1.3 ASCII

(American standard code for information interchange)

character sets:

- many different character sets exist
- charakter sets map symbols/characters to binary numbers
- used to be non-standerdized, 7bit, and different for each language
- Original ASCII was 128 characters (7 bit) but later extended: 256 characters (1byte)

Encoding Tables

- characters encoded in binary
- basically a "lookup table"
- characters are grouped
  - ↳ if 97 = a, then 100 = d always

Unicode

- uses 16 bits, way larger than ASCII (extended)
- aims to cover all symbols and characters ever used
- has 120,000+ characters
- actually one Unicode set can contain just 65,536 characters

Converting from ASCII 7-bit to 8-bit

- 8-bit ASCII is created so that 7-bit ASCII transformed / be used along 8-bit ASCII

- UTF-8 is the most popular Unicode set can easily be

eg. 7-bit 'a': 1100001

8-bit 'a': 11100001

↑ just add a 1 bit to the most left-hand side

Some key-facts about ASCII

- there are graphic codes and control codes
  - ↳ used to assist in data transmission and representation (formatting)
- The codes are sequential: code for 'F' + 1 = code for 'G'
- The codes for upper- and lowercase letters only differ in one bit
- ASCII numbers are only used in the context of stored, displayed, or printed text.

Unicode UTF-8

- most popular Unicode character set
- mainly includes 1 byte characters (therefore the '8'), but also some characters using 2, 3, or 4 bytes
- ASCII is also contained in UTF-8, and written with a 0 for the 8th bit
- Byte formats for Unicode UTF-8

ASCII: 0???????

2 byte symbol: 110????? 10??????

3 byte symbol: 1110???? 2x 10??????

4 byte symbol: 11110??? 4x 10??????

number of leading '1', followed by a '0' determine number of bytes coding for the character

following bytes are indicated by a leading '10'

All the ? bits are used to code for characters, so eg. the three-byte format allows for 11 coding bits, so 2^{11} possibilities

Question 1.02

What characters would be included in a 6-bit character set?

$2^6 = 64$ characters $26 \times 2 = 52$ $52 + 10 = 62$

This is just enough for

- upper and lowercase letters
- full set of denary numbers
- space character, enter character

missing: symbols, emojis, formating/instructional characters

1.4 Multimedia

Representing images in binary

- Bitmap: an image stored as a 2d array of pixels
- Pixel: an individual picture element
  = the smallest distinguishable feature
- Monitor: VDU (Visual display unit)
- resolution: width × height, but also can be eg ppi (pixels per inch)
  ↳ usually describes pixel amount / density
- representing an analog picture in binary: a grid is placed over the image and each color gets mapped to a binary representation (eg. 1 or 0 for black & white, or multiple bits for color)
- color depth: number of allocated bits for each pixel's color (eg. color depth of 3: ??? or 2^3 possibilities)
- increase in resolution and color depth increases filesize

$$Size(b) = W \times H \times D$$

$$Size(B) = \frac{W \times H \times D}{8}$$

(width, height, color depth)
in pixels, in bits

- Metadata: data about data
  - required to display an image properly, stored alongside the raw image data
  - Required:
    Filename, file format, color depth, resolution
  - Non-Required: camera details, etc.

Representation of Sound in Binary

Analogue vs. Digital

- Analog is continuous, has no defined range -> often shown as wave
- Digital data has defined ranges and only is able to show discrete values

Digitising Sound

- Sampling: recording a signal at regular intervals
- Sampling frequency: number of samples taken per second of audio in Hz
  ↳ Higher Sampling Frequency → better sound accuracy but larger file
- Sample Size: How many bits are available for each sample (eg. Sample size of 3 is 2^3 steps so only 8 values)
  ↳ Sample Size corresponds to levels of loudness
- Bit rate: number of bits used per second

$$\text{Bit rate (bps)} = \text{Sampling frequency (Hz)} \times \text{Sample size (b)}$$

- $\text{Total Filesize} = \text{Bit rate (b)} \times \text{Time (s)}$

Sampling Frequency = Sampling Rate | Synonyms
Sample Size = Sample Resolution

Advantages of Analog Sound:

+ more accurate representation of sound
+ can be represented by voltage strength
- vulnerable to noise, hard for long-distance transmission
+ high information density
- Needs amplification

Advantages of Digital Sound:

+ less influenced by noise: clear 1 / 0 data transmission
- less accurate, lower information density
- great for binary systems and long-distance transmission
- can carry much more information easily (if bit rate is high enough)
  ↳ able to carry two waves at once (eg.)

Vector graphics

- Images stored as vector graphic files consist of many individual drawing objects
- each drawing object has properties
- those objects, with their properties, are stored in a drawing list
- properties contain atributes such as geometrical shape, color, and shape-specific information
- drawing Objects are defined relative to an imaginary drawing canvas, not with an explicit position
- vector images are therefore scalable
- for a vector graphic to be displayed on a conventional display it has to be transformed into a bitmap

drawing list [
drawingObject_0 (x y, etc.
drawingObject_1 (x y, color, ...
⋮
]

Worked Example 1.04

bit depth of 8
72 dpi
5 × 3 inches
How many bits for this image?

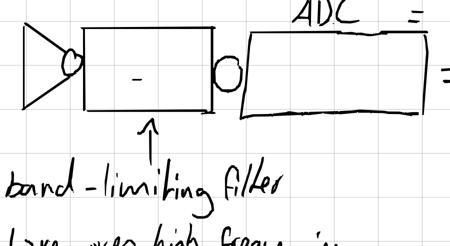
→ $8 \cdot 3 = 24$ bits for color
$5 \cdot 3 = 15$ square inches
72^2 = pixels per square inch

$$\frac{24 \times 15 \times 72^2}{8} = 233{,}280 \text{ bytes}$$

$$\frac{233{,}280}{1024} = 227.81 \text{ KiB}$$

Book: On Audio Sampling

sound encoder:



ADC = Analog-to-digital converter (Sampler)

Nyquist's theorem:

- The sampling frequency must at least be twice that of the highest frequency

---

End of topic review

p. 24, 25

2a. 2b., 3, 5ab

2a i For vector graphics all the geometric components of an image are stored in form of drawing objects in a drawing list. Those drawing objects have attributes that define e.g. color, size, shape, etc.

ii For a bitmap image pixels are stored with their color attribute in a 2D array.

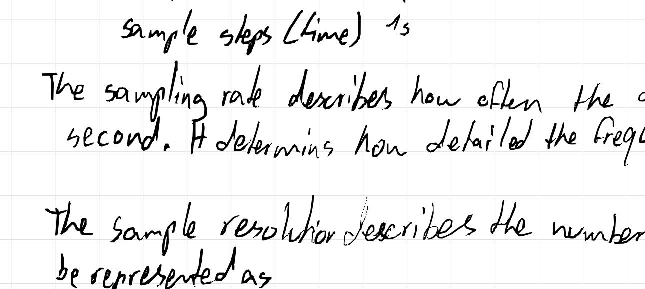
iii Vector graphics has better immage quality when scaled up, as objects are shapes existing merily relative to an imaginary canvas, whilst bitmaps have a pre-defined resolution that doesn't scale

2b i $\frac{640 \times 480 \times 16}{8 \times 1024} = 600$ KiB

ii This is an approximation, as no metadata is included in the result

3a i It's required to sample the analog measurements into binary values, which can be stored or transmited by digital systems.

ii



The sampling rate describes how often the analog signal is measured a second. It determins how detailed the frequency is captured

The sample resolution describes the number of steps the amplitude can be represented as

3b i This would be the band limiter.

ii It filters out high frequencies which are not hearable by humans and might cause damage to the ear in some cases or [cause issues with the digital systems.] → Nyquist's theorem wouldn't be satisfied by any sensible value

5a → 3a ii

5b i $\frac{44100 \times 16 \times 2}{8} = 176400$ bytes

ii Calculate number of bytes required per second and multiply by total time in seconds then I would divide by 1000 to get to KB and divide by 1000 again to get a result in MB.

$$\frac{\text{sample frequency} \times \text{sample resolution} \times \text{number of tracks}}{8 \times 1000 \times 1000} = \text{Total Size in MB}$$

1.5 Compression

Lossy vs Lossless compression

- Lossy compression:
  - store larger areas of pixels as a single color
  - reduce color depth
- Lossless compression:
  - store the color and pixel of that color in a row (eg)
  - ideal for images with large patches of exactly the same color (eg. logos, cartoons, etc.)

↳ excellent for vector graphics

purpose of compression

- reduce file size
- reduce download times
- reduce storage requirements
- make best use of bandwith ← critical

! a compressed file needs to be uncompressed in order to read it

difference between lossy and lossless compression:

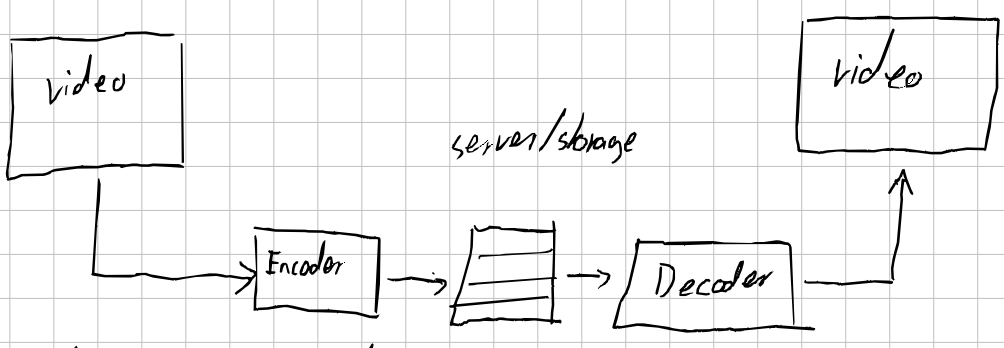
lossy: data gets lost

lossless: all data can be restored through decompression to the original

When to use which?

- files such as text documents are not applicable for lossy compression
- lossy compression is good for image, audio, video, multimedia...

Codec: A piece of software that encodes and decodes information (data)



encoder: compresses video

decoder: de-compresses (original) file

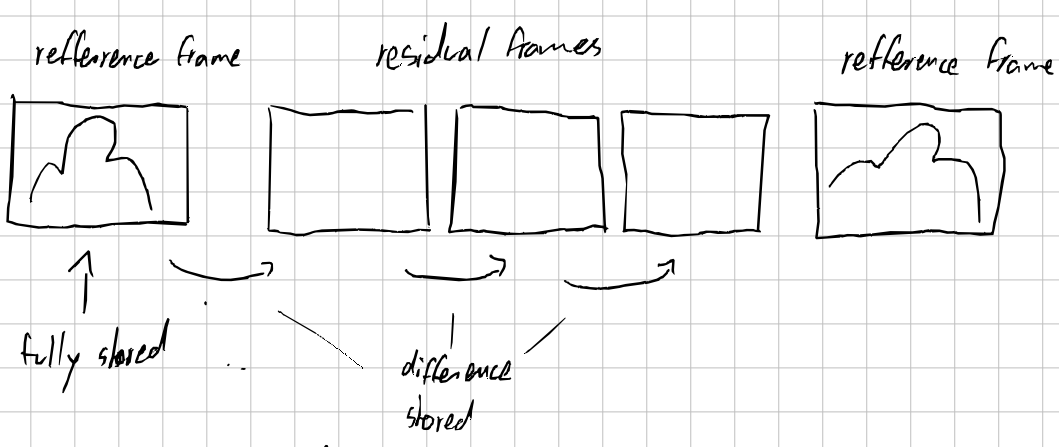
Video compression methods:

- Spatial (inter-frame) coding
  - compressing video frame by frame

  ↳ good but not great, as many frames are nearly alike → wasted compression opportunity (cross frame)
- block motion estimation, aka motion compensation
  - tile an image and for the next frame determine motion vectors that shift the blocks to closely match the next frame
  - keep the tiles that are not changing (just changing position)

Inter-frame: compression over multiple frames

Intra-frame: compression of individual frames



residual frames hold very little information, therefore are very compressible. (motion compression)

Book additions

- often lossy and lossless compression are combined for the best results

Run-length encoding

- lossless compression method, works well with bitmaps

How it works?:

Sequences of the same byte are compressed by defining the byte and then the number of times it is repeated (the count)

e.g. 01100110 01100110 01100110 01100110

is replaced by: 00000100 01100110 (simplified)

4 × ↷

it's a little more complicated, as multiplier bytes are currently not distinguishable from the image data. There are many different methods to do so

Huffman coding

- used for lossless text compression
- sometimes used for sound files and vector graphics
- works by finding the most common values and assigns shorter codes to them, therefore sound with similar values for amplitude, and scalable vector graphics format (svg) which is text work great

Simplified:
1. look for the most used characters and assign them new codes (in the lookup table)
   eg. e = 01 t = 01
   ! no code is the prefix of an other code
2. The codes are packed into a byte stream, where one byte contains multiple character codes and some 0s for padding

Two lossy compression methods for audio:

- convert a file of amplitude values to a file of amplitude differences, then use a lower sampling rate to store those differences
- determine the frequency of rather consitent intervals and store one frequency and the time frame rather then multiple amplitudes

---

Extension Question 1.01:

Format	Compression Type	Main algorythms
JPEG	Lossy	DCT, Quantization, Huffman/Arithmatic coding
GIF	Lossless	LZW Compression, Indexed Colors
PNG	Lossless	Deflate (LZ77 + Huffman coding)
TIFF	Lossless/Lossy	LZW, PackBits, Deflate, JPEG (optional)

Exam style questions:

p. 24–25 2c, 5c

2c i 1/2 The run-length encoding is ideal for lossless bitmap compression. Sequences of the same bytes get replaced by a sample of those bytes and a count, telling the computer how often in a row this sample exists.

ii The color depth could be reduced. Explanation missing: Each pixel's color value gets changed to an approximation requiring fewer

5c 0/3 When playing the MP3 file, the compressed data first has to be decompressed (restored) through many calculations. The decompressed file is very similar to the original file.

New try: 2/3 The human does not (hardly) notice the details lost through compression. Some noises might be out of the human's hearing range (to high or to low) and therefore can be neglected. Also a softer sound in the presence of a louder sound will not be heared.

1.6-1.7 Programming Basics

Python string basics

name = 'Tom'

print('Name: %s' %name) >> Name: Tom

Nassi-Scheiderman Diagram

1. Task 1 — Basically an action flowchart
2. Task 2 — ordered chronologically
3. Task 4
   ⋮

2 12

128 64 32 16 8 4 2 1

1 1 0 1

1 1 1

Exam style questions 2.2

1. a) i They could consider twisted-pair, coaxial, or ~~optic-fiber~~ fibre optic

coaxial with 'i' ✓

Explain factors...

ii They might want to take permiability to interferance, and the attenuation of the different cables into account.

^explanation missing

b) i I'd recommend the default WiFi standard with a router that is connected to multiple WAPs.

ii It is easy to extend a wireless network, as new nodes don't require new cables (within limits of course).

iii To many devices connected to the same WAP might result in a congestion. more interference likely would be better?

c) i PSTN is the Public Switched Telephone ~~System~~ Network (✓) which handles calls on a local to even global scale. It is a large network of once analog (now some are digital) switches that create a direct connection from one telephone / end-system to another, unlike sending split packages like the internet

ii PSTN could provide a communication network that would significantly reduce cost of inhouse calls between colleagues. This would work with switches that could be placed in a certain hirachy, one for each office building and then one to connect all buildings on the sight. ✓

1/3, missing two more examples!

e.g. PSTN could be used with modems for Data transfer.

PSTN could provide a leased line for data/call connection

2.4 Practice & Exam-style

2.02 Buffer: 1 MiB

Low-water mark: 100 KiB

High-water mark: 900 KiB

Input: 1 Mbps ≈ 122.07 KiBps

Output: 300 kbps 36.62 KiBps

Initial: 160 KiB

$1\,Mbps = 1000\,kbps = 125\,KBps$

$\frac{125 \times 1000}{1024} \approx 122.07\,KiBps$

$300\,kbps = \frac{300 \times 1000}{8 \times 1024} \approx 36.62\,KiBps$

$Buffer = f(t) = 100 + (122.07 - 36.62)t$

$f(4) = 441.8$ KiB

$f(6) = 612.7$ KiB

$f(8) = 783.6$ KiB

$f(10) = 954.5$ KiB

$f(12) = 1125.4$ KiB

] would stop before, not correct

$900 = 100 + (122.07 - 36.62)t$

$\frac{800}{85.45} = t$

$t = 9.362$ s

after $t = 9.362$ s $f(t) = 900 - 36.62(t - 9.362)$

$f(10) = 876.64$ KiB

$f(12) = 803.40$ KiB

$0 = 900 - 36.62t$

$t = \frac{-900}{-36.62} = 24.58$ s

Examstyle question #3

Describe: 6 points not explain. 1 point for every example

- He would have to ensure a stable internet connection so he is able to reciere data at all. (L) not listed
- He should have a proprerly sized buffer that is able to compensate for an unstable bit rate. ✓
- He should have enough baudwith ✓ so that the bit rate surpasses that of his stream, which is necessary for streaming.
- He should make sure his end-device is capable of fast download speeds, as the connection will only be as fast as the weekest link in the chain, not listed so (L)

not a good score :-: $\frac{2}{6}$ to $\frac{4}{6}$

we have to improve understanding of operators:

e.g. explain, describe, etc

2.5 Addressing

What happens when you click on a URL?

http:// example.com /some/path

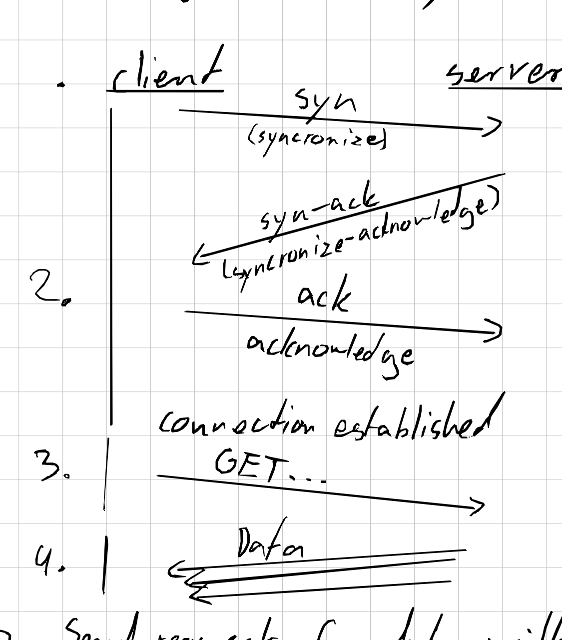
- http:// = protocol / schema
- example.com = domain name
- /some/path = path

On click:

1. Transform domain name into IP-address (destination address)
   - Either it's statically configured:

     HOSTS.txt is a file on your local system that maps common domain names to IP-addresses. This file doesn't really get modefied, but visited domain IPs are often saved in a DNS-Cache.
   - More commonly, a 'Domain Name System' (DNS) protocol is run. Explained later
2. Connect to server through Transmission controll protocol (TCP)

