Gannt-charts, models:

  FCFS: First Come, First Serve

  (P)SJF: (Preemtive) Shortest Job First

  PHPF: Preemtive Highest Priority First

  preemptive (interrupt running task for one with higher priority)

  EWMA: Exponentially Weighted Moving Average, tries to calculate CPU burst times via past bursts, older bursts are less weighted

  round robin: gewährt Prozessen in einem Ringverfahren jeweils kurze Zeitschlitze

- **Protection:** Kernel/User mode, memory protection (process cant write outside its allocated memory

Vorlesung 3 - Assembler

- **Assembler** *mnemonics* → machine language, down to the roots, incredibly fast (airbag, brake control, . . .

Opcode	Operation
add r, a, b	r $\leftarrow$ a+b
sub r, a, b	r $\leftarrow$ a-b
mul r, a, b	r $\leftarrow$ a*b
div r, a, b	r $\leftarrow$ a/b
mod r, a, b	r $\leftarrow$ a%b
neg r, a	r $\leftarrow$ -a
sign r, a	r $\leftarrow$ signof(a)$\{-1, 0, +1\}$
abs r, a	r $\leftarrow$ $\lvert a\rvert \equiv r \leftarrow a\cdot$ signof(a)
mov r, a	r $\leftarrow$ a
set const, cexpr	$\leftarrow$ define constant *const*
and r, a, b	r $\leftarrow$ $a \wedge b$
or r, a, b	r $\leftarrow$ $a \vee b$
xor r, a, b	r $\leftarrow$ $a \oplus b$
not r, a	r $\leftarrow$ $\neg a$ (logical)
cmnt r, a	r $\leftarrow$ $\neg a$ (bitwise)
shr r, a, b	r $\leftarrow$ $a \gg b$
shl r, a, b	r $\leftarrow$ $a \ll b$
call label	PC $\leftarrow$ label
jmp label	PC $\leftarrow$ label
brcmp OP, label, a, b	PC $\leftarrow$ label, if a OP b
brtst OP, label, a	PC $\leftarrow$ label, if a OP 0
cmp OP, r, a, b	r $\leftarrow$ (a OP b)
tst OP, r, a	r $\leftarrow$ (a OP 0)
stop a	stop if a $\neq$ 0

Operator	(OP)
LT	$<$
GT	$>$
LTEQ	$<=$
GTEQ	$>=$
EQ	$==$
NEQ	$!=$

`push` / `pop` used for stack access
Auf jeden Fall bei der Assembler-Aufgabe reinschreiben...

```
begin or start or subroutine
    .
    .
    .
    return;
end;
```

- **Programm:** ausführbare Binärdatei, Folge von Anweisungen

  Prozess: Instanz eines Programms, Programm in Ausführung

  Thread: kleinste auszuführende Einheit (im gleichen Speicherbereich wie Prozess)

  Zombie: nicht ordnungsgemäß beendeter Prozess (Eintrag in Prozesstabelle)

  Dämon: Prozess im Hintergrund, keine Benutzerinteraktion

  Verhungern: Prozess bekommt nie die Resourcen, die er benötigt, um ausgeführt zu werden

  Prozessmanager: Multiprogramming/Multitasking/IPC (interprocess communication), minimize waiting and response time, maximize CPU usage

Vorlesung 4 - Data Encoding

- common memory sizes:

4 bits	nibble
8 bits	byte
16 bits	half word, word, short, int
32 bits	word, dword, int, long
64 bits	long, long long

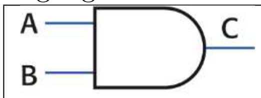
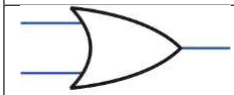
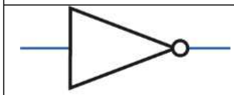
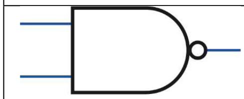
- Historic: Morse, Baudot, Telex, SIXBIT
- **ASCII** (American Standard Code for Information Interchange, memory efficient, no support for other languages
  7 bits + 1 bit checksum
  country specific versions (ISO 646)
- **ISO 8859** ASCII + extended control characters + nbsp + more printables, 15 different versions
- **UNICODE**: code points for all “usefull characters “, 17 “planes“ of 65536 code points, character = glyph (pixel-adressable display/printer)
  NFC: composed (U+00fc: ü) vs NFD: decomposed (U+0075: u + U+0308 ¨)
- **UTF-8** most common, most complex, variable length, ascii compatbile, efficient for most languages