   TCP: three-way-handshake



3. Send request for data with a TCP packet

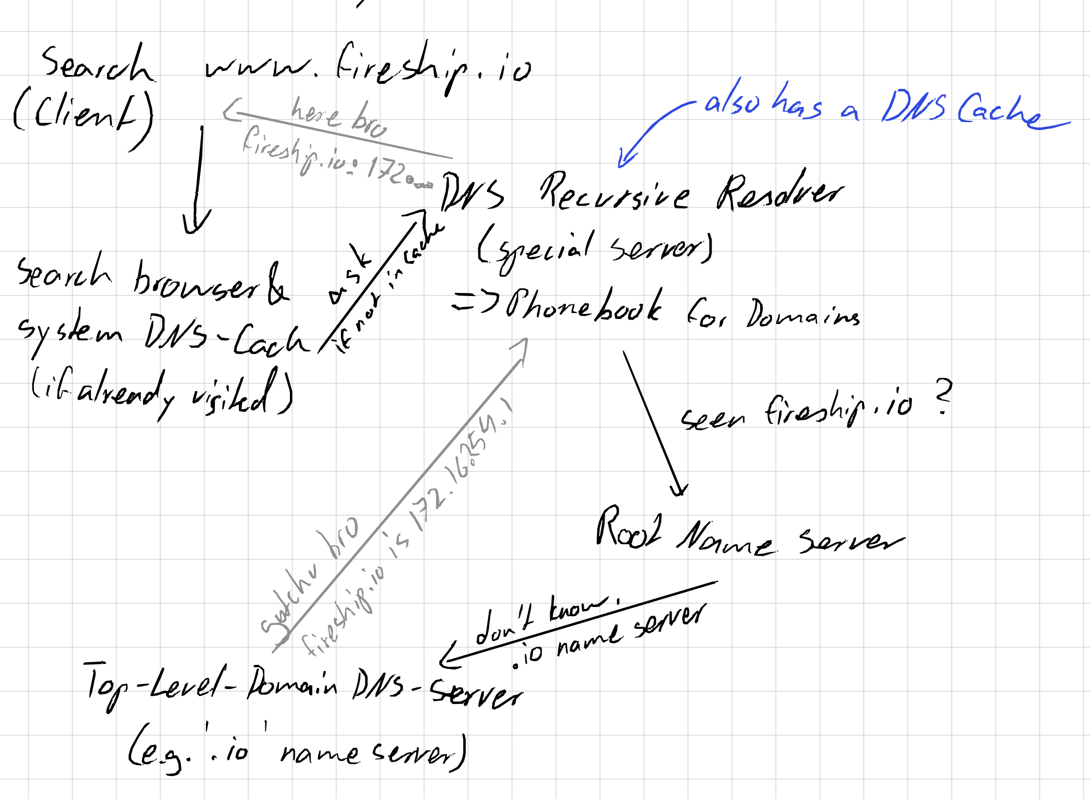
   usually with a GET /some/path/ HTTP/Server IP
4. Recieve data from server with TCP

DNS Domain Name Service/System

www.fireship.io gets mapped to IP-Address 172.16.259.1

- www = sub-domain
- fireship = domain
- .io = Top-Level Domain (TLD)

- DNS is executed when a client wants to connect to a server but only has a domain name.



- Domain Name Servers are set up in a hierarchical manner, and the DNS name space is devided up amongst the top-level domain name servers so that there is no overlap
- There are more than 250 top-level servers
- The DNS search process is called "name resolution"
- if the domain is under control of the connected server (e.g. a local request) than the server directly provides the matching IP addres

IP addressing

IPv4 addressing

- developed in the 1970s,
- not enough unique addresses for all global clients
- 32 bits long (4 bytes) ~ 4 billion addresses

original addressing scheme:

- five rigid categories that allowed for allocating more or less bits to the hostID and the netID.
- Destinguished by a classefier, some schemes allowed for a select few IPs to have many hosts in the network.

Class	Class identifier	bits allocated to netID	bits for host ID
Class A	0	7	24
Class B	10	14	16
Class C	110	21	8
⋮			

- There can be only 128 Class A IP addresses that allow for over 16 million devices. On the other hand ~ two million Class C IP addresses allowed for 256 hosts each.
- There weren't enough Class B addresses available, though companies required more than the 256 host IDs class C provided

An IP address is written as the denary equivalent of each byte:

10000000 00001100 00000010 00011110

⇓

128.12.2.30 — This is called dotted decimal

Classles inter-domain routing (CIDR)

a fix for IPv4 addresses: flexible IPs

32+8 bits: 11000011000011000000011000001110/0010101 *suffix (in this case = 21)

or 5 bytes — netID (len 21) — hostID

= 195.12.6.14/21

*the suffix determins the number of bits used for the netID

This scheme allowes for Classes A, B, C+ to be integrated into the new system by adjusting the suffix

- Class A: suffix = 1 class identifier bit + 7 netID bits = 8
- Class B: suffix = 2 + 14 = 16
- Class C: suffix = 3 + 21 = 24

Sub-netting

a different approach to fixing the IPv4 crises

- first three bytes used for netID, leaves 256 hostIDs for each network
- bits from the hostID can be used to create sub networks, usually the first three bits are used
- 3 bits are standard and allow for 8 subnetworks with 32 devices each

00000000 00000000 00000000 | 000 | 00000

netID | sub-netID (can vary) | HostID

! xxx.xxx.xxx.255 can't be used for hosts !

it's for broadcasting only

Sub-net mask:

Allows to identify bits used for the netID, including the subnet IDs

a subnet mask of

255.255.255.0 is multiplied with the IP-address and results in the netID

subnet mask × IP address = netID (, subnet ID)

A subnetmask of 255.255.255.224 (224 = 1110000)

allocates three bits of the HostID to be used as the subnetID, allowing for 8 sub-networks

binary multiplication:

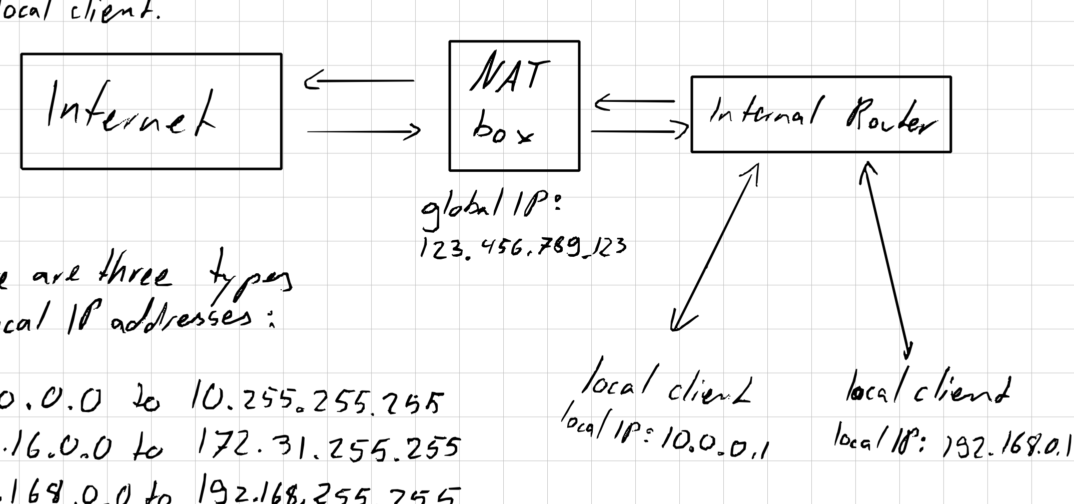
1×1 = 1
1×0 = 0
0×1 = 0

mask 'cuts out' portions

Network address translation (NAT)

Another fix for IPv4: IPs inside LAN are only locally unique

- All clients in the Lan connect to a private network, allso called intranet.
  - An intranet works just like the internet and therefore has its own IP system
- The private network connects to a "NAT box" which has a single IP with which it connects to the Internet
- Trafic from the private network is forwarded through that single IP address of the NAT box, while traffic to the NAT box is delivered to specified individual ports of the NAT box and forwarded to the appropriate local client.



There are three types of local IP addresses:

10.0.0.0 to 10.255.255.255
172.16.0.0 to 172.31.255.255
192.168.0.0 to 192.168.255.255

You don't have to remember those numbers, but you will anyway

Static and dynamic IP address

- Internet Service Providers usually have more users than IPv4 addresses available to them. So private networks get assigned a global IP address when they reconnect to the ISP/Internet. This would be a dynamic IP address.
- Some users pay more so that their private network is always reachable through the same (static) IP address.

IPv6 addressing

The Internet will soon migrate to IPv6 as it offers significantly more addresses

- 128 bits per address (16 Byte)
- broken into 8 2-byte parts that code for 4 hex digits
- segments are devided by ':'
- Multiple continuous segments of zero can be shortened by '::'
- this can only happen once in any address, as the position has to be clear
- leading zeros are omitted
- IPv4 addresses can be represented in IPv6

00F6:7C48:FFFF:0000:0000:3D20:0000:00F1

=> F6:7C48:FFFF::3D20:0000:F1

(0000 not shortened because it can only happen once; leading zeros disappear)

IPv4 address:

::192.31.20.46

2.5 Exam-Style Questions

#2, 4, 5, 6

2. a) i A software is used that compares domain names revieved from the client with a local database of domain name and IP pairs. The software runs on a variety of servers connected in a hierarchical fashion.

ii A browser uses DNS. As the user enters a domain name, the browser has to communicate with domain name servers in order to resolve the domain name (find the matching IP address).
Also Email uses DNS. When sending an e-mail the domain name has to be resolved into its current IP address.
A domain Name server is contacted before sending the mail.

b) i 205 128 64 32 16 8 4 2 1
1 1 0 0 1 1 0 1 ✗ the notation is dotted decimal

This would be class B, as the top bits are '10'.
The 32 bits for the IPv4 address are split into four 1-byte groups, which each code for an individual number. The left-most number codes partially for the IP class, at least in the classful scheme. The notation is in denary form.

ii Class B, as the top bits of the most significant byte are '10'.

iii The netID addresses identifies the LAN in which the client can then be identified through the hostID.
A netID is partially determined by the location, which significantly increases the efficiency of the routing process.
+ Addressing is hierarchical (-1 point)

c) 201.124.16.152/24

8 + 8 + 8

= 24

128 64 32 16 8 4 2 1
152 = 128 + 0 + 0 + 16 + 8 + 0 + 0 + 0

10011000

The 24 defines how many bits are used to determine the netID. Subtracting 24 from 32 leaves 8, as the IP address is segmented into 8 bit chunks and the NetID is to the right left this leaves the 8 bit number to the right of the netID as the hostID. The chunk in denary is seperated by a dot, and the number 152 in binary is 10011000.

4. a) Private IP addresses can be used in private networks, also called intranets. End systems in a LAN have different private IP addresses and share the same public IP through a NAT box that connects the routers to the Internet.
+ that uses TCP/IP (-1 point)

b) Due to the nature of the intranet all local traffic stays inside the intranet if no external IP is addressed. All traffic to the Internet has to pass through the NAT box first, which replaces the private IP with a public one. When recieving data from the Internet the NAT-box then forwards what was recieved on a specific port on the public IP address to the appropriate private IP, inside the intranet.
missing three points bro. They want you to go deep but also broad

5. a) i 11.64.255.90 128 64 32 16 8 4 2 1

10 A
11 B
12 C
13 D
14 E
15 F

11: 0 0 0 0 | 1 0 1 1 0B
64: 0 1 0 0 | 0 0 0 0 40
255: 1 1 1 1 | 1 1 1 1 FF
90: 0 1 0 1 | 1 0 1 0 5A

= B.40.FF.5A (✗) not explicitly wrong, book says to keep leading zeros in hex

ii 32 bits are segmented into four 8-bit (1-byte) groups. Each 8-bit segment codes for a denary number ranging from 0 to 255. Segments are seperated with a dot and leading zeros are neglected.
+ could add seperation of hostID and netID

b) First the web browser checks its own DNS cache and then the cache of the OS. If the matching IP address can not be found, the browser contacts a DNS recursive resolver server that will follow through a recursive programm until it has resolved the DNS. It also has a DNS cache which it first checks, though if that cash doesn't contain any matching address a root name server is contacted. This server then responds with an appropriate top-level server that matches the Top-Level Domain of the address. This server should be able to provide the IP address for the requested domain which is then forwarded over the DNS recursive resolver back to the client's browser.
good, less detail was requested, rather definition and explanation of

6. a) IP stands for 'Internet Protocol'. URL and DNS would be necessary

b) 3.2A.6AA.BBBB invalid, '6AA' and 'BBBB' exceed the maximum vale of 'FF' (255 in denary) per segment.

2.0.255.1 Valid. (All denary numbers range from 0-255 there are four segments, all seperated with a dot. not required, bad question)

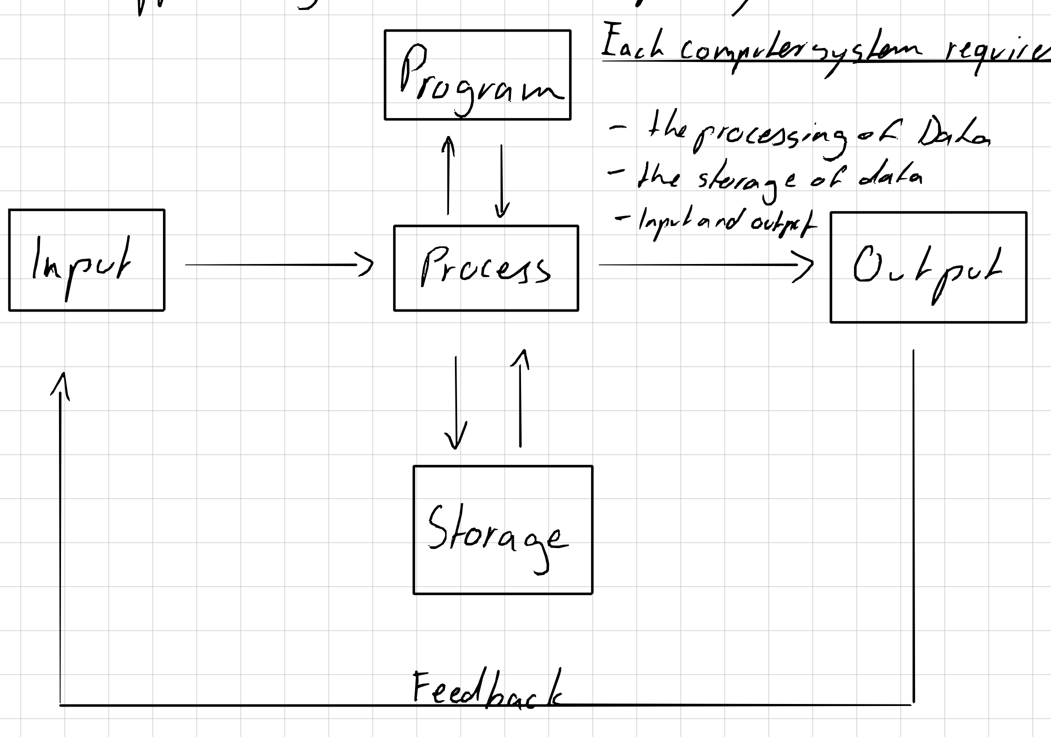
b.0.257.6 Invalid: The value 257 exceeds 255 and cannot be represented with 8 bits.

A.78.F4.J8 Invalid: J is not part of the hexadecimal system.

c) Public IP addresses are globally unique while private IP addresses are not.
Public IP addresses are exposed to the public, while private IP addresses exist in 'secure' private networks. This means that exclusively public IP addresses are used to directly communicate over the Internet.

3.1 Input, Output, and Storage Devices

Basic opperating structure for computer systems:



Each computer system requires:

- the processing of Data
- the storage of data
- Input and output

Input:
Any signals from input devices e.g. Mouse, Microphone

Storage:
Information stored on storage devices, e.g. hard drives, SSDs, DvDs, etc.

Program:
Provides the CPU / Process with instructions on how to handle inputs and outputs, basically giving the process logic

Process:
Inputs are recieved by a central processing Unit (CPU) which interprets the input and communicates ~~(usually)~~ with a storage device.
Decides what to do next

Output:
A system can give output through output devices, e.g. Screen, speaker, electric motor

---

Embedded Systems:

= A special single-purpose computer usually embedded in a larger system. Those systems run on so called 'micro controllers'.

Such systems can perform only a select few tasks and are hard to modify/programm. Some examples:

Embedded systems	Non-Embedded Systems
◦ Microwave with processor	◦ Smartphone
◦ Hospital devices (E.g. EEG)	◦ Laptop / PC
◦ Laundry Machine	
◦ Calculator	

Important!
Embedded systems require:
- processor
- memory
- inputs and outputs

Benefits:
- cheap to produce
- same embeded system required in many different applications

Problems:
- hard to re-programm
  ↳ ungraceful with development mistakes
- very low memory - challenging programmers
- network capable versions have low security standards
  ↳ Bot-nets

Data Storage:

Primary vs Secondary Storage RAM, ROM, Cache (not register!)

The storage usually closest to the system is the RAM which can be accessed directly by the CPU.
Long-term storage refered to as (file-) store would then be a secondary storage device

Secondary Storage is cheaper and used to compromise cost.
It is slower, takes up more space, but has a higher capacity for less money.

Primary storage is expensive and used for the internal processes that have to run fast. It is significantly faster, but has a lower information dencity, costs more and has usually a smaller capacity.

Types of storage devices:

- Integral Storage: build into the machine / Chips e.g hard drive
- External Storage / peripheral storage: storage that can easily be connected or disconnected from a device, e.g. USB stick, DVD
- Remote / Cloud storage: Accessible via network, e.g. SAN (Storage area network) or cloud

Common Data Output:
- display
- printer
- speaker
- output to storage device
- transmission over a network

Common Data Input:
- mose / controller / keyboard
- touch screen input
- scanner
- microphone
- read from storage device
- recieving transmissions over a network

All inputs and outputs of a computer are handeled by an I/O subsystem. It also handles memory.

Main Memory:

... directly accessible by CPU: RAM, ROM, (cache) ↩ sometimes

volatile memory: faster, but looses its data when power is lost
non-volatile memory: slower, but retains memory when power is lost

RAM and Cache are volatile, ROM is non-volatile memory

RAM:

(Random Access Memory)

Every programm currently in use is copied or 'loaded' into RAM because it's much faster.
Ram is called Random Access Memory because it takes always about the same time to retreave a certain amount of information from anywhere on the RAM, independent from the previous location accessed.

Dynamic RAM (DRAM):

made up of small capacitors which constantly need to refresh their values as they're leaking electricity. Has significantly less components then SRAM

Slower then SRAM but has a higher bit density, used for Main memory

Static RAM (SRAM):

Stores bits with flip-flop circuits, unlike DRAM memory precists, as long as power is provided → signal doesn't fade. They are more complex and more expensive to produce.

Faster (access speed) then DRAM but has a lower bit denity, often used for cache,
Embeded systems usually only use SRAM for main memory

ROM:

(Read-only memory) also a type of RAM

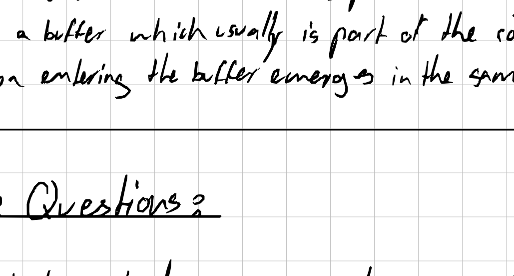
Can't be changed and is used in very small amounts to store essential programms
↳ e.g. Firmware
Now replaced by the more practical flash storage

- simplest form of ROM: Data is embeded during the creation of the chip → unchangable
- Programmable ROM (PROM) → Data can be loaded (after manufacturing) ONCE.
  ↳ not re-programmable, better for testing as previous
- Erasable PROM (EPROM) → using ultraviolet light the PROM can be areased and re programmed.
  ↳ Chip will have to be temporarily removed from circuit
- Electrically Erasable PROM (EEPROM) → major advantage for development as the chip can stay on the circuit
  ↳ chip is still used as read-only memory!

Cache:

Much faster then RAM but with significantly less capacity

Stores frequently used data, though it's volatile



Buffer:

*Even when transfering from a slow to a faster medium, a buffer is still used to ensure a steady smooth data stream

used to match unequal trasmission and reception speeds:
whenever data moves from one part of the system to another it's temporarily stored in a buffer which usually is part of the computer memory.
Information entering the buffer emerges in the same order as it entered the buffer.*

Practice Questions:

3.01 When loading a computer game from storage to the significantly faster RAM it would be reasonable to use a buffer in order to compensate for the difference in speed. The buffer smooths out the single data packets that come in intervals from the secondary storage

Exam Style Questions

1 a) i Information on the RAM can be changed why the ROM is read-only. RAM is volatile, while ROM is non-volatile, therefore precists even in the absence of power.

ii Both have the random-, or "direct" access capability, both are a type of RAM.

iii DRAM is Dynamic RAM. It's made up of capacitors, needs constant refreshes to not loose its data, is slower then SRAM but has a higher bit density. DRAM is cheeper.
SRAM is Static RAM. It's made up of flip-flop circuits which are faster then DRAM but take up more space. Therefore the bit density for SRAM is lower.

4. a) Hard disk: a fast rotating disk that stores data through magnetic encodings on that disk. → magnetic medium
DVD-RW: a fast spinning disk that uses optics to read and write from to the disk (LASER). It stores data as indents of different lengths in its surface. → optical medium

b) haven't done... Flash memory → solid state medium

5. a) see 1A

b)

Dynamic RAM	Static RAM
uses capacitors for data storage	uses flip-flops for data storage
requires constant refreshes	doesn't require constant refreshes
is very compact due to the few components per cell	is less compact due to complex circuitry.

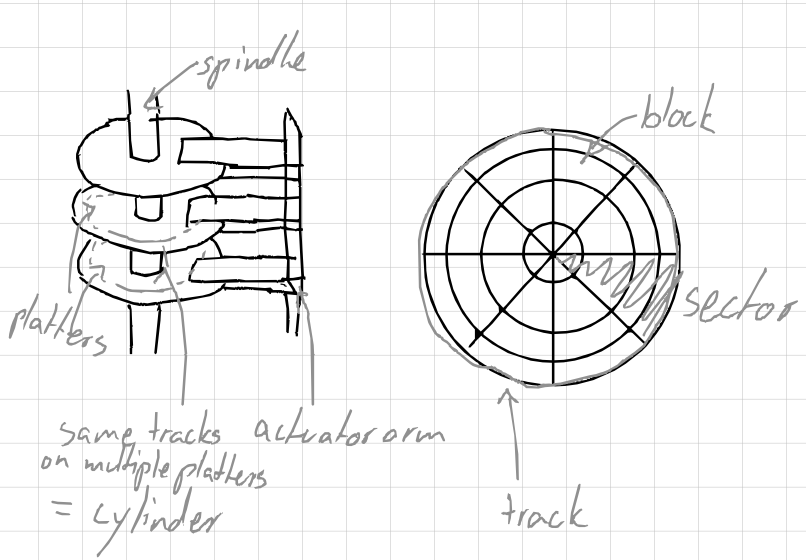
3.3 Internal Operation of Hardware

Every hardware device requires a program that enables its basic functionalities. This program is called a '<u>device driver</u>'.

Magnetic Media

note: peripheral devices refer to devices that can be connected to a computer

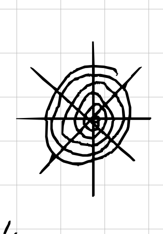
- first ever medium used as file store



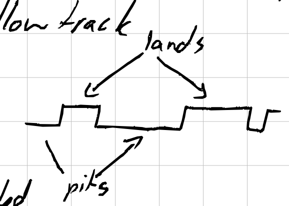
- Data is stored in tracks, which go all around the platter.
- Small areas of the disk can be magnetized to set the bit
  - ↳ the read-write head uses electromagnetism to write and detect magnetic fields.
- all platters spin continuous, and at the same speed
- closely related data is often stored close to the same radius, but on different platters → same track radia on different platters are called 'cylinder'
- While data is (preferably) stored on continuous tracks, the platters get segmented fast when editing and removing files
  - ↳ reduces performence
- Hard Drives are direct access read-write devices but are considered sequential as data has to be accessed in order
  - ↳ a sector is selected and read 'in order', meaning that for each reading the entire sector will be re-read.

Optical Media

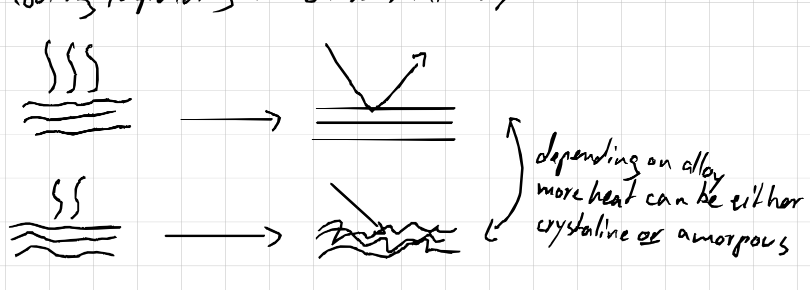
Compact Disk (CD)
- CD digital audio (CD-DA) ⇒ music
- CD-ROM (read-only) ⇒ software distribution
- later: CD-RW (read-write) ⇒ storage medium
  - ↓ digital versatile disk (DVD)
  - ↓ Blue-ray disk (BD)
    - + most effective as blue light can be focused even tighter → more data



- One <u>single</u> spiral track
- uses sectors
- laser moves to follow track
- '<u>pits</u>' and '<u>lands</u>'
- intensity of laser reflection is detected



- CD-RW/DVD-RW heat up an alloy which goes amorphous or crystaline after cooling, depending on the laser's intensity



- amount of data that can be stored is dependent on
  - physical limitations of the disk
  - wavelength of the light: smaller wav. len → more intense → easier to focus
- speed depends on the speed of rotation of the disk

Solid-State Media

- uses semiconductors (transistors) to save memory in little cells
  - ↳ transistors are arranged into 'NAND gates'
- solid-state media is also known as flash-media, as 'blocks' of memory can be erased all at once (in a flash)
- Reading and writing are managed by a 'NAND flash controller'
- Before data can be written the memory in that place first has to be erased
  - ↳ unlike e.g. magnetic tape
- currently the technology of choice for removable data, secondary storage generally
  - ↳ most common forms:
    - solid state drive (SSD)
    - USB flash drive
  - despite the absence of moving parts there is a limited amount of read-write cycles per cell
  - degradation can be detected and compensated for by the computer
  - Data is accessed in pages (one page at a time)

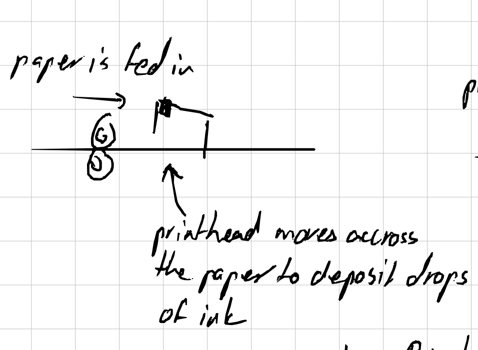
Screen Display

liquid-crystal display (LCD):
- uses liquid crystal, polarizers and a backlight to create images
- depending on the voltage applied to the liquid crystal at the sub-pixel, the polarization changes
  - ↳ this leads to a shift in the amount of light passing through the polarizing filter
  - ↳ The light for each sub-pixel passes a color filter (red, green, or blue)

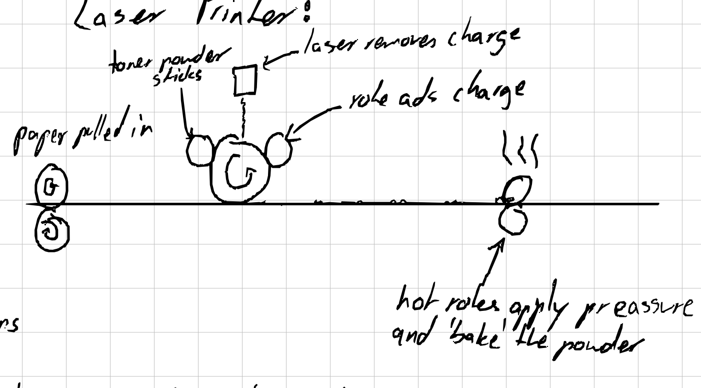
Cathode Ray Tube (CRT):
- an electron gun shoots electrons onto the screen which is coated with a layer of phosphor and starts glowing where the gun hit
  - ↳ pixels are defined by the size of steps the gun takes
  - ↳ an electro magnet is used to controll the beam's direction
- Color CRT works by creating a matrix of red, blue and green phosphor and shooting each sub-pixel

Printers

Inkjet Printer:



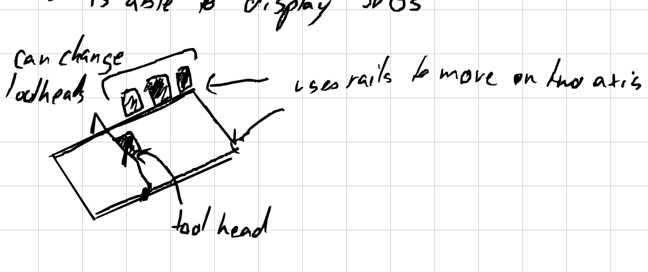
Laser Printer:



note: Printers use subtractive color creation. Colors used are cyan, magenta, yellow, black

Plotters

Graphic Plotter (2D)
- used for highly detailed drawings in engineering
- is able to display SVGs



3D Printer (Plotter)

- used for manufacturing
- a nozle squirts material onto a build plate layer by layer
- often the print has to be cured to ensure that the layers stick together

Input devices for general purpose computer systems

Keyboard

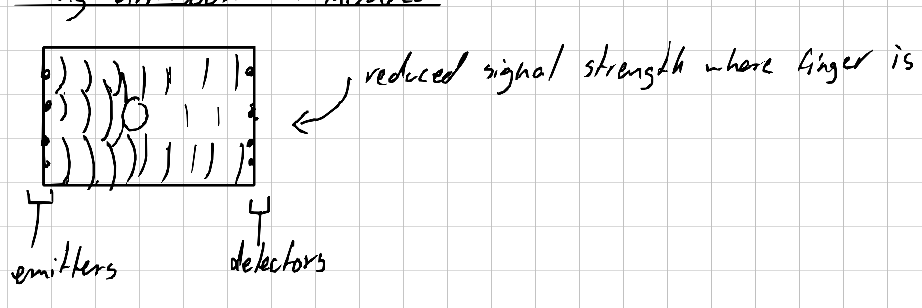
- with its own microcontroller the keyboard converts keystrokes into binary codes for keys.
- Keys are positioned over a so called '<u>key matrix</u>' which consists of two sets of grids.
  - ↳ on a keypress both matrecies are connected below the key
- the micro controller has to continuously check for input
- The microprocessor matches the signal from the matrecies with a binary sequence for a character from its ROM

Screen (GUI)

- Screens can be used as part of the input system too:
- graphical user interfaces provide digital controll surfaces that can be selected and manipulated
- note that the physical screen is considered an output device though interaction with the digital screen is considered input
  - ↳ a 'pointing mechanism' is required and turns the GUI into an input device

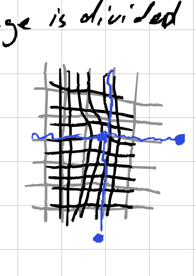
Touch screens

Using ultrasound or infrared:



Resistive Touch Screen

- two grids are seperated by a thin space
  - ↳ on touch both connect at the point of contact and the voltage is divided



- A soft screen is required to allow for slight bending

Capacitive Touch Screen

- when the finger touches the screen it causes a change in capacitive charge directly beneath the screen.
- electrodes span the width of the screen
  - ↳ an xy grid is used to determine the exact location of contact
- projective capacitive touch (PCT) allows for multiple fingers to be detected simultanously. (It's one of the newest versions of capacitive touch)

Input of Graphics

- webcam / digital camera
  - videos and images
- scanner
  - highly detailed and accurate digital copy
  - how it works:
  - a light source covering the width of the paper moves over the stationarry paper.
  - The light gets reflected by mirrors onto a 'charge coupled device' (CCD)
  - A CCD gives an analog signal proportional to the amount of light it recieves
    - ↳ an analog to digital converter is needed.
    - ↳ it mainly consists of photo-sensitive cells

Input and Output of Sound

Microphone

- uses a diaphrame to detect vibrations in the air
- electric circuit detects vibrations and outputs an analogue signal
  - ↳ condenser microphone measures change in capacitance
  - ↳ alternatively a 'piezoelectric crystal' can be used
- analog-to-digital converter (ADC) is needed

both require an audio card

Speaker

- basically a reverse microphone:
  - signal converted by DAC
  - electromagnet causes vibration in the diaphrame

Exam-Style Questions

2. a) i The key creates a connection between two electric grids (key matrecies). ✓ A processing unit recieves a signal of the x and y location of the key. ✓ The processing unit accesses the ROM and finds the matching binary character code. ✓ The keyboard sends the data to the PC. ✓ (computer)

ii A user could use Text-to-speech. ✓ ... screen, keyboard
Question asked for 1 but expected two?! Alr.

b) i (An embedded system converts the image to digital instructions for the laser.) A charge is applied to the photoreceptor drum. ✓ A laser removes the charges at certain places ✓
could add: moving mirror, moving drum

ii A toner role applies toner to the charged places, as the toner only sticks to those. The toner gets baked into place by heated pressure roles

iii The entire process for depositing the toner is repeated four times. Once for black, cyan, magenta, and yellow.

3. a) This technology would use either ultrasonic waves or infrared and detect a difference in the amount of recieved waves. This works by placing emitters and receptors at opposite ends both horizontally and vertically. A processing unit can then work out the exact location of contact.

b) An example would be resistive touch technology, as it requires a soft but flat surface to provide a precistent gap between the two conductive grids. By pressing onto the screen the two grids come in contact. A processing unit can then make out the exact position by looking at the voltage division that occured in the x and in the y.

3.4 Bolean Logic

Logic proposition: A statement that is either true or false

e.g. 5 is larger than 2: TRUE

Problem Statement: A construct of one or multiple logic propositions connected to each other

e.g. If it rains and I don't have an umprella, I'll be wet

Boolean opperators:

AND	AB	• A AND B = TRUE if A = TRUE and B = TRUE
OR	$A+B$	• A OR B = TRUE if A = TRUE or B = TRUE
NOT	$\overline{A}$	• NOT A = TRUE if A = FALSE
NAND	$\overline{AB}$	• A NAND B = TRUE if A = FALSE or B = FALSE
NOR	$\overline{A+B}$	• A NOR B = TRUE if A = False and B = False
XOR	$A \oplus B$ $= \overline{A}B + A\overline{B}$	• A XOR B = TRUE if A = TRUE or B = TRUE but not both of them

Logic expressions: An equation made up of logic propositions and boolean operators

AND, OR, NOT

can make up every logic composition

Truth tables

TRUE = 1
FALSE = 0

Convention: Variables: A, B, ...
Result: x

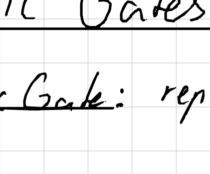
also count binary:
00
01
10
11
⋮

x = A AND B (x = AB)

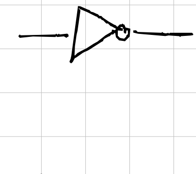
A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates & Circuits

Logic Gate: represents a boolean operation in circuit form.