U+0000...U+007f ($\leq$ 7 bits)	$\rightarrow$	$0xxx\ xxxx_2$ ASCII kompatibel
U+0080...U+07ff (8...11 bits)	$\rightarrow$	$110x\ xxxx_2\ 10xx\ xxx_2$
U+0800...U+ffff (12...16 bits)	$\rightarrow$	$1110\ xxxx_2\ 10xx\ xxxx_2\ 10xx\ xxxx_2$
U+0001 0000...U+0010 ffff (17...20.1 bits)	$\rightarrow$	$1111\ 0xxx_2\ 10xx\ xxx_2\ 10xx\ xxxx_2\ 10xx\ xxxx_2$

- **UTF-16** common code points: U+0000...U+D7FF, U+E000...U+FFFF direct (U+D800...U+DFFF forbidden), 16 Bit für normale Zeichen, für seltene Zeichen zwei 16 surrogate bit pairs
  for code points U+10000 and beyond, substract U+10000, result fits into 20 bit, split in halves, Upper half with U+D800 stored first, Lower half with U+DC00 stored second
  BOM start file with FEFF (big endian) or FFFE (little endian)
- **UTF-32** fixed-length encoding
- **Numbers**
  - **unsigned integers:** 0...65 535
  - **sign-magnitude:** bei Hälfte des Bereichs beginnt negativer Bereich
  - **Einerkomplement:** ein Bit zum negativieren der Zahlen
  - **Zweierkomplement:** wie Einerkomplement, nur dass neg bei -1 startet, Addition und Substraktion sind konsistent
  - **floating point** IEEE754 and IEEE854, 32 bit (single precision) or 64 bit (double precision)
    $\underbrace{S}_{1bit}\ \underbrace{EEEEEEEE}_{8bits}\underbrace{MMMMMMMMMMMMMMMMMMMMMMM}_{23bits}$
    value $= (-1)^s \cdot 1.MMMMMMMMMMMMMMMMMMMMMMM \cdot 2^{EEEEEEEE-127}$

Vorlesung 5 - Digital Logic

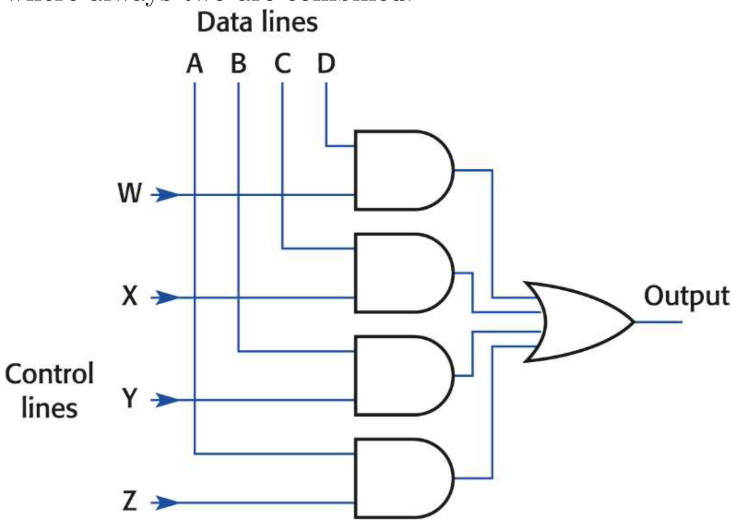
- Logic gates are atomary elements in modern circuits

 AND	A AND B	$A \wedge B$	$A \cdot B$	Konjunktion
 OR	A OR B	$A \vee B$	$A + B$	Disjunktion
 NOT	NOT A	$\neg$ A	$\overline{A}$	Negation
	A NAND A			

NOT, AND, OR are functionally complete, so are NAND and NOR

NOT A	=	A NAND A
A AND B	=	(A NAND B) NAND (A NAND B)
A OR B	=	(A NAND A) NAND (B NAND B)

Multiplexers are more complex designs using $2 \times n$ inputs, n data lines and n control lines, where always two are combined.