NOT 

A	x
0	1
1	0

AND 

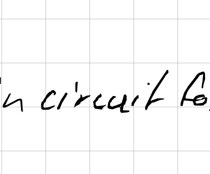
A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

OR 

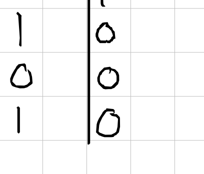
A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

NAND 

A	B	x
0	0	1
0	1	1
1	0	1
1	1	0

NOR 

A	B	x
0	0	1
0	1	0
1	0	0
1	1	0

XOR 

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

Task 4.01

covered by copyright ≙ A
Permission obtained ≙ B
Can be copied ≙ x

X = NOT A OR (A AND B)

$x = \overline{A} + AB$

Task 4.02

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Exam-style questions

1a) i Gate 1: OR
Gate 2: AND
Gate 3: NOT

ii Gate 1 OR

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

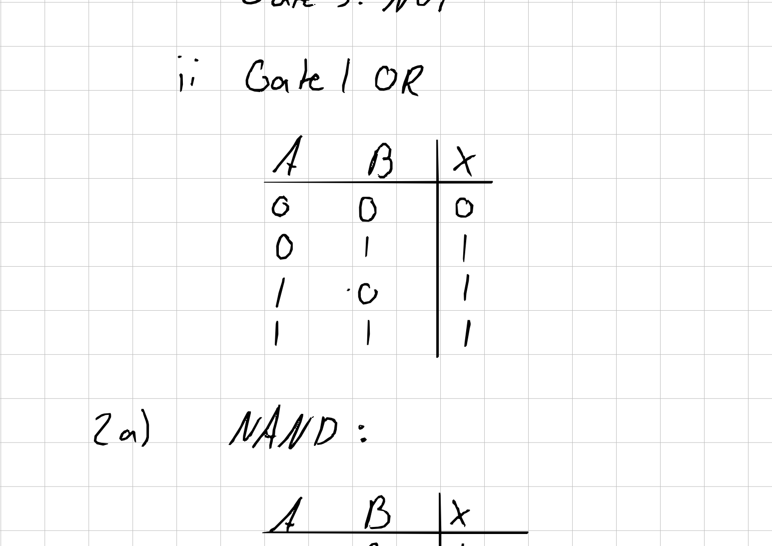
2a) NAND:

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

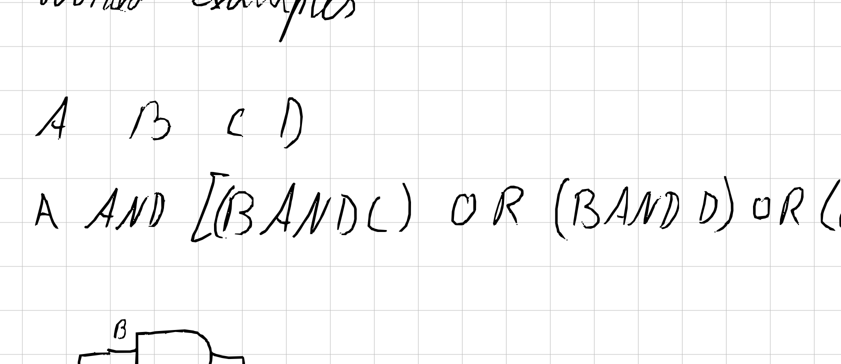
Worked examples

A B C D

A AND [(B AND C) OR (B AND D) OR (C AND D)] = x



How to solve datatables from circuits:



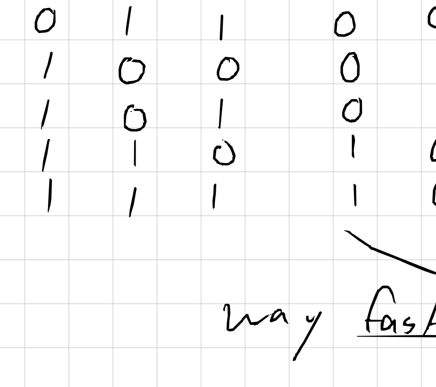
• define helper points (M, N, P, Q) to determine complex logic circuits.

Inputs			Workspace				output
A	B	C	M	N	P	Q	X
0	0	0	0	1	1	0	0
0	0	1	0	1	1	1	1
0	1	0	0	0	1	0	0
0	1	1	0	0	1	0	0
1	0	0	0	1	1	0	0
1	0	1	0	1	1	1	1
1	1	0	1	0	0	0	0
1	1	1	1	0	0	0	0

way faster

• Note: When just the solution (which combination) is asked for, not the table, then you can take 10x shortcuts.
E.g. note how C has to be '1'? This throws out half of the combinations.

Task 4.03

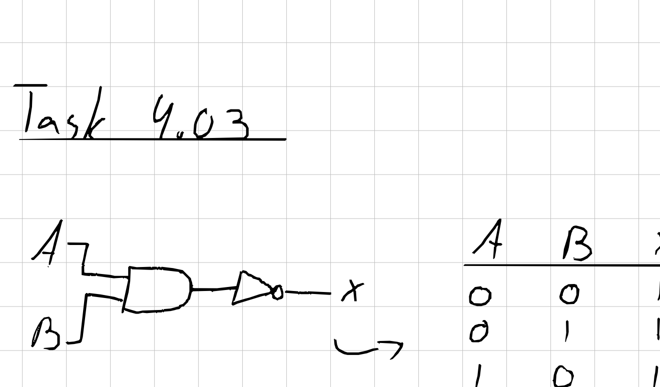


A	B	x
0	0	1
0	1	1
1	0	1
1	1	0

same for both

Task 4.04

C AND ((NOT B) OR (NOT A))

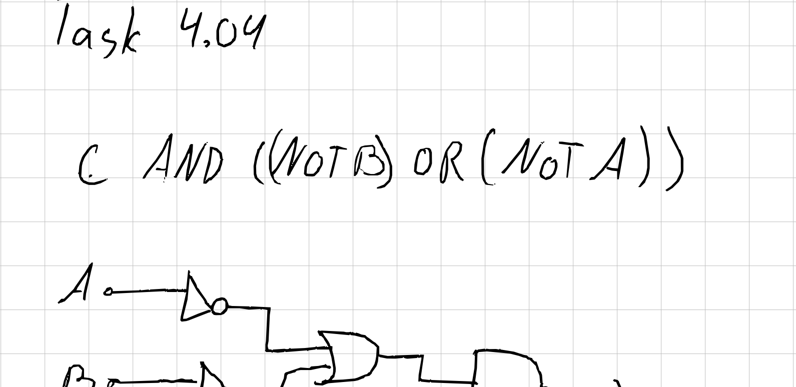


Exam Style Questions Round #2

2b) i, A ≙ won match #1
ii B ≙ won match #2
C ≙ have higher score

Different interpretation

(A AND B) OR (A AND C) OR (B AND C) = x



1b) i

A	B	C	#1	#2	#3	x	
0	0	0	0	0	0	0	✓
0	0	1	0	0	0	1	✓
0	1	0	0	0	0	0	✓
0	1	1	0	1	1	1	✓
1	0	0	0	0	0	0	✓
1	0	1	0	0	0	1	✓
1	1	0	1	0	1	1	✓
1	1	1	1	1	1	1	✓

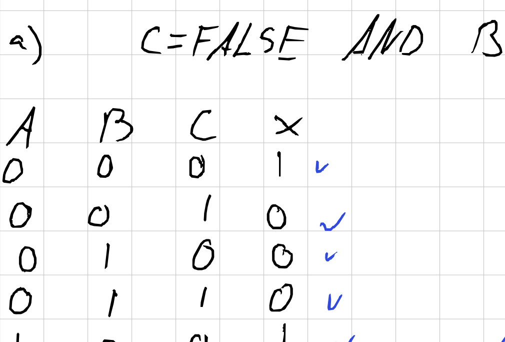
✓

ii If C is TRUE x will always be TRUE. ✓
Therefore the AND gate comparing C and B is redundant. ✓
2/2

3. a) C = FALSE AND B = FALSE

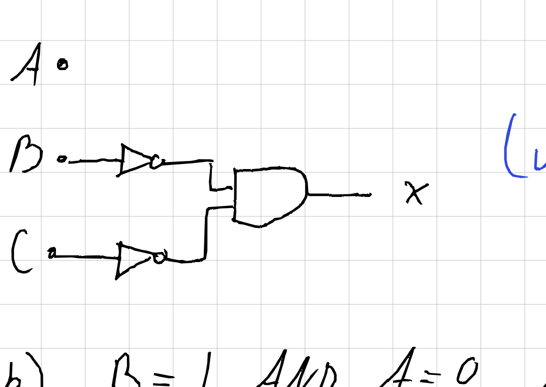
A	B	C	x	
0	0	0	1	✓
0	0	1	0	✓
0	1	0	0	✓
0	1	1	0	✓
1	0	0	1	✓
1	0	1	0	✓
1	1	0	0	✓
1	1	1	0	✓

✓



(✓) book is different but still correct? xD

b) B = 1 AND A = 0 AND C = 0



(✓) ?
still not wrong?!

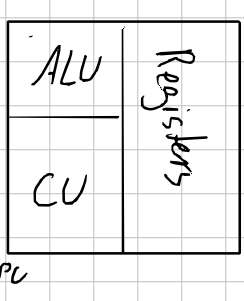
4. a)

4.1 Central Processing Unit

The CPU controlls <u>manipulation of data</u>
CPU == Processor

CPU Architecture:

Arithmatic/Logic Unit (ALU)
- performs all calculations/opperations

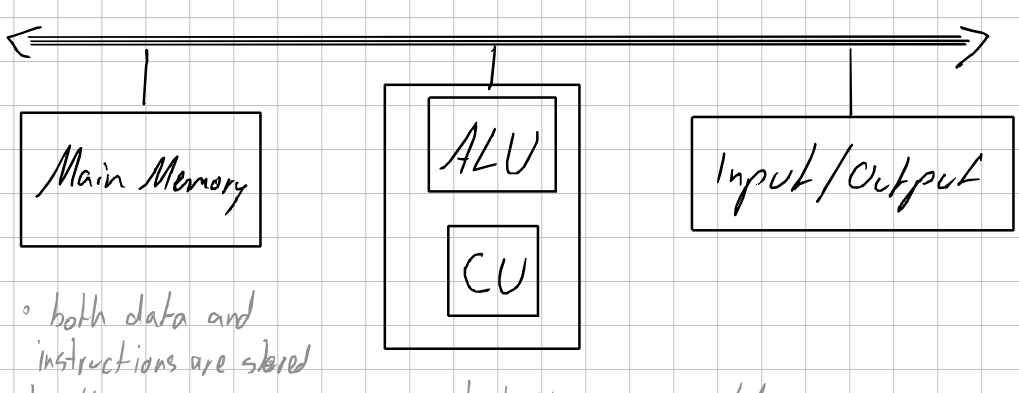


Registers
- super-high speed close stores of data inside the CPU

Control Unit
- coordinates all activities of the CPU
- Checks that instructions are handled correctly

Von Neuman Achitecture:

- all modern computers are based on this architecture



- both data and instructions are stored together
  = <u>stored program concept</u>
- Instructions are executed sequentially → one instruction at a time is fetched from memory and executed
- von Neumann bottleneck refers to the issue that a bus as fast as the CPU is required
- Computers use a <u>system clock</u> to syncronize circuits
  ↳ not time but delivers a constant pulse ⎍⎍⎍
  ↳ located on motherboard
- Each instruction takes a certain amount of cycles/tickets

Fetch-Execute Cycle:

Fetch:
CPU retrieves next instruction from main memory

Decode:
The instruction is broken down into smaller components
↳ what opperation should be done
↳ which data should be used

Operator, operands: ADD R1 R5 | R5 is read and placed into R1

Execute:
- CU activates required circuits for data transfer and the operation inside the ALU
- The output is stored in a register
- Data might be read/written to/from main memory

If the clockspeed is higher, this cycle happens faster

General Purpose Registers:

Registers which can be used by programmers. If there is only one general purpose register, it has to be the ACC. This is only in simple CPUs

Accumulator (ACC):
- holds the result of an operation performed by the ALU

Special Purpose Registers

Program Counter (PC):
- holds the memory address for the next instruction (highly important)
- increments after being fetched (usually)
  ↳ usually instructions are being stored sequentially, not allways

Memory Address Register (MAR):
- stores the memory address of the data which the CPU needs to access

Memory Data Register (MDR): aka. MBR (B = buffer)
(memory buffer register)
- holds data which is being transfered between CPU and memory location

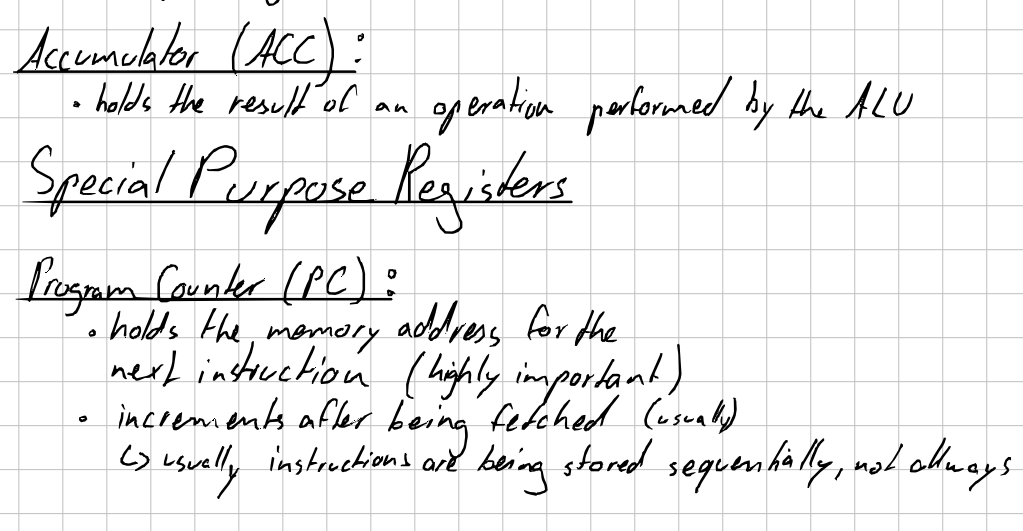
linked

When <u>reading</u>: data at the address in MAR is stored to MDR

When <u>writing</u>: data inside the MDR is stored to the address in MAR

↳ might seem redundant, BUT... act as buffer registers and compensate for different speeds of CPU and RAM

More Accurate CPU Architecture



minimum number of components required
arrows indicate <u>data: address, instruction, or value</u>

Clocks:

<u>Internal clock:</u> controlls all cycles inside the CPU

<u>system clock:</u> controlls all activities outside the processor

<u>clock speed</u> refers to the frequency

Current instruction register (CIR):
- stores current instruction while it's being decoded and executed

Index Register (IX):
- holds a single value that describes a relative range of addresses, starting from a base address that is stored somewhere else.
  ↳ used for 'Index addressing'

Status Register (SR):
- stores single bits of helpful data that helps the CPU make decisions
  ↳ e.g. "the result is negative == 1", or "overflow == 0"
- consists of multiple bits where each bit is considered a boolean <u>flag</u>, and each bit has a predetermined meaning.

! All special purpose registers, <u>except status registers</u> contain <u>one value only</u>

Examstyle questions

1. a) Accumulator ✓

   b) i) when reading, the MAR stores the location of what is to be read, when writing, MAR stores the address of where data should be written to ✓
   ii) memory addresses (✓) → in binary
   iii) PC (✓) Program counter

   c) i) it stores the current instruction while it's beeing decoded and executed
   ii) operation instructions in binary
   iii) Memory buffer data register (MBR / MDR)

   d)

MAR	MDR
stores memory addresses	stores actual data
recieves data from PC	recieves data from outside the cpu
sends data to Control Unit	sends data to other registers

4.2 System Architecture

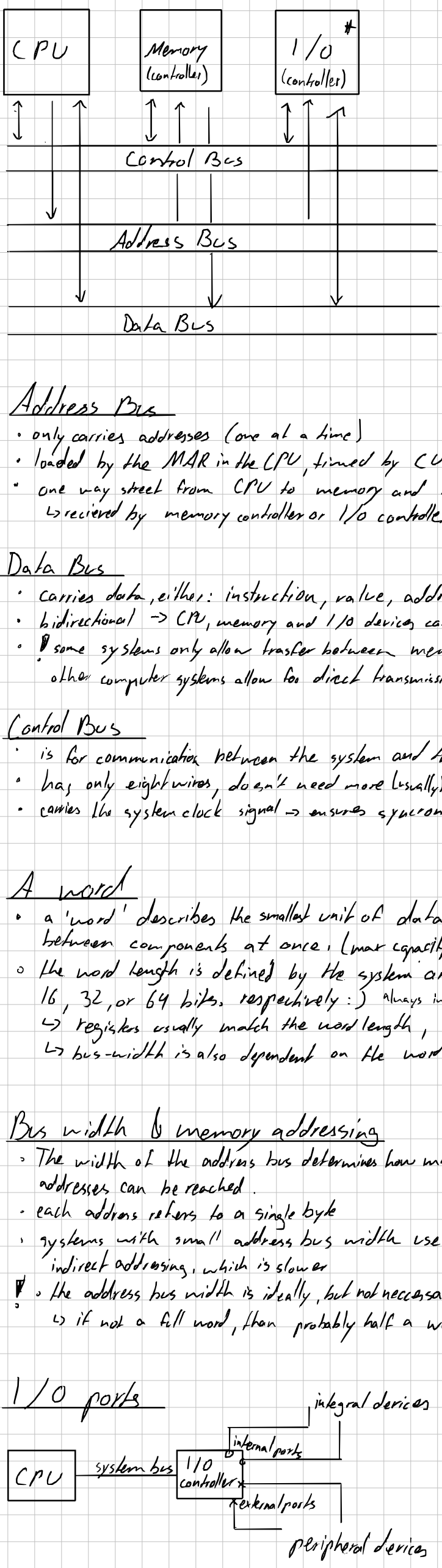
CPU performance

- clock speed (measured in Hz = cycles per second)
  - ↳ a given number of instructions is executed every cycle
- Cache Size
  - ↳ having to access slower RAM takes longer, having more cache is beneficial
- Cache Type: there is a trade-off between latency and the hit rate
  - ↳ 'hit-rate': how fast data is found → highly dependent on size
  - ↳ speed vs. size tradeoff
  - ↳ solution: multiple levels of cache
- Number of cores: today's chips have multiple processing units (CPUs)
  - ↳ double cores doesn't quite mean double the actions/speed, it's more complicated
  - ↳ multiple cores share a larger cache, but each also has their own

System Bus

- parallel transmission component
  - ↳ many parallel wires transmit an array of bits (at the same time)
- there are other busses than the system bus located inside the CPU
- System bus connects the CPU to memory and input-output devices:
  - ↳ address bus → memory address
  - ↳ data bus → actual data
  - ↳ control bus → e.g. timing of the system

I/O = input output



Address Bus

- only carries addresses (one at a time)
- loaded by the MAR in the CPU, timed by CU
- one way street from CPU to memory and I/O devices
  - ↳ recieved by memory controller or I/O controller

Data Bus

- carries data, either: instruction, value, address
- bidirectional → CPU, memory and I/O devices can load / recieve data
- ! some systems only allow trasfer between memory and I/O through CPU, other computer systems allow for direct transmission

Control Bus

- is for communication between the system and the Control Unit in the CPU
- has only eight wires, doesn't need more (usually)
- carries the system clock signal → ensures synchronization of components

A word

- a 'word' describes the smallest unit of data that can be transfered between components at once. (max capacity of CPU)
- the word length is defined by the system architecture, usually 16, 32, or 64 bits, respectively :) always in byte increments
  - ↳ registers usually match the word length,
  - ↳ bus-width is also dependent on the word length

Bus width & memory addressing

- The width of the address bus determines how many individual addresses can be reached.
- each address refers to a single byte
- systems with small address bus width use special techniques for indirect addressing, which is slower
- ! the address bus width is ideally, but not neccessarilly the size of a word
  - ↳ if not a full word, than probably half a word.

I/O ports

- I/O controller, aka device controller manages connections between CPU and I/O devices

Universal Series Bus (USB)

- a hierarchy is supported by the USB standard
  - ↳ The computer is at the root of this hierarchy
- Computer can connect 127 devices

Special Multimedia Ports

- Video Graphics Array (VGA) port
  - ↳ for second monitor, no sound
- High Definition Multimedia Interface (HDMI) port
  - ↳ video + audio

other Bus Types

Exam style questions

2. a) Data bus: carries data between memory, I/O devices, and CPU (specifically MDR) missing 'value, address, instruction', bidirectional
Address bus: sends memory addresses from the MAR to memory and I/O controller ✓ unidirectional
control bus: connects the Control unit inside the CPU with the system components. ✓ This transmission e.g. includes the system clock signal. ✓ bidirectional

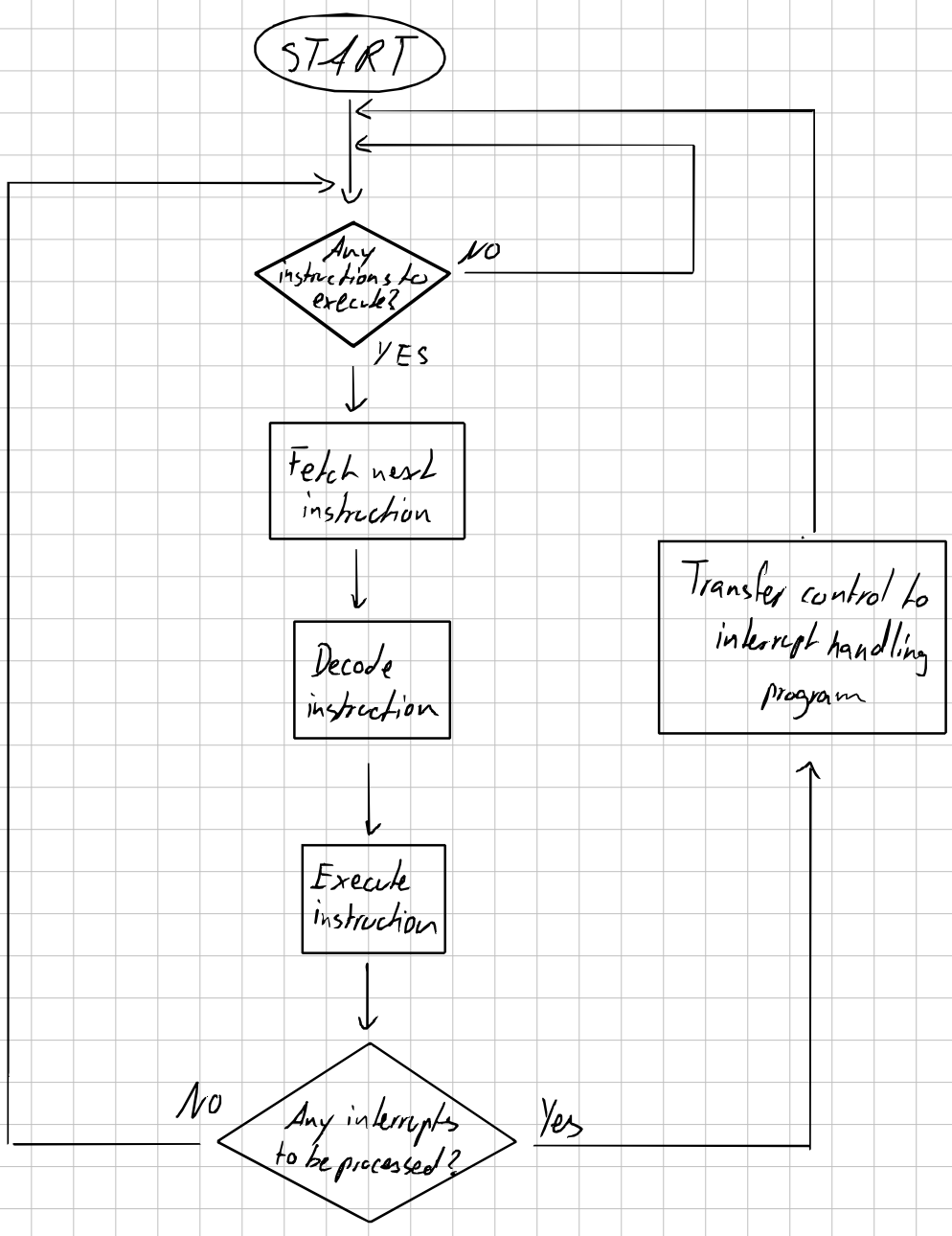
b) i Data bus: preferably, but not always the length of a word, whatever the MDR can handle at once.
Address bus: preferably, but not always the length of a word whatever the MAR can handle at once
Control Bus: 8 bits (1 byte) doesn't need to carry much
Defined by the number of bits carried simultaneously

ii Control bus has the least width as it doesn't carry much data

iii now there are significantly more addresses, allowing for the use of pure direct addresses. This will significantly speed up the system. given of course the MDR scales. This would be 2^{64} addresses which is a lot of TB. 32 bits would only allow for 2^{32} addresses which wouldn't cut it for most systems

4 a)

4.4 Fetch-Execute (F-E) Cycle



Example when program already running: (each point one cycle)

(
- Program counter already stored next address)
- address from PC copied to MAR
- At the same time:
  - data at address in MAR is fetched into MDR
  - PC is incremented
- Instruction stored in MDR is transfered to CIR
- Instruction is interpreted by CU and ALU might be activated if arithmetic is involved

Note:
- The system clock is used, as data from outside the processor is fetched.
- PC isn't always simply incremented: e.g. when there is a 'jump instruction' than an entirely different address might be loaded

Register transfer notation

- Describes operations between registers
- e.g. the fetch part of the F-E cycle

$MAR \leftarrow [PC]$

$PC \leftarrow [PC] + 1$; $MDR \leftarrow [[MAR]]$

$CIR \leftarrow [MDR]$

Notation:
- first item is data destination
- abreviations are used (e.g. MAR)
- second item is the data's definition (where it came from)
- square brackets indicate a register's data is being moved
- arrow underlines flow of data
- tasks seperated by a semicolon happen at the same time
- The double brackets mean that not the address in MAR but the data at the address in MAR is being transferred.

Interrupt handling

- some reasons for interrupts:
  - fatal error
  - user interaction
  - need to start I/O processing
  - a timer signal
  - hardware fault

- interrupts have different priorities
- some interrupts will erase the current process, others just pause it
- a way for the processor to identify the type of interrupt would be an 'interrupt register' (IR)
  ↳ works similar to status register with flags.

How interrupts are hadled:
- content of program counter and other registers saved to memory
- an 'interrupt service routine' (ISR) is choosen according to the problem.
  ↳ start address loaded into PC
- at end of execution, check if there are still other interrupts
- further interrupt are processed
- if no interrupts, load original data from memory, resume

Exam-style questions: 3, 4 b, c, d, 5

3. a) On the left there is the destination. On the right is the data description (from where it is transfered). The arrow underlines the flow of data. Single brackets mean that the information of the register in brackets is being moved. Double brackets mean that the data stored at the address which is located in the register is transfered. A semicolon deviding two operations in the same row indicates that they occur simultaneously.
First the address stored in the PC (program counter) is transfered to the MAR (memory address register). Then the PC is incremented while the data stored at the address inside the MAR is transfered to the MDR (memory data register). Finally the data from the MDR is loaded onto the CIR (current instruction register)

b) When data is fetched from the address stored inside the MAR, the address bus is used to send the according address to a memory controller. The memory controller responds with the matching data and sends it over the data bus to the CPU. MRD

4 b) i The address stored in the program counter gets incremented.
ii The data at the address stored in the memory address register gets copied into the memory data register.
iii The instruction stored in the memory data register gets copied to the current instruction register

c) not in chapter? LDD 35 loads address 35 into MAR, the whole F-E cycle follows → data stored at the address is loaded into the accumulator

d) i An interrupt is a signal which causes a pause in the current process that allows for multiple programms running at once, despite the fact that only one process can be executed at a time
: pause the program
ii Store register content to memory
Identify the interrupt and load the matching interrupt service program.
Execute the ISP
Evidently check if any more interrupts are waiting
Resolve any other interrupts
load back original memory into registers

5. Program counter holds the next address to be fetched
address is loaded into MAR
contents of the address is stored to MDR; Program counter is incremented
data is copied from MDR to CIR

1. B
2. D, A, C

4.4 Machine Code & Assembly

- CPU only recognizes machine code (binary)
- machine code consists of a sequence of instructions
  - Instructions must contain an '<u>opcode</u>', i.e. an operator
  - Instructions can include zero to three <u>operands</u>
- Instruction sets are specific to their processor type
- Instructions for different processors will be similar, but their machine code looks different

This must be defined for each individual machine code instruction:
- length in bits/bytes of entire instruction
- number of bits for the opcode
- number of operands (they cover all remaining bits)
- is opcode in the front or in the back?

e.g.

Opcode			Operand
4 bits	2 bits	2 bits	16 bits
operation	Address mode	Register addressing	

❗ note: it's good if the operand length and the address bus width match, as the opperand might be an address

TL;DR : Operand width = address bus length

Decoding + register transfer notation

- after the instruction arrived at the current instruction register (CIR), the opcode part is transfered to the control unit (CU) and decoded:

$CU \leftarrow [CIR(23:16)]$

- bits 16 to 23 are copied, denoted by (:)
(note that 23 is first)

Assembly Language

- writing machine instructions is the most efficient at runtime but virtually impossible for serious development
- assembly is a low-level language strongly tied to the actual machine code. Opcodes are written short abriviations. (as <u>mnemonics</u>)
  - ↳ machine code is specific to the processor
  - ↳ assembly too is specific to the processor
- for each machine code instruction there is an equivalent machine code instruction

What Assembly Adds

- comments: makes the program more comprehensive
- macros: like functions, abriviations for entire code snipets
  - ↳ snippet is copy-pasted to all locations by assembler
- directives: directions for the assembler, e.g. on how to handle memory, or how to compile something
- subroutines: basically functions, during runtime the program will jump to a different code snippet and then return
- system calls: when the program requests services from the system
  - ↳ e.g. writing to a screen, reading files or exiting the program

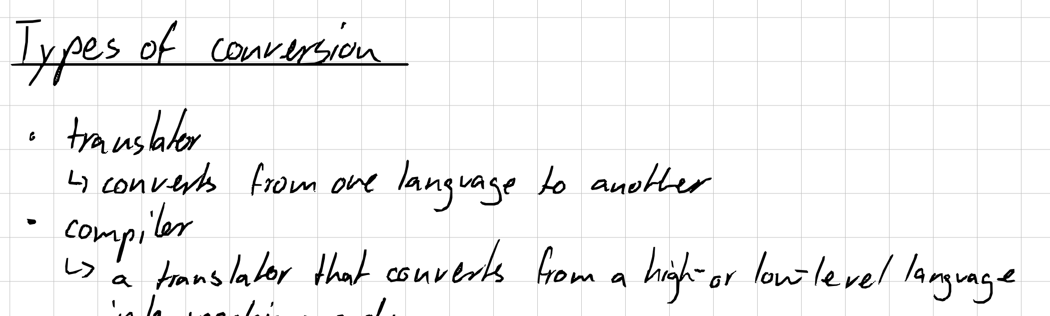
- symbolic names for variables (if wanted)
- denary and hex-representation of variables
  - ↳ #49 ; #B1001 ; #&H5C7

≙ value & ≙ Hex
B ≙ binary

Assembler:

- The compiler that converts the already rather specific assembly code to even more specific machine code (specific to the processor that is)

~~Programming Languages in Perspective~~



Types of conversion

- translator
  - ↳ converts from one language to another
- compiler
  - ↳ a translator that converts from a high- or low-level language into machine code
- assembler
  - ↳ a compiler specifically translating from assembly into machine code

Types of Addressing

Symbolic addressing:
- Data locations are refferenced by variable names, e.g. 'TOTAL'.
  - ↳ the programmer doesn't have to worry about the actual location where the data is stored.

Absolute addressing:
- Data locations match their actual memory address
  - ↳ all variables are numbers, which are usually rather long and incomprehensible e.g. 249782

Relative addressing:
- a special-function base register (BR) is loaded with the <u>base address</u>, which is the address of the first instruction (which is 0)
- All other addresses are given relative to the base address
  - ↳ numbers are significantly smaller and easy to comprehend
  - ↳ program is independent from its location but control over memory placement is maintained.

Examples for all three can be found on p. 94f.

How a Two-Pass Assembler works

- used to handle programs with recursion or symbols
  - ↳ symbols or functions might be refferenced that weren't read yet
- The assembly code is read line-by-line

<u>First pass:</u>
- removal of comments
- replacement of macros by code snipets
- removal and storage of directives (to be implemented later again)
- creation of a symbol table (<u>resolving forward refferences</u>)

Use of symbol tables:

Symbol	Offset
Symbol 1	+ 15
Symbol 2	+ 13
⋮	⋮

- On the first pass a symbol table is filled with all variable names encountered
- the address (might be absolute, usually rellative) is added to the table when encountering the symbol value
- <u>On the second pass:</u>
  - the symbol table and an opcode lookup table are used
    - ↳ opcodes are replaced by binary
    - ↳ all symbols are replaced by binary addresses

Memory in Runtime

- empty storage spaces should be reserved for all variables
- if the memory is being loaded into memory to be executed, changes to the addresses will have to be made

Addressing Mode

- two bits of the opcode are allocated to the addressing mode.
  - ↳ this allows for four modes:

Immediate:
The operand is the exact value used for the instruction
- ↳ e.g. SUB #48
- ↳ There are three options to define this address
  - #49 : denary value 48
  - #B00110000 : binary equivalent of 48
  - #&30 : hexadecimal 48

Direct:
The operand is an address that refers to the data that should be used for the operation
- ↳ e.g. SUB AGE or SUB 17

Indirect:
The operand stores an address at which a second address is stored. The data at the second address is then used for the operation
- ↳ this allows for larger addresses further out in the system to be used

Indexed:
similar to relative addressing: the address is given relative to a base address stored in the index register (IX)

Exam Style Questions 2a, 3

2a)
Macros: Act as a sort of function where a snipet of code is refferenced at multiple locations
Comments: Help to make the code more understandable
Directives: Give instructions on how the code should be compiled, memory addresses should be handled, or etc.

3a) The program takes inputs from a keyboard

b) no Idea, haven't done

c) CHARACTER
START
OUTPUT

~~10~~ 10 → 1010
~~8~~ 4 → 0100
~~8~~ ??? didn't do → 0111
7
count from 0 bro
also the table is always in binary

4.6 Assembly Instructions

Types of Addressing: (recap of 4.5)

- Immediate Addressing: operand is value
  ↳ e.g. LDD #4
- Direct Addressing: operand holds the memory address for data used
  ↳ e.g. LDD VARIABLE or LDD 216
- Indirect Addressing: Holds an address that holds a usually larger address which holds the data used
  ↳ e.g. LDD VARIABLE or LDD 216 (same)
- Indexed addressing: operand is an offset of the value currently stored in the index register (IX)
  ↳ e.g. LDD 5
- Relative Addressing: operand gives address relative to its own instruction position. (positive is down, own instruction not counted)
  ↳ e.g. (line 5) LDD 5 → data from line 10 loaded

Instructions

! conflict with previous page

Data movement

LDM	#denary_value	• Loads number into ACC | immediate
LDD	<address>	• Loads number into ACC | direct
LDR	#denary_value	• Loads number into IX | immediate
LDI	<address>	• Loads value at address to ACC | indirect
LDX	<address>	• Loads value to ACC | indexed
MOV	<register>	• <register> ← [ACC]
STO	<address>	• stores value of ACC to address

Input & Output

IN	(no operand)	• stores the next ASCII character typed on a keyboard to ACC
OUT	(no operand)	• displays the value in the ACC as ASCII on the screen (assumes value is ASCII code)

Comparisons and Jumps

JMP	<address>	• jumps to the instructions at the given address
CMP	<address>	• compares contents of ACC with contents at address
CMP	#denary_value	• Compares value of ACC with number
CMI	<address>	• compares contents of ACC with contents at second (indirect) address
JPE	<address>	• following a compare instruction, if True jump to address
JPN	<address>	• following a compare instruction, if False jumps to address
END	/	• returns the control to the operating system

Arithmatic Operations

ADD	<address>	• adds value at address to value at ACC
ADD	#denary_value	• adds value to value at ACC
SUB	<address>	• subtracts value at ACC by value at address
SUB	# denary value	- subtracts value at ACC by value
INC	<register> !	• increments either ACC or IX
DEC	<register>	• decrements either ACC or IX

Shift Operations

LSL	#n	• Bits in ACC shifted left by n bits, most significant bit stored in carry bit in status register
RSL	#n	• Bits in ACC shifted right by n bits, LSB saved as carry bit in status register (SR)

! empty bit is replaced by a zero, last bit to move out becomes carry:

eg. ACC: 1 0 1 1 LSL #2

ACC: 1 1 0 0 carry: 0

Multiply / Devide by two

- unsigned binary numbers can be multiplied / devided by two

multiply: shift left by one (MSB must be '0' !)

divide: shift right by one (remainder can be found in carry bit)

Cycle shift

- the bit that got pushed out moves into the carry and reemerges on the other end in the next step

ACC: 1 0 1 1 → ACC: 0 1 1 0
carry: 0 carry: 1

↓

ACC: 1 1 0 1
carry: 0

Arithmatic Shifts

- Basically logical shifts but specifically intended for multiplication and division by two
  ↳ carry bit always stored, unlike for some logical shifts

Bitwise Logic Operations

AND	#Bn / <address>	contents of ACC compared with binary value or value at address.
XOR	#Bn / <address>	
OR	#Bn / <address>	all can be used with two operands or with one and the ACC
NOT	(#Bn / <address>) (← not in book)	

- Logic operators are sometimes called a 'mask' as they may only affect specific bits
  ↳ consider 1110000 OR 0001110

Operations with multiple bits:

0 0 0 1 1 AND
0 0 1 1 0

=> 0 0 0 1 0

Note!: 0 0 0 1 1 AND
1 1 0

not allowed in most assembly languages

length has to match!

Status Flag Naming Convention

Z	zero flag → was the result zero?
N	negative flag → was the result negative?
C	carry flag
V	overflow flag
I	interrupt flag
⋮	

Trace Tables

- help to understand actions of a program or test for mistakes in a dry run

Rules:
- for each column of the program a matching row
- columns are individual memory addresses
- a value is only entered when it is new
- if there are jumps, then the program counter is also traced!