- **sum-of-products:** find where output is 1, connect corresponding inputs via and, connect and-groups via or; DNF
- **product-of-sums:** find where output is 0, connect inputs via and and negate groups, connect resulting groups via and; KNF
- **circuit simplification:** Use boolean laws or karnaugh maps (from one field to the next, only one value changes!) bsp-karnaugh map einfügen
- **Instruction Set Architecture:**

  Instruction Set: Menge aller von der Hardware bereitgestellten Instruktionen, Datentypen, Datenstrukturen (RISC (reduced instruction set), CISC (complete instruction set))

  Mnemonic: Lesbare Representation des Opcode

  Opcode: „Befehl" in Maschinensprache, Nummer des Maschinenbefehls, Representation durch Mnemonic, $\sum$Opcodes$\equiv$ Befehlssatz

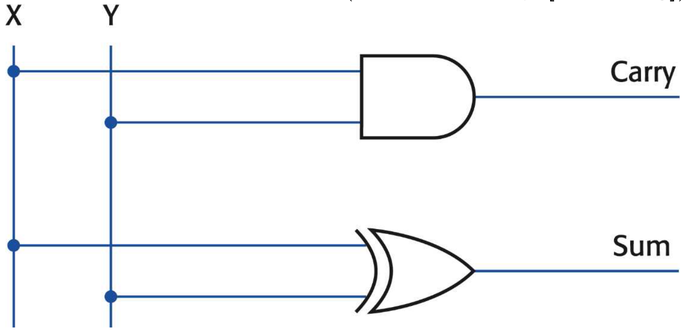
Pseudo-Opcode: kein wirklicher Opcode, wird in mehrere Operationen expandiert

heap: gesammter Speicherbereich eines Prozesses

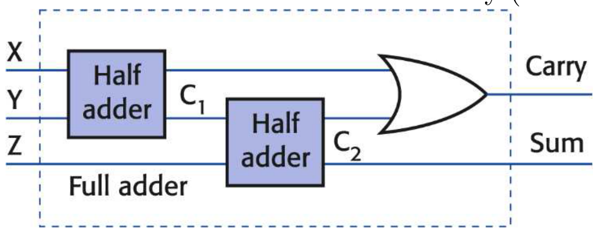
stack: Speicherbereich, LIFO, Rücksprungadresse, Daten, Parameter, Rückgabewerte, ...

Vorlesung 6 - Integer Arithmetic

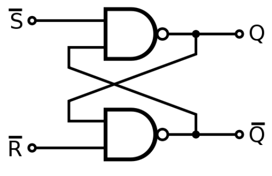
- **ALU** - Arithmetic and Logic Unit → performs binary integer arithmetic operations
- **FPU** - Floating Point Unit, seperarate processor in larger systems else done in software
- **half adder:**, two outputs (sum and carry [Übertrag])



- **full Adder:** adds two bits and a carry ($Z = C_{in}$)



- **(parallel) n-bit adder:** assembly of full-adders...
- **(parallel) n-bit substractor:** consists of n-bit adders and negation gates and adding one (like when done by hand)
- **multiplication:** could be achieved by mutliple additions, but is more practical with bit-shifting techniques (like multiplication with 10 done by hand)
- **Flip Flop:** A flipflop has two stable states, upon signal in, one state will stay, until other input fires, `NAND`s in example can also be `NOR`s



Vorlesung 7 - networking intro

- **Network:** infrastructure (hard-/software) that enables endpoins (hosts) to communicate

  Frame: data transmission or data packet with frame synchronisation (sequencing of frames)

  Repeater: amplifies & copies any electrical signal (including noise)

  Bridge: copies frames between segments, can interpret frame data and drops frames not addressed to other segments

  Router: copies packets between segments, can interpret final destination address and has routing tables to optimize packet route

Gateway: generic term, encompassing hardware (repeater, bridge, router) and software (router)

- **layering**

application	supporting network applications	FTP, HTTP, SMTP, POP, NNTP, IRC, STTP
presentation	(OSI) representation of data, de-/encryption, machine-de/independet data	
session	(OSI) Kommunikation zwischen Sitzungen von Anwendungen/Prozessen, „interhost“	
transport	host-to-host data transfer	TCP, Ports, UDP
network	path determination, logical addressing	IP, routing protocolls, IP, IPX, IPsec, Router, ARP, ICMP
data/link	physical addressing	PPP, Ethernet, IEEE 802.11, Switch, Bluetooth, Modem
physical	“bits on the wire“	cable