Example:

Problem (currently executing line 99, 100 is next)

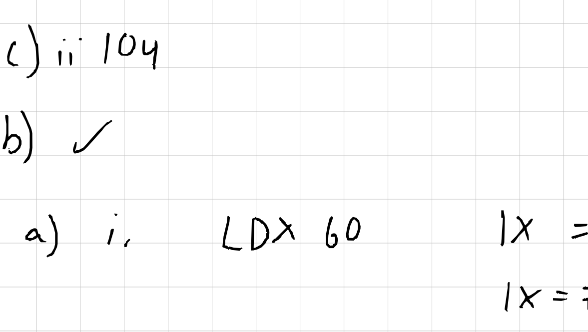
```
100: LDD 200        200: 0001
101: INC ACC        201: 0011
102: ADD 201        202: 0101
103: CMP 202        203:
104: JPE 106          ↑ on purpose
105: DEC ACC
106: INC ACC
107: STO 203
```

PC	200	203	line executed (not required)
100	1000		99
101	0001		100 ← program starts here
102	0010		101
103	0101		102
104			103
106			104 ← JPE evidently overrides PC
107	0110		106
108		0110	

value precists but isn't shown ↳ bc it stays the same

Exam Style Practice

1. a) i



ii 110

b) ii 112

c) ii 104

2. b) ✓

4. a) i. LDX 60 IX = 0000 1000 (8 4 2 1)
IX = #8

LDD 60 + 6
LDD 68

68: 0100 0101
↳ ACC: 0100 0101

ii IX: 0000 1000 | -1

IX: 0000 0111

4.8 Monitoring & Control Systems

Monitoring Systems

- creates a record of a system's condition over a period of time.
- Sensors are used to gather data
  - ↳ sensors only output information, the control system is the one to take action
  - ↳ a sensor lacks intelligence

Control Systems

- consist of a monitoring system and one or more actuators connected to controlling devices

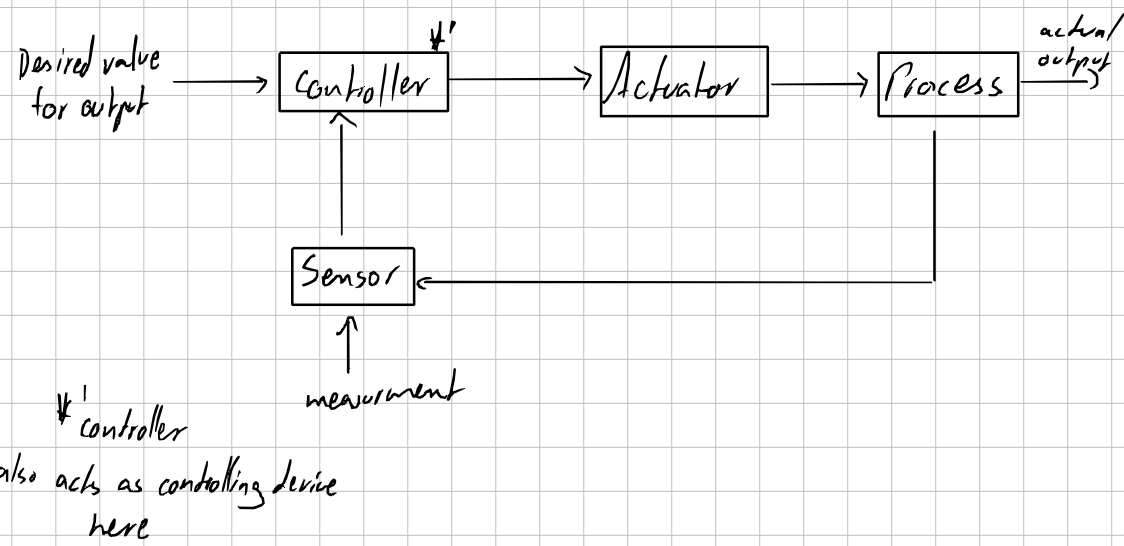
Actuators: Electrical motors that act upon an electrical signal
- ↳ adjusted by controlling devices, e.g. a flap that restricts airflow on a plane wing
- ↳ examples: Electric motors, valves, relays

Controlling Devices: Take in a control signal from a controller (computer / microprocessor) and turns it into actionable signals for an actuator
- ↳ e.g. motor driver, relays
- ↳ not PID controllers, etc. as they would decide action

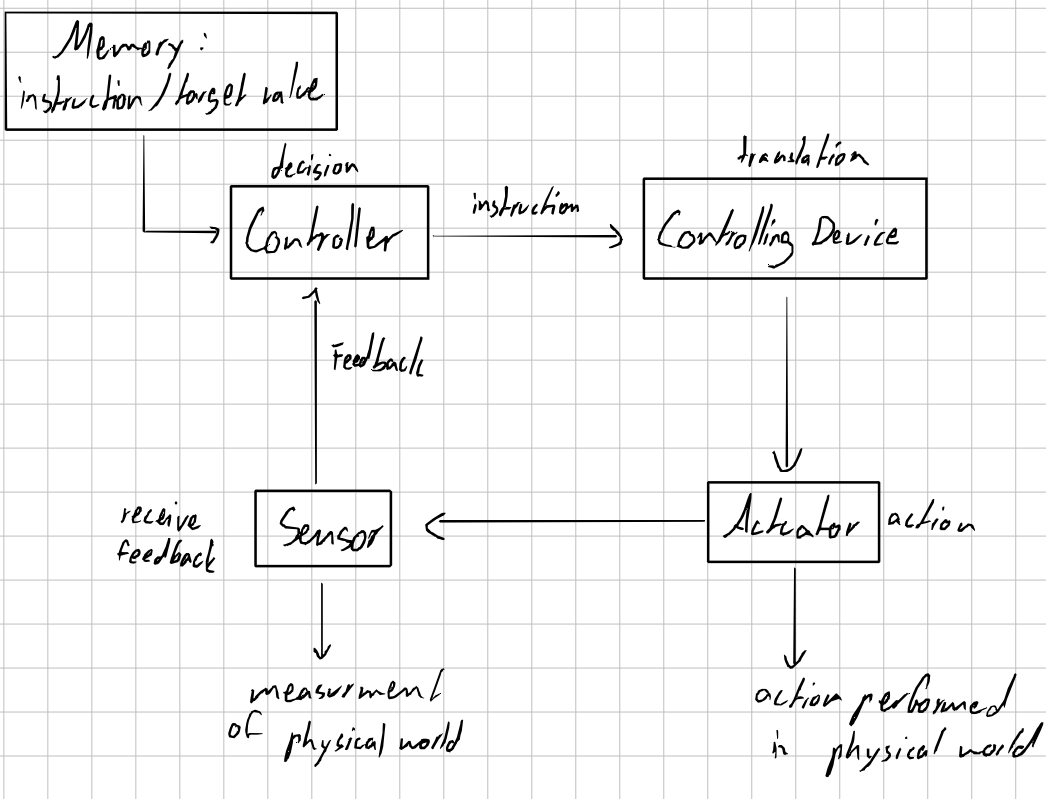
Controllers: The brains of the operation which decide what action to take.
- ↳ e.g. a computer or microprocessor
- ↳ send signals to controlling devices / directly to actuators

Sensors: Measure data either continuous or in intervals, output is usually an analogue signal
- ↳ often have a build-in analogue to digital converter

Closed-Loop Feedback Control System



My version:



4.9 Bitwise Operations

Control systems often use closed feedback loops where a program checks for a combination of flags set by a sensor. Bitmanipulation is used to interpret those signals.

TL;DR: control systems use flags set by sensors

Flags are set by assigning statements.
This is the syntax (pseudo):

IF SensorDifference / > 0 THEN SensorHighFlag ← TRUE
IF SensorDifference < 0 THEN SensorHighFlag ← FALSE

Flags might be stored in the first few bits of an operand or in a designated byte.

Bit Manipulation on Flags:

use: OR, AND, XOR, NOT

AND #B 0000 ← delete all bits

OR #B 0010 ← set flag regardless of previous state

XOR #B 0100 ← Toggle flag (0 → 1 ; 1 → 0)

Check if Flag is up
AND #B 0010 ← isolate Flag in ACC
CMP #B 0001 ← compares ACC with content at address 0001

Exam-Style Tests

ACC = 0101
XOR #B 1100
ACC = 1001

XOR actually flips all bits where the operand contains a 1

2, 3c, 4

2. a) AND #B0000 0000

b) XOR #B0000 0100 / OR #B0000 0100

c) XOR #B0011 0000

3c) i The 1 in the bit at location 5 indicates that the measurement is from location 5. That the bit in location 0 is 1 means that the measurement has been processed.

ii
0100 0000 11111011

-5
5: 00000101
-5: 11111011

5.1 System software - OS

System Software:
- programs that help the computer run
  ↳ e.g. boot loader or OS

Application Software:
- programs that perform specific tasks to serve the user
  ↳ e.g. Browser, Computer Games, etc.

Operating System (OS)

- The software that manages the computer system and links other software with hardware

=> Provides an environment for other programs

Basic Tasks:
- managing resource allocation
- providing access to hardware (only OS has direct access)
- provide a user interface (e.g. CLI or GUI)
- Provide a layer of security

User - System Interface
- provides the user with access to the computer's resources
- Allows for input and output

Comand-Line Interface (CLI):
- text based terminal
- simple but inconvenient

Graphical User Interface (GUI):
- E.g. a desktop: highly functional
- high level of abstraction
- resource intensive

Program - Hardware Interface
- The OS bridges abstractly written programs with the specific underlying system
- The OS enables programs to be run
  ↳ by e.g. loading it into memory

Resource Management
- A running program is called a 'process'
- The OS performs multitasking by scheduling and interrupting multiple processes
  ↳ There are usually more processes then available CPUs
- The OS resolves conflicts between processes
  ↳ e.g. if both require the same resource

Memory Management
- Memory protection: prevent multiple processes from using the same memory
- Optimize resource usage: The memory organization scheme optimizes how limited memory is used for best performance
- Decides Memory Priority: Which processes should be in memory, and where

Device Management
- The OS installs device driver software
  ↳ Drivers remove the complexity from communicating with devices
- Controlls the device usage by processes

File Management
- OS determines:
  - file naming conventions
  - directory structures
  - access control mechanisms

Security Management
- data recovery -> creates backups
- data privacy
- makes intrusion difficult

Error Detection & Recovery
- is able to shut down faulty processes
- provids error diagnostics
- orderly shuts down system on major issues and prevents data loss.

Exam style questions

1. a) i. Graphical user interface, Command line interface (GUI; CLI) ✓ ✓ on spot bro

   ii Keyboard & Mouse; only keyboard ✓ ✓ ✓ mouse = pointer
   touch ~ you got all ✓

   b) Error Management: The computer gracefully shuts down on major errors to prevent data loss, and provids diagnostics for any errors that occur. ✓

   File Management: The OS determines file extensions, directory structures and how files are controlled (accessed) ✓

5.2 Utilities & Libraries

Utility Software

- analyzes, configures, optimizes, or maintains the system
- can be called by other software or the OS
- might come with the OS or is installed later

Tasks involve e.g.: encryption, compression, anti-malware, etc.

Book example: Hard Disk Formatter and Checker

- formatting (clearing) a disk
- setting up the file system and pointers (table of contents)
- partitioning the disk
- disk repair → mark 'bad sectors' and recover data

Hard Disk Defragmenter

- reorganizes files so that data from the same file is closer
  ↳ faster to access
- defragmentation means making the data for each file more continuous and less spread out
- becomes impossible if disk is too full

Backup Utility:

- schedule backups
- create backups only on changes

Program Libraries

=> contain many little programs that are used as reliable subroutines

- helpful as the coder doesn't have to build everything from scratch

Linking Libraries to Code:

Embed libraries in source code

- libraries are added to the source code and compiled alongside it
  ↳ likely results in many copies of same libraries in memory

Dynamic Linked Libraries (DLL)

- The library is seperate from the compiled source code
- It's loaded into memory and executed by refferences in the code
- Many programs can use the same library simultaneously
  + less storage & memory required
  + easy to update libraries
  − programs are dependent on external library
    ↳ version errors, etc.

Exam Style 2, 5, 6

2a) i) 1. Hard Disk formatter utility (✓)
2. Hard Drive defragmentation utility (defragmenter) (✓)

ii. 1. Used to clean all data of a disk and recreate fresh look-up tables and file structures ✓
2. Used to increase the access speed of data on the disk by by resolving any fragmented files. ✓

b) i They are reliable options ✓ that replace code the developer would else have to create himself. more efficient, already tested
When using dynamic linked libraries, ~~memory and storage are also saved.~~

ii Mathematical / statistical processing library ✓
Computer graphics calculations? (mb GUI element library)

5. a) A Disk formatter is needed to make a disk useful and ~~sometimes even~~ compatible with the system: ✓
- It clears all existing data on the disk ✓
- It creates lookup tables for the data in a way that fits the specific system ✓
- It partitions the disk if needed ✓

Disk is devided into sectors >

b) Hard Disk Defragmenter: Bundle files into continuous data and increase speed ✓
Disk Repair Utility: Marks bad sectors and recovers lost data ✓
File Compression Utility: Decreases the file size and therefore increases the free space available on the disk ✓

6. a) 1. He doesn't have to write the code himself. ✓
2. The code is already known to work reliably ✓

b) i 1. Less memory and storage are used (✓) ...explain: because the same library is used multiple times
2. When updating a library it happens for all, as the library is centralized ✓ no need for re-compilation

ii If the library changes unexpectedly and contains an error, this will also cause an error in the code, even though the source code stayed the same

Abstraction: Faulty changes in libraries might result in unforseen errors in the future. ✓

5.3 Compilers & Interpreters

Interpreter

- Source code is compiled at runtime
  - ↳ code gets translated and then executed, for every line
  - ↳ if an error occurs / is found, the program halts emideately
- code is translated into <u>intermediate code</u>: e.g assembly or byte code
  - ↳ (not machine code!)
- Both source-code and interpreter are needed at runtime

  => code has to be re-interpreted for every execution
  - ↳ generally slower at runtime, faster for development

Compiler

- reads program line by line and converts it to <u>intermediate code</u>
- if an error is found it's recorded, program continues
  - ↳ compiler spits out list with all errors at the end
- If no errors occured, intermediate code is converted to object code.

<u>On runtime</u>

- program is extreamly fast
- only compiled machine code needed
  - ↳ source code and compiler aren't needed
  - ↳ source code stays secret

<u>Object code</u>: Platform specific machine code that has yet to be linked with files, libraries or other object code, and made executable.

<u>Intermediate Code</u>: Low-Level platform-independent code which is not yet fully executable (e.g. Bytecode)

Interpreter Pro-Con

+ fast for development: errors found fast
- errors are only found at execution of the faulty line of code
- the source code has to be shared with all users
- the interpreter is required to run the program

Compiler Pro-Con

+ the executable can distributed on its own
+ source code stays secret
+ only the object code required
+ faster on runtime
- less secure as it might contain hidden viruses
- slower at development -> recompile again for every error

Java

- Java is extra :)
- source code is converted to universal 'Java byte code'
- byte code is interpreted by system-specific 'Java Virtual Machine'
  - ↳ Java byte code works for all Java vms

IDEs

Integrated Development Environment

<u>Prettyprinting</u>:

- more comprehensible code representation that <u>automatically</u> occurs:
  - color-coding
  - formatting, etc

<u>Context-Sensitive Prompts</u>:

- code recommendation
- autocomplete
- 2025 edit: vibe coding xD

- <u>Dynamic Syntax Checks</u>:
- <u>Expanding & collapsing code blocks</u>
- <u>Debugging support</u>
  - ↳ breakpoints: program stops when reached
  - ↳ line-by-line execution (manual)
  - ↳ examine current program state -> variables, etc.

<u>Exam Style</u> 3, 4

3. a) An assembler exclusively translates assembly code to machine code, while Interpreters and Compilers translate high-level languages to intermediate or object code.

b)

i	Interpreter	Compiler
		(doesn't execute)
	• Compiles at runtime	• compiles to an executable first
	• Stops when encountering an error	• creates a list with all errors found
	• requires source code and interpreter at runtime	• only requires object code to execute the program
	produces intermediate code	produces object code (a form of intermediate code lol)

ii: + Interpreter is faster during development -> errors found fast
- errors only found when specific line is executed
- slower at runtime as it is compiled while being executed
= Interpreter and sourcecode are needed to execute the program

iii: A Java Virtual Machine is used. (to run the code)

4. a) • Prettyprinting:
- automated caller coding
- comprehensible text formatting (automatic indentation)

• Context-sensitive prompts:
- code completion and recommendation (at current insertion point)

• Dynamic Syntax Check
- IDE checks for syntax mistakes (constantly) after a line has been typed

• Debugging
- The process of finding and correcting mistakes in the code, often assisted by tools offered through the IDE

b) • Breakpoints: markers that allow the program to be stopped at a specific line
• manual line-by-line execution: The coder can cycle through every line manually.

6.1 Data Integrity, Privacy and Security

Data Integrity

- The requirement for data to be accurate and up-to-date
  ↳ 'accurate' might mean unchanged, e.g. when transferring a file

Data Privacy

- keeping data safe (private)
- individuals generally have control over who has access to their data
  ↳ which of their data is being shared
- organizations also have a right to data privacy
- Data protection laws enforce data privacy
  - major focus on protecting individuals from cooperations
    ↳ data only used for purposes agreed upon
    ↳ organizations have to ensure data privacy and integrity
  - Those laws can't garantee anything

Data Security

- The requirement for data to not get lost, corrupt, or change unwantedly
- Data security is a prerequisite for data integrity and privacy, as lost/corrupted data is useless
- System security is crucial for data security:
  - system executes requested tasks as long as needed
    ↳ prevent data los
  - ensures only authorized users can access the data /sys

Exam Style 4a

4.a) mistake in course - didn't cover firewalls and authentication yet.