All **P**eople **S**eem **T**o **N**eed **D**ata **P**rocessing
Alle **P**riester **S**aufen **T**equila **N**ach **D**er **P**redigt

- **Internet principle:** “We reject kings, presidents and voting. We believe in rough consensuns and running code.“
  End-to-End principle ⇒ dumb-network, smart endpoints

- Berechnungen:
  packet size $p = n \cdot h[B] + d[B] = (n \cdot h + d)[B]$
  gross transfer rate $R = b[Mbps] \to R' = R \cdot \frac{10^6}{8}[Bps]$ (R is bandwidth)
  package rate $D = \frac{R'}{p}[packets/s]$

- **TCP/IP:**

- **packet switching:** data is split into packets, which are sent seperately, packets carry sequence numbers, network layer uses them to fragment and reassemble packages, transport layer uses them to detect and retransmit lost packages
  great for “bursty“ data
  excessive congestion: packet delay, loss of data

- **circuit switching:** i.e. telephone, direct physical link used exclusively by the communications endpoints
  great for constant bit rate
  no resource sharing

- **port number:** differentiates processes, some protocols have recommended ports (25=smtp, 22=ssh, 80=http; 21=ftp, ...)

Vorlesung 8 - Packet Switching

- **protocol:** Vereinbarung, nach der die Verbindung, Kommunikation und Datenübertragung abzulaufen hat

- **static multiplexing:**
  FDMA: frequency division multiple access, each host gets equal amount of time frame
  TDMA: time division multiple access, each host gets equal amount of frequency band

- **statistical multiplexing:** packets from diff hosts have no fixed flow pattern, past usage is monitored and used for assigning of slots

Vorlesung 9 - Access and WLAN

- **delay** $d_{nodal} = \underbrace{dproc}_{\text{some ms}} + \underbrace{d_{queuing}}_{\text{depends on congestion}} + \underbrace{d_{trans}}_{\frac{L}{R}\text{, significant for slow/long links}} + \underbrace{dprop}_{\text{wenn überhaupt ms}}$

(nodal) processing	checkbit error, output link
queuing	time waiting at output link for transmission (dependant on congestion level
transmission	time to sent bits into link $\frac{L}{R}$, R=bandwidth, L=packet length
propagation	$\frac{d}{s}$, d=length of link, s=propagation speed $(2 \cdot 10^8 \frac{m}{s})$

- **Packet loss:** ie buffer full, packet may be retransmitted or not at all
  loss is part of system to maximize network usage

Vorlesung 10 - HTTP Intro

- **TCP** Connection-oriented, reliable transport (lost packages are retransmitted, flow control, congestion control (throttle intervall, when to much lost)
  3-way-handshake: `syn` → `synack` → `ack`, RTT (round trip time) time between `syn` and `synack`, used for timeout
- **UDP** unreliable, just sends on, no resending, if overhead bad, streaming, voip, dns, request time, ...
- **nonpersistent HTTP** one object per connection
  one RTT for connection initiation, one RTT for request and one for response, file transmission time ⇒ total: 2n RTT + transmission time
- **persistent HTTP** (/1.1) multiple objects per connection, one RTT for each object, client issues request only, when response has been received
  ⇒ n RTT + transmission time
- **pipelining** introduced in HTTP/1.1, client request as soon as object is encountered, nearly one RTT for all objects.
  ⇒ 3 RTT + transmission time

Vorlesung 11 - HTTP Specials, Mail

- **Cookies - keeping "state"**
  components:
  - cookie header line in the HTTP response message (if no client-side cookie present)
  - cookie header line in HTTP request message (if client-side cookie present)
  - cookie file kept on user's host and managed by user's browser
  - back-end database at Web site
- cookies can be used to store information about user
- **conditional GET:** (only transmit new object, if changed): `If-modified-since: <date>`
- **web caching:** have a proxy server with cached pages, origin-server only used, if page out-of-date (bottleneck, reduce outside traffic, etc)
- **Web hosting:** multiple domains per machine *or* multiple machines per domain
- **user-server authorization:** clients needs to send authorization with every message
- **FTP:** *File transfer protocol*, RFC 959, port: 21

used to transport files, control connection: port 21, data connection port: 20
browse server via control connection, when file transfer is requested, TCP connection is established
text is sent in ASCII (USER, PASS, LIST, RETR <filename>, STOR <filename>

- **SMTP:** *simple message transfer protocoll*, RFC 2821, port 25, uses TCP (handshake, transfer, closure), header and commands do not have to contains the same data (!), persistent connection, components:

  user agent mail reader like thunderbird, opera, webmail, outlook ...
  composing, editing, reading mail

  mail server mailbox, contains incoming messages for users, outgoing messages (to be sent) stored in message queue
  smtp between servers to send email messages

  MIME used for multimedia content: text (plain, html, tex-source, etc), image (jpeg, gif, png, etc), audio (mpg), model (vrml), video (mpeg, quicktime, etc), application (other data, that mus be processed by reader)

- **IMAP** *Internet Mail Access Protocoll*, RFC 1730 and **POP** *Post Office Protocol*, RFC 1939
  POP just downloads (stateless), IMAP more complex, manipulation of msgs on server, all messages stay on server (keeps user state)

Vorlesung 12 - Transport Layer: UDP, TCP, IP, NAT

- **UDP:** unreliable, unordered, unicast or multicast
  socket identified by <dest ip> and <dest port>, upon receiving: host directs segment to corresponding socket
  different source ip but same dest directed to same socket
  segments may be lost or delivered out of order → ignore lower seg#
  no handshake, each segment handled seperately
  why? no connection establishment (delay), simple, small segment header, no congestion control (maximum speed), streaming (clients can "just connect")

- **UDP Checksum:** segment treated as 16-bit integers, checksum = sum(16-bit words in one's complement)
  if checksum is all 0, value should be all 1s
  if checksum value is all 0, no computation of checksum is assumed
  receiver: add complement of checksum ("negation"), all 1s → ok, not all 1s → error

- **TCP:** reliable, in-order, unicast
  congestion + flow control + connection set
  socket identified by <source ip>, <source port>, <destination ip> and <destination port>, all four values used for socket identification (web servers have different sockets for each connecting client

  - **point-to-point:** one sender, one receiver
  - **reliable, in-order byte stream:** now "message boundaries"
  - **pipelined:** congestion and flow control set window size
  - **send and receive-buffer**
  - **bi-directional data flow**
  - **MSS** maximum segment size: 1460 bytes (+40 bytes header)

  - **handshaking**
  - **congestion control** ensures, sender will not overwhelm receiver/network
  - **seq #** and **ack #** ensure no packages are lost, retransmission by timeout, multiple acks
  - **timeout** based on average of several recent SampleRTTs by Exponential Weighted Moving Average (EWMA) (older less important) and safety margin by standart-deviation
  - **closing of connection:** similar to three way handshake, clients sends `fin`, server sends `ack` and `fin`, client sends `ack`

- **multiplexing/demultiplexing:** based on sender, receiver port and ip
- **multiplexing:** gathering data from multiple app processes, enveloping data with header
- **demultiplexing:** delivering received segments to correct app layer process
- **DHCP** *Dynamic Host Configuration Protocoll* = "plug-and-play" Users should be able to get multiple sessions with same ip-address
  user: `DHCP discover`, DHCP server: `DHCP offer`, user: `DHCP request`, DHCP server: `DHCP ack`
  DCHP assigned IPs have a lease time, lease can be renewed
- **ICANN** *Internet Corporation for Assigned Names and Numbers*: allocates ip-addresses, manages dns, assignes domain names, resolves disputes
- **NAT** *Network Address Translation*: local network just uses one IP address as far as outside world is concerned (devices not directly visible to outside world, but through ports)
  NAT has to replace source ip and port in outgoing messages and
  NAT has to remember (in table) which pairs correspond
  NAT has to replace destination source and port
  60 000 simultaneous connections with single lan-side address
  NAT violates end-to-end principle (has to be taken into account by app designers, routers should only go up to layer 3, address shortage should be solved by IPv6
- **Private Adress Space** IANA has reserved three spaces (RFC1918) 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16