6.2 Security

Threats to system and data security:
- human error (mistakes)
- internal problems / mismanagement
- natural disasters
- unauthorized intrusion
- installation of malware

hacker => someone that wants to gain unauthorized access to a system or data

malware => malicious software that wants to harm the system

Types of Malware

virus: tries to replicate itself into other executable programms
  ↳ boot sector virus: runs on boot
  ↳ macro virus: hides in macros, usually in Word, Excel, PowerPoint

worm: runs independently and spreads to other hosts (computers)
  ↳ a spreading virus

logic bomb: stays inactive until some condition is met

Trojan horse: replaces all or part of an initially useful program

Spyware: collects information and sends it to another system

bot: uses the infected computer to launch attacks

Introduction of Malware:

Phishing: sending a digital message from a semingly authentic source that asks for confidential information

Pharming: A mock up website that asks for your credentials while impersonating a different website

keylogger: recordes all keys pressed by the user

Sources of Vulnerability

User:
- creates weak passwords
- might fall for phishing or pharming attacks
- introduce malware:
  - attatching e.g. a foreign USB
  - opening an email attatchment
  - accessing malicious websites
  - download maliceous files

System:
- The OS usually is very vulnerable. More complex os → more weak spots
- Viruses hidden in defaut macros/dependencies of software
  ↳ will always be installed
- buffer overflow can be used to gain unauthorized access
  ↳ hacker overides permission files

Security Measures

Disaster recovery
=> restart programs fast to prevent other dependent programs from breaking down
- Disaster Recovery Contingency Plan
  = provides a packup system at any times
    ↳ packup system is called 'hot site'
    ↳ has to be remote from original system (bc. e.g. earthquake)

Parallel System Update
- used to run a system continously, e.g. web-services
- runs two systems in parallel:
  - one updating
  - one providing the service

User Authentication
- Even single-user systems should have user-accounts
  ↳ restricts easy done damage
- Users have to authenticate themselves with (usually) an ASCII password
- a good password is long and complex
- Alternative authentication methods:
  - biometric auth. (e.g face ID or fingerprint)
  - security tokens: physical hardware for identification (e.g. NFC)

Good Practice
- turn of computer when it's not in use
- enter the password in secret
- keep the password only in your mind
- organizations banning the use of portable storage devices

Firewall
- pimary defence against malware
=> a hardware device that inspects and controlls all trafic
  ↳ sometimes it's a software
- checks system addresses and IPs
- (sometimes) checks for maliceous data
- can include filters (e.g. for inapropriate websites)

Digital Signature
- a way of verifying a computer's identity (e.g. sometimes sent along emails)

Anti-Virus Software
=> software that regularly scanns the system for malware/viruses and deletes them

Intrusion Detection
- monitors physical inputs and checks for differences in performed actions
  ↳ recognizes when someone remotely takes controll or programs that act as users

Arms Race
- Hacking software and anti-malware software become increasingly sophisticated
- It's a constant race of trying to supass eachother

Security Measures for Data Protection

Data Loss
- reasons for data loss:
  - storage device gets corrupted / destroyed
  - system crashes
  - faulty deletion / overwriting of files
  - file location forgotten
- A good backup procedure:
  - full backups made regularly (e.g. weekly)
  - two or more full backups kept in storage
  - daily incremental backups
  - backups stored outside of the system
    ↳ fire- and floodproof location
- Ways of copying continuosly changing data:
  - freeze file store while copying → temporarily use memory as storage
  - use of disk-mirroring: files are always stored on two identical disks during normal operation
    ↳ one disk is remote

Restricting Access to Data
- user categories that restrict information available (e.g. employe vs. employer vs. admin)
- Authorization policies grant access rights (e.g read or write) to users

Storage Encryption
- All data on a disk can be encrypted to give intruders a hard time

---

Exam Style   2, 4a, b

2a) i
1. Data is lost
2. There is no connection to the server (✓)
3. The user isn't authorized

ii
1. Turn of the computer when not using it
2. Don't plug unknown external storage devices into any computer
3. Keep your password secret (✓)

b) i This could be a natural desaster, e.g. an earthquake or a hacker attack that compromised the main system ✓

ii A hot-site which is a backup system that takes the place of the main system if it gets compromised for some reason.
(book has different interpretation ↗)

c)
1. Regular backups could be created every week and incremental backups every day.
2. All storage could be mirrord onto a remote storage device that is locked away in a flood- and fireproof location.

4a A firewall inspects and controlls all traffic entering and leaving a network. It is able to block connections if required. → exist both as a device but also locally
Authentication means that users have to prove that they are allowed to have access to a system or file. ✓ this can be through a password, biometric data, etc. ✓ ✓

4b.) Data integrity includes the condition that the data has to be up-to-date while data security simply garantees that data doesn't get lost. ✓
Another difference is that data integrity requires data to be accurate while data security requires data not to corrupt or change unexpectidly ✓
(In both cases Data Security is contained in data integrity, as it is a prerequisite for it.)

6.3 Validation & Verification

Common types of Verification:

Data Entry verification:
- ensure what was entered matches the user's intend
- many different implementations.
  (e.g. show password, or double entry)

Data Transfer Verification:
- ensure that data received is the same as the data sent
- Ways of implementation:
  - Parity Bits: In every byte one bit states if the byte is odd or even (number of 1s). Then there are parity bits for rows and columns of bytes stating if the number of 1s for parity bits is even or odd.
  - Checksums: Somehow creating a unique value individual to the message that's processed by the checksum function. If the checksum for the sent and received data matches, then it's likely equal
    - ↳ uses e.g. hashing or bit adding

Parity Bit Explanation

	1	1	0	0	0	1	1
0	0	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	0	1	1	1	0	0
1	0	0	1	1	0	1	0
1	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1
0	1	1	1	0	0	1	0

← parity bits (top row)

↑ parity bits (left column)

Book Notes ♡

- Validation and verification are processes used to improve data integrity. → doesn't garantee it!
- Validation and verification happen during data transfer (usually?)

Validation of Data Entry
⇒ check if entered data meets specified requirements (is reasonable)
  ↳ e.g. birth date < 1000
- validation only includes following checks:
  - presence check → for entry fields
  - format check → (e.g. yy/mm/dd)
  - length check → (e.g. characters)
  - range check → (e.g. 1 to 12)
  - limit check → (e.g. > 100)
  - type check → (e.g. is integer?)
  - existance check → for files lol

Validation doesn't garantee data is correct

Verification of Data Entry
⇒ ensuring data entered is alligned with what the user intended to enter
- doesn't garantee entered data is correct
- Methods for data entry verification:
  - Double entry
  - Visual check, (showing what was entered)

Check Digit
- a very simple implementation of check sum, used for ISBN (books) and barcodes
- only used for all-number strings
- How it works:
  - numbers are added, multiplied, etc. in various ways
  - final step is deviding the number and taking the remainder
  - the remainder is the check digit
    - ↳ usually 1 to 10 (but X is used for 10)

Verification During Transfer

One-Bit Parity Check
Even parity → all correct bytes must have an eaven number of 1s

Odd Parity → there must be an uneaven number of 1s for it to get accepted

Assuming even parity: • One parity bit is added to every 7 bits of data

1 0 0 1 1 0 1 0

↑ parity bit; data (7b)

↳ makes a total of 4 (even) 1s
- If any one bit is flipped, the 1s won't be even anymore
  - ↳ the entire byte will be resent
- we can detect when there was only one error ← only one error!
- we don't know which bit is wrong

Checksum
⇒ another method to validate accurate transfer
- Sender:
  - Transmission is devided into blocks.
  - at the end of each block a few bytes are reserved for the checksum
  - All bytes in the block are treated as binary values and summed up
  - The sum is stored as the checksum.
  - The block is sent
- Receiver:
  - Adds all block bytes together and compares sum with the checksum
  - might accept block or request to have it resent.

↳ This process can't find the faulty bit position

Good to know:

Universal Product Code Checksum Verification

- UPC barcodes have 12 digits
  ↳ 11 for the code, 1 as check digit

0 36000 29145 2 ← checksum
(0 36000 29145 = coding)

- formula for check digit:

x = (sum of odd position digits) × 3 + (sum of even position digits)

x % 10 = y (10 − y (if y ≠ 0)) = check digit

- example: 0 36000 29145 2

(0 + 6 + 0 + 2 + 1 + 5) × 3 + (3 + 0 + 0 + 9 + 4)

= 14 × 3 + 16 = 42 + 16 = 58

58 % 10 = 8

10 − 8 = 2   2 is the check digit   confirmed ✓

Exam Style Questions 1, 3, 5

1. a) i accuracy ✓

ii Validation is concerned with the correctness of the form (and type −1) of the data, e.g. formatting. It checks if the data is feasable. ✓ Verification ensures that the data doesn't corrupt or unintentionally change along the way. ✓

iv. As data validation is only concerned about the right (type and) format and data verification with the data being transmitted correctly, the data entered doesn't have to be correct. E.g. when asked to enter your name, you can just enter the name of your friend instead. It passes both validation and verification. ✓

iii Entry validation: Prompt the user to enter a value, e.g. his password two times to increase the chances of data integrety, i.e. the password being what the user intended it to be. ✓
more detail: strong password, biometric tests unauthorized acc.

b) The data is stored on a system that is password protected, and the hard-drive is encrypted ✗ access rights

3. a) i First the system has to decide it it chooses even or odd parity. ✓ This has to be equal for both the sending and receiving end. ✓ We assume even parity.
On the sending side:
- for each 7 bits the number of ones is counted.
- If the number is odd, a one is addet at the parity bit to make the number even ✓
- The data is sent

On the receiving end:
- The data is received and the ones are counted.
- If a byte has an odd number of ones there was a mistake ✓ and the byte is requested again ✓

ii Checksum works as follows: Data is split into blocks of data. To each block a few bytes are appended that will later contain the checksum. ✓ All bytes in the block are treated as if they were binary numbers ✓ and added together. ✓ Usually they undergo some complex calculations and are then stored as checksum. Often the last step is to take the mode of the value and store that as checksum ✓
– The value is recalculated on the receiving end and compared

b) It cannot be determined which exact bit is wrong. → both
Only one mistake per byte can be recognized → parity bit
two might cancel

c) ~~0✓ 0✓ 0✗ 1✗ 0✓ 1✗ 0✓ 0✓~~
~~bytes 3, 4, and 6 would be resend~~
bad question, actually 2d parity check

5. Verification is needed to ensure that the data received is the same data that was sent.
Validation is needed to ensure the data is feasable, which increases the likelyhood of it being correct.

Both are needed to improve, not ensure data integrity.

5. Verification: • Ensures that data entered is what was intended
- Can include comparison with the source data if there exists such source data
  ↳ e.g. checksum
- Options for manual data entry verification:
  - double entry check
  - visual check → show what was entered
- For transmission verification:
  - checksum & check digit
  - parity bits

Validation: Used to check if data is in the correct form and is the correct type
Checks might include:
- type check
- presence check
- format check
- range check
- limit check
- length check

Both are needed to improve the chances for data integrity, ie. the data to be accurate. Validation and verification don't garantee data integrity though.

7.1 Ethics

=) Collectively agreed upon moral principles that decide between right or wrong

=) In comsci: Ethics = Rules of Conduct

- Professionals should seek ethical guidance by organizations which provide a code of conduct:
  ↳ e.g. • (BCS) British computer Society
  • (ACM) Association for Computing Machinery
  • (IEEE) Institute of Electrical and Electronics engineers

- 8 principles by IEEE and ACM:
  1. Software engineers should always act within public interest
  2. Software engineers should act in the best interest of employer/client
  3. Products and modifications meet highest professional standards
  4. Maintain integrity and independence in their judgement
  5. Promote ethical approach as a manager
  6. Advance the profession of software engineering in the public interest
  7. Be fair and supportive to colleagues
  8. Practice lifelong learning and promote an ethical approach to the profession

What most codes have in common:
- Public interest is key (public good / public safety & welfare)
- The code presents fundamental principles
- Own judgement is expected
- Advice should be requested when unsure

- It is usually impossible/harmful for individuals to speak out against powerful cooperations about not being ethical

BCS Code of Conduct

Public Interest:
- public health, security, privacy, wellbeing
- recognize intellectual property rights
- don't discriminate
- promote equal access to software

Professional Competence and Integrity:
- only undertake work that is within your abilities
- always develop your professional skills / knowledge
- seek criticism
- don't bribe, be bribed

Duty to the Relevant Authority:
- Meet the authorities requirements while exercising personal judgement
- avoid conflict of interest between you and authorities
- accept professional responsibility
- don't misinterpret instructions or share confidential information

Duty to the Profession:
- uphold the reputation of the profession and BCS
- improve professional standards → development & enforcement
- treat colleagues with respect and offer support

Exam Style 1, 4, 5

1. a) i Public interest ✓ / to find an ethically 'right' decision
   ii professional ✓ judgement (✓)
   iii (the public, employer), colleagues (clients, self)
   iv (colleagues), employer, clients
   v. Jobs / tasks (duties) ✓ the required skills and (expertise) experience ✓
   b) It (promotes transparency and allows for better access to the code by less-skilled developers.)

4. answered in book
   - superior to verbal instructions
   - ...which might be forgotten
   - can be made universally available, can be updated

5. The software engineers should...
   1. work in the interest of the client.
      ↳ Instead of slacking off and delivering way past the deadline, the programmers strive to provide the client with a good product.
   2. work in the interest of the public.
      ↳ The programmers will refuse to implement the shady tracking service requested by their client.
   3. Only perform tasks that they have the skills and experience for
      ↳ The programmers divide up the tasks so that everybody works on what they specified in (e.g. frontend or backend)

7.2 Copyright & Licensing

Copyright

=> formally acknowledged ownership

When copyright can be claimed:
- a person creates an original piece of software -> he claims copyright
- He actually works for a company -> organization claims copyright!

When copyright Doesn't Apply:
- Ideas
- Already published work owned by someone else

Types of media that Copyright can be claimed for:
- a literary work (e.g. book)
- musical composition
- film
- music recording
- radio / TV broadcast
- work of art
- computer program

Why Copyright exists:
- enable people to claim the profit of their hard work
- prevent others from profiting without credit to the original creator

Widely Accepted Copyright Laws
- registration must include date of creation
- a period for which the copyright applies
- what happens when the owner dies
- How to indicate copyright, e.g. ©
- Owner can set rules for use: e.g. ethics
- When a work is bought, copies can be created for personal use

Commercial Software

=> users buy an 'end-user license'
↳ permission to use the software, not own it

Variations of Software Licenses
1. a fee is payed for each copy
2. company buys a 'site licence'
   -> allows for n copies to run simultaneously
3. Discounts may apply e.g. for educational purposes

Licenses Free of Charge

- Shareware
  - Software made available on a trial (limited time)
  - Might include the full software or a limited version of it
  - Examples:
    - Free trial
    - Testing / Beta versions
- Freeware
  - Might be a full, limited, or old version
  - Has no trial and the end-user license is free

=> Any license will define limitations on how the software can be used

=> Rarely will the sourcecode be provided for any commercially licensed software

When commercial Software is Used:
- software is available and user friendly
- is created to work with already installed software
- continuous support and maintainance provided
- Shareware might allow for suggestions for improvement (bad reason)
- Freeware is sufficient

Open Source (Open Licensing)

- Software for which the source code is available
- Usually users are allowed to:
  - modify
  - distribute ... the software, though within the defined bounds
  - use by the license
- Open source promotes
  - collaborative development
  - the sustainability of a software -> people will update it

Free Software (not free of charge)

=> open source software sub-category which uses 'copyleft'
- copyleft: any modified source code must be made available under the same conditions of usage
- free software has following requirements:
  - a program's inner workings must be accessible
  - one can modify the code
  - One can run the code for anything
  - one can redistribute the code, modified or not

Organizations

Open Source Initiative:
- Philosophy: Destribute software free of charge and promote collaboration in development

Free Software Foundation:
- Philosophy: Users should be free to use software in any way they wish (Not necessarily free of charge)
- usually includes a small fee for distribution cost
- All published software is 'free software' -> copyleft applies

Why to use Open Source software
- it's free / cheap
- software can be customized to fit one's needs
- fosters collaboration with community

Exam-Style 2

2.a) i Copyright defines the terms for destributing, modifying and using a piece of work?
ii Date of Creation, Conditions on creator's death
iii Copies can be created for personal use
iv ! A fee of 1$ has to be payed to the original creator for every application sold that includes this product
2. The software may only be used for military development.

b) i The license can either be applied to any direct copies or with copyleft. Copyleft means that instead of being able to re-use the software under one's own conditions, the modified software has to be submitted with the same conditions as the original work This might be highly problematic for a company which actually wants to develop a software to capitalize on.

ii Freeware is commercial software that is given out to the user with no time constraint while trial ware is given out on a trial - for a limited amount of time.

7.4 Functions & Procedures

=> Functions and Procedures are <u>Subroutines</u>

=> Both may take an input

Difference: A function returns some value(s) while a procedure never returns anything

<u>function ex.</u>:

```
def add(val1, val2):
    return val1 + val2
```

returns value to code

<u>procedure ex.</u>

```
def print_name(name):
    print(f'your name is {name}.')
```

doesn't return value (back to code at least)

- Subroutines have to be defined before the main program
- Subroutines can be called (e.g. in the main program) by referencing them
- <u>Return value</u>: The value returned by a function
- Procedures are often referrenced as 'void function'
  - ↳ void means 'nothing'

<u>Passing Parameters to Subroutines</u>

- <u>Arguments</u>: The actual values passed onto the subroutine
- <u>Parameters</u>: The placeholders for arguments within a subroutine
- <u>Subroutine interface</u>: Process of matching arguments to the corresponding subroutines
- <u>Function / Procedure header</u>: Parameters are defined in the function (...) header

example:

```
def say_hello(name1, name2):        } function header
    sentence = "Hello " + name1 + " and " + name2 + "!"
    return sentence                 <- return value

say_hello(Philipp, Toni)
```

(parameters: name1, name2; arguments: Philipp, Toni)

- Variables can be passed either...
  - <u>by value</u>: a copy of the value is passed to the subroutine
    - ↳ changes to the copy don't affect the original
  - by reference: the address of the value is passed on
    - ↳ any changes to the value are changes to the original

Coding

```
    A
   A A
  A A A
 A A A A
AAA A A A
```

1	1
2	3
3	5
4	7

8.1 Relational Databases

Problems with File-Based Systems

- all databases, file-based or not, are vulnerable to wrong data entry
  ↳ fix: validation
- File-based systems using only a single file don't allow for permission handling

File-based systems using multiple files:

- are vulnerable to data redundancy
  ↳ means same data stored more than once
- data dependency becomes an issue:
  - changing data in one file doesn't update the same data in other tables → integrity issue
  - pre-defined structure issues: when inserting an extra column, a program that targeted e.g. column 4 might have to be rewritten
  - databases tend to be more rigid, making it harder to create new programmes that use a database that was optimized for other programs
    ↳ I imagine this being an issue with any database...

Relational Databases

Structure:

- data is stored in relations which are special types of tables
  ↳ it is given a name and a list of attributes on creation
  ↳ Users (userID, balance, recent...) ← created like this
- Attributes are the column headers of a table, e.g. 'User_ID'
- There is no order for columns or rows
- A row in a relation is called a tuple
  ↳ sometimes reffered to as 'record' and attribute values as 'fields'
- A tuple is reffered to as 'one instance' of a relation
- Relational databases use 'atomic' values: an attribute value holds either one value or no value

Keys

- Primary key: each relation has one attribute (or a combination of attributes) that are used as primary key
  - each tuple must hold a unique value for that key
  - ensures integrity → no duplicates → used as a selector
- Candidate keys: some relations have multiple attributes for which all values are unique
  ↳ those attributes are considered candidate keys
- Secondary keys: candidate keys that weren't chosen to be a primary key

- often there is no candidate key, so a primary key has to be created
- the DBMS (database management system) rejects any new tuples with already existing primary key values

- Foreign keys: an attribute (a column) that refers to a column in another table (relation)
  - prevents data redundancy and protects integrity as data gets changed at all location
  - provides referential integrity: foreign keys can be used to link attributes to prevent entry of non-existing values
    ↳ the foreign key lists all acceptable values
    ↳ the DBMS checks for referential integrity

Exam-Style Questions 1, 5a, 6a

1. a) i Relation → the entire table
   Attribute → a column, eg. 'subject studied'

   Tuple → a row, e.g. Xiangfei, 3, MUBS, Computing, A, DER

   ii There are no (attributes) column headers, so it's not clear which column matches to which attribute. This could be fixed by addid column headers.
   Also there's no truely unique attribute, so there are no candidate keys that could be chosen as a primary key. This could be solved by creating a private key, eg. 'user_id' and generating the primary key attribute for each new user. This could be an index number.

   iii) haven't done — Student Name has Primary key → Student Name

   b) i) 1. Prevention of data abundancy through foreign keys
   2. The splitting of one table into ~~multiple for permission handling~~ or better accessability...?
   3. Tutorial has a compound primary key

   ii) It would be a problem if no foreign key was used for Student Name as it might cause integrity issues and would result in data abundancy
   ↳ Student names might not be unique but it's a primary key (haven't seen where)

   iii haven't done

5 a) i Database Management System ✓

   ii He could introduce different levels of authorization that allow access to different attributes ✓

   He could do regular updates? ✓

   iii The school secritary could use the DBMS to search for all students that attend a specific course. ✓
   This could e.g. be done over the ~~terminal or by dedicated software~~

   iv. Data integrity and abundancy are no more of a problem due to the use of foreign keys. ✓
   It's easy to augment the database and insert new attributes, as there is no pre-defined order. (✓)
   no dependency between DB and program

6. a) 1. Data security: relational databases allow for easy permission handling ✓ (no different authoriz. for one file)
   2. Data integrity: foreign keys change data globally and prevent data abundance redundancy (✓)
   3. Augmentability: due to the modular structure it's easy to create new relations or insert new attributes. There is no predefined order for attributes which allows for insertion of new attributes ✓

   The exact data structure don't have to be known by the program ✓ format and better

8.2 Entity-Relationship Modelling

=> Entity-relationship diagrams (E-R diagrams) are used to visualize which data is linked between different tables

Entity

- some concept that will become a table in a database
  - e.g. a 'member', 'class', or a 'booking'
- entities might share the same attributes through foreign keys
  - e.g. BookInd: band name and BandInfo: band name

Stepwise Refinement

=> E-R diagrams use a top down approach:

1. broad relations are made between rather unspecific concepts (entities)
2. Entities and connections are repeatedly subdivided until they are specific enough to work efficiently in code

ER-Diagram Syntax

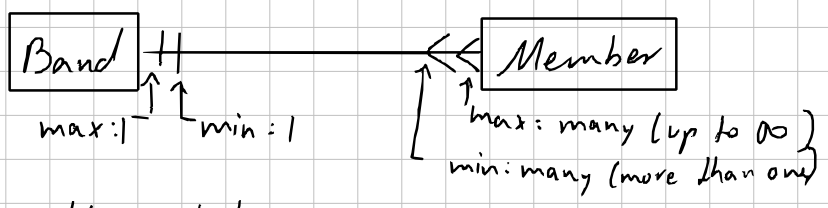
=> relations between entities are called <u>cardinalities</u> which have different symbols

Preliminary (undefined)

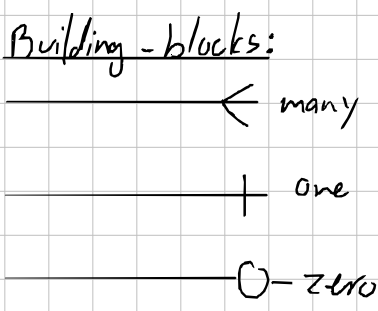
Cardinality Notation

* the relations are between single entities of each table (relation)
  - e.g. 1 band entity has many member entities

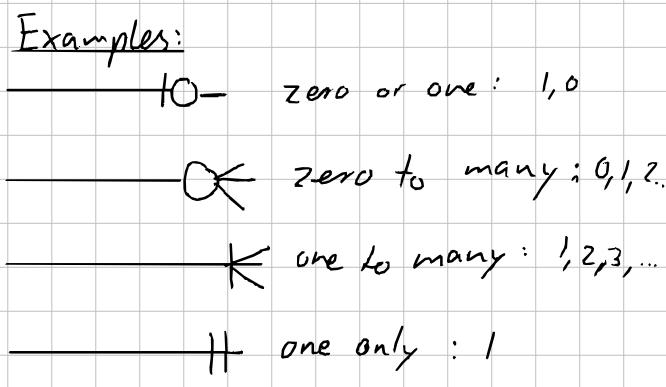
=> At each end of the relation there is a maximum and a minimum defined:



Building-blocks:

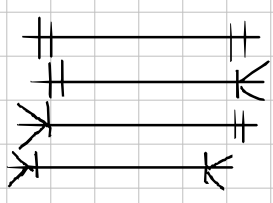


Examples:



Types of Relationships

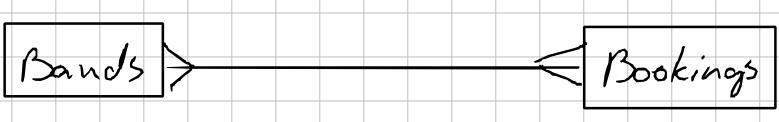
- one-to-one (1:1):
- one-to-many (1:M):
- many-to-one (M:1):
- many-to-many (M:M):



Problem with 'Many-to-Many' Relationships

=> M:M relationships can't be implemented in databases, <u>link entities</u> are required

- a foreign key can link a single attribute in one table with an attribute in one or many other tables (entities)
- a foreign key cannot link multiple attributes in one table with multiple attributes in an other table (entity)
  - the foreign key would lack an origin - a place where it is created first



Solution: Link Entities

=> The linked entity can store the two foreign keys of the two entities. They both refer to the primary keys of the other entities:

Bands
- Yoasobi, members
- Daft Punk, ...
- Stoners, ...

Band-Booking
- 1, Yoasobi, June 1st
- 2, Daft Punk, March 2nd
- 3, Yoasobi, April 5th

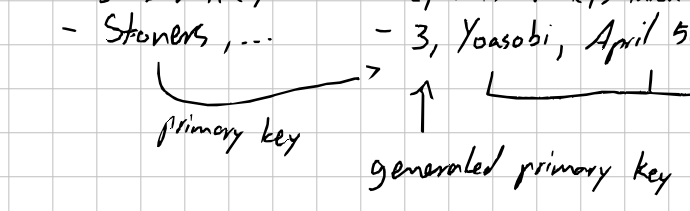
Bookings
- June 1st, 4k $
- March 2nd, 10k $
- April 5th, 6€ $

primary key (Bands) → generated primary key (Band-Booking) ← foreign keys ← primary key (Bookings)

- the linked entity keeps track of all combinations

Solution #2: Subdivide

=> Split many-to-many relations until they become 1:1 or 1:M



e.g. 30. June, 6th July, 9th July

=> This might not always be possible, so remember link entities

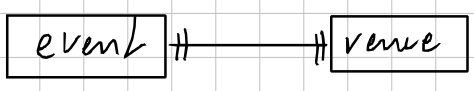
---

Exam Style 2. 6b

2. a) event, dishes, venue, staff

b) venue-event, dishes-event, staff-venue

c)



?

one event has one venue

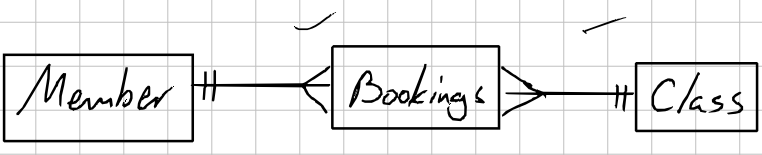
?

one event has multiple dishes

?

multiple staff members will be at a single venue

6. b)



revisit

8.3 Normalization (Database-related)

=> Restructure databases to reduce data redundancy

Normal Forms

=> Describe the grade of optimization regarding data redundancy

- 1NF has lots of abundancy, 3NF is aimed for

1NF:

- All values (fields) are atomic, meaning that they hold only a single value, not a list.
- Each row has a primary key

2NF:

- database fulfills 1NF
- no partial dependencies of non-key attributes on composite primary keys
  - ↳ ask yourself: could I identify this (non-key) attribute with only part of the composite primary key? -> move it to it's own new table
  - ↳ single primary key relations are automatically in 2NF

StudentClassInfo (<u>StudentID, ClassID</u>, Grade, Teacher)
primary key (composite)

↳ grade is dependent on student id and class id -> it's good

↳ teacher is only dependent on Class id -> it ain't good

StudentClassInfo (<u>StudentID, ClassID</u>, Grade)

Class (<u>ClassID</u>, Teacher)
foreign key

3NF:

- database fulfills 2NF
- non-key attributes don't contextually depend on any other non-key attributes

CharacterDescription (<u>DescriptionID</u>, Described character, book title) X
↳ depends on

CharacterDescription (<u>DescriptionID</u>, characterID)
Characters (<u>characterID</u>, character name, book title)

Summarized:
3NF means each non-key attribute is dependent on the primary key, the whole primary key, and nothing but the primary key.

Un-Normalized Data

=> Data is said to be 'un-normalized', if it contains repeating groups or straight up lists

e.g. (CustomerID, Name, Phone number #1, Phone number #2)
repeating group

- repeating groups are sets of attributes which represent similar data

↳ leads to data redundancy because not all customers have two phone numbers

---

Exam Style 3.

a) Booking Number (non-repeating), hotel (non-repeating), date (non-rep.), Room type (repeating), number of rooms (non-rep.) room rate (non-rep), address (rep), rating (non-rep.)

b) Booking (<u>Booking Number</u>, date, hotel (fk), room type, number of rooms, room rate)

Hotel (<u>hotel</u>, address, rating)

c) The requirements for 2NF are that all values are atomic, there are no repeating groups, and no partial dependencies of non-key attributes on composite keys. As 1NF is already satisfied (at least when fixing the repeating atomic (formatting) issue) and because there are no composite primary keys that could have any partial dependencies, the database is automatically 3NF.
satisfies

8.4 Database Management System

=> DBMS provides tools to create, retrieve, update, and manage data inside databases

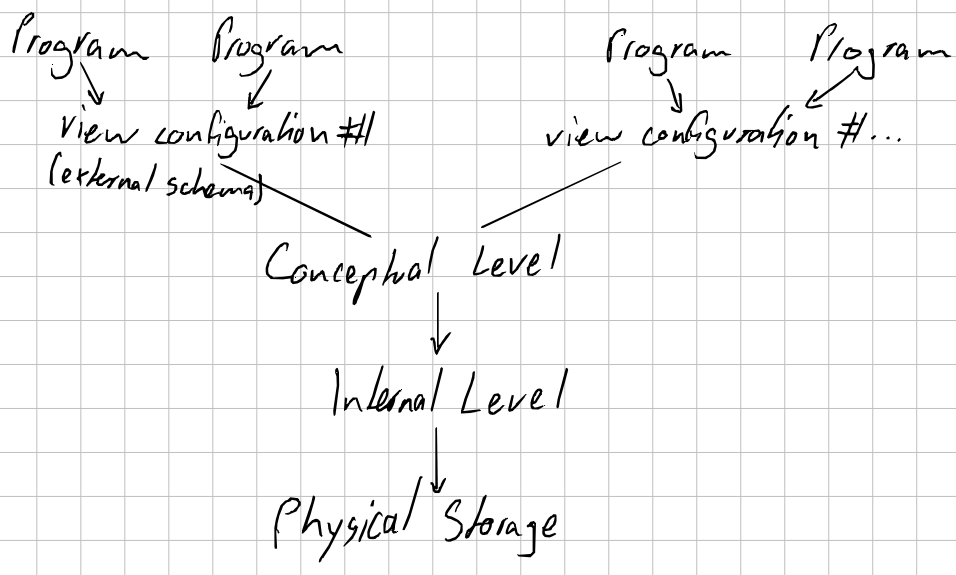
Database Model as of ANSI

⇒ ANSI: American National Standards Institute

- Devision into three levels:
  - external level
  - conceptual level
  - internal level

  ] all managed by DBMS
- all is on top of the physical storage

External Level



Database Seperated From Storage - Internal Level

- only the internal level is concerned with the actual memory addresses and provides a layer of abstraction
  ↳ it performs all physical changes on the storage medium

Universal View of the Database - Conceptual Level

- The conceptual level provides access to all data in the database
- It provides an all including universal view
  ↳ only the Database Administrator (DBA) has access
  ↳ a view is refered to as a 'logical schema'

External Schemata and Access Management - External Level

- The external level defines different logical schemata that have different permissions
  ↳ show only x to y
  ↳ only read, don't edit, etc.
- external schemata provide customized views that improve efficiency

Developer Interface

- The DBMS provides a developer interface
  ↳ allows developers to communicate instructions with SQL
- A query processor takes queries (SQL instructions) and processes them
- Queries can manipulate and retreive data

Reports

⇒ The DBMS can create reports which are predefined views of partially processed data

Tools for the Database Administrator (DBA)

- tools for defining access rights

Indexes

- the DBA controlls and creates indexes.
- Indexes are secondary tables that hold pointers to where data is stored
  ↳ may faster to search through them through actual data
  ↳ pointers point to entities (touples) in the original table
- Indexes can be on a primary or secondary key

(secondary key = an attribute where all values are unique, but it's not the primary key)

Data Dictionaries

- only the database administrator has access to the database's data dictionary
- It contains metadata about the data:
  - e.g. attribute names, table names
  - relations between tables
- information on how the physical storage is structured
  ↳ only high level, e.g. storage space & performance
  ↳ DBA accesses conceptual level, not internal level

Data Security

- a half completed instruction is very dangerous to data integrity
  ↳ leaves the whole database corrupt / in an undefined state

  undefined state: data is only partially updated and 'in limbo' until query either fails or completes
- The DBMS has to prevent half-executed queries
- DBMS also is responsable for regular backups

Exam Style Question 5b i, ii

i. Relationships between relations (tables) are created by the database administrator using queries. Using the DBMS developer interface, he can query the creation of both linked relation by sending sql commands to the query processor.

ii many-to-one (many Class-Group, one Student)

i In the Class-Group Relation, the Class ID is a foreign key which links to a primary key in Class. This is most likely going to be class ID ✓

8.5 Structured Querry Language (SQL)

=> language used to interact wit databases

- Two categories of SQL: DDL & DML...

Data Definition Language (DDL)

=> SQL used to manipulate the structure of a database

- Comman DDL commands are:
  - CREATE DATABASE <name>;
  - CREATE TABLE <name> (attribute#1 datatype, attribute#2 datatype...);
  - ALTER TABLE <name> ADD PRIMARY KEY (<name>);
    - sets existing attribute as primary key
    - ↖ or secondary key
- keys can be created on creation of the table or added later

Some SQL Rules

- consists of sequence of commands
- commands end with ';' and can span lines
- no case sensitivity (T = t)
- lists of items are seperated with a comma
- some commands require their variables to be enclosed by '( )'
  - ↳ e.g. create table

Datatypes in SQL

- datatypes have to be defined on creation of an attribute
- datatypes and datatype names might varry with different DBMS
- Common Datatypes:
  - character(<value>) -> uses predefined amount of storage (text)
  - varchar(<value>) -> only uses as much storage as text needs
  - boolean
  - integer
  - real -> basically a float
  - date
  - time

  ↖ requires maximum characters to be defined

Data Manipulation Language (DML)

=> SQL used to modify data inside a database

↳ This includes insertion of data, modification, deletion, and reading

- Common Commands:
  - SELECT <name> FROM <name>;
  - INSERT INTO <name> (attribute#1, attribute#2) VALUES (<value>,<value>);
    - can be used alone if attribute order of table is known
    - is used when order is unknown: attribute #1, #2 will be attribute names and order in values matches order of insert into

- commands that can be added on top:
  - ORDER BY <attribute>* orders alphabetically by selected column
  - GROUP BY <attribute> eliminates duplicate or entities with identical value for <attribute>
  - WHERE <attribute> = <value> ← filters entities. '=' can be e.g. '>'

* ASC and DESC can be specified to change the order

Aggregate Commands

=> SQL Commands that perform some operation with data from multiple tuples

- Example Commands:
  - SELECT Count(<attribute>) counts rows that have a value for <attribute>
  - - AVG(<attribute>) attribute average
  - - SUM(<attribute>) sum of attributes

Join Conditions

=> two tables are joined based on a common attribute, data can be fetched from that temporary combined table.

example:

```
SELECT VenueName, Date        <- values to fetch
FROM Booking
WHERE Band-Booking.BookingID = Booking.BookingID   <- combine both tables with foreign key 'BookingID'
AND Band-Booking.BandName = 'Daft Punk'            <- filter combined table for specific entity
```

Modify Data

- UPDATE <table> SET <attribute> = <value>
  WHERE <attribute2> = <value>;
  - sets the data for all values of <attribute>
  - filters for specific entity to modify
- DELETE FROM <table>
  WHERE <attribute> = <value>; ← delete all entities in <table> where the attribute has a certain value

- Deletion Error with Foreign Keys:
  - if an attribute acts as a foreign key in an other table, the DBMS won't allow the deletion of the original attribute
    - ↳ foreign keys have to be deleted before the original attribute

---

Exam Style 4, 5b iii, iv, 6c

4. a) create table subject (

subjectname varchar primary key,
subjectteacher varchar);

create table student (studentid integer primary key,
studentname varchar,
studentothername varchar,
date of birth date);

b) alter table subject set primary key (subjectname);

alter table tutorial set studentid forei

9.1 Computational Thinking

=> Step-based problem-solving through ordered logical reasoning

Five Components:

Abstraction

=> Summarize the key information which is important to the goal

↳ filter unnecessary complexity and unrequired information

- e.g. abstract datatypes remove a layer of complexity by grouping similar data together and providing tools that work with this group

Decomposition

=> Breaking problems into smaller more managable problems

- e.g. functions

Data Modelling

=> Identifying what information is needed in which way
↳ this ties into creating custom abstract datatypes

- e.g. The program requires a queue-like datatype, but no such datatype is provided → an abstract datatype is coded

Pattern Recognition

=> Finding patterns in problems to find common solutions improves efficiency in work and code

- e.g. There are many standard algorithms that cover most logical problems

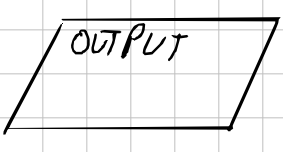
Algorithm Design

=> Define step-by-step instructions to solve a problem

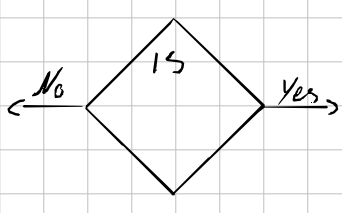
9.2 Introduction to Algorithms

Flowchart Rules for Algorythms

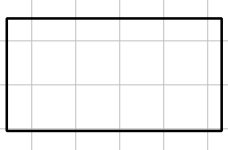
Input / Output



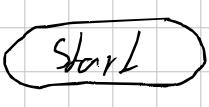
Decision



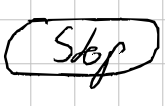
Process

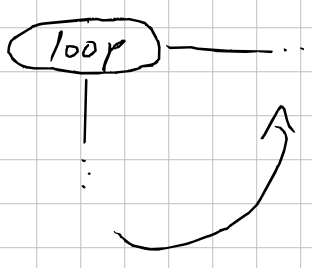


Start



Stop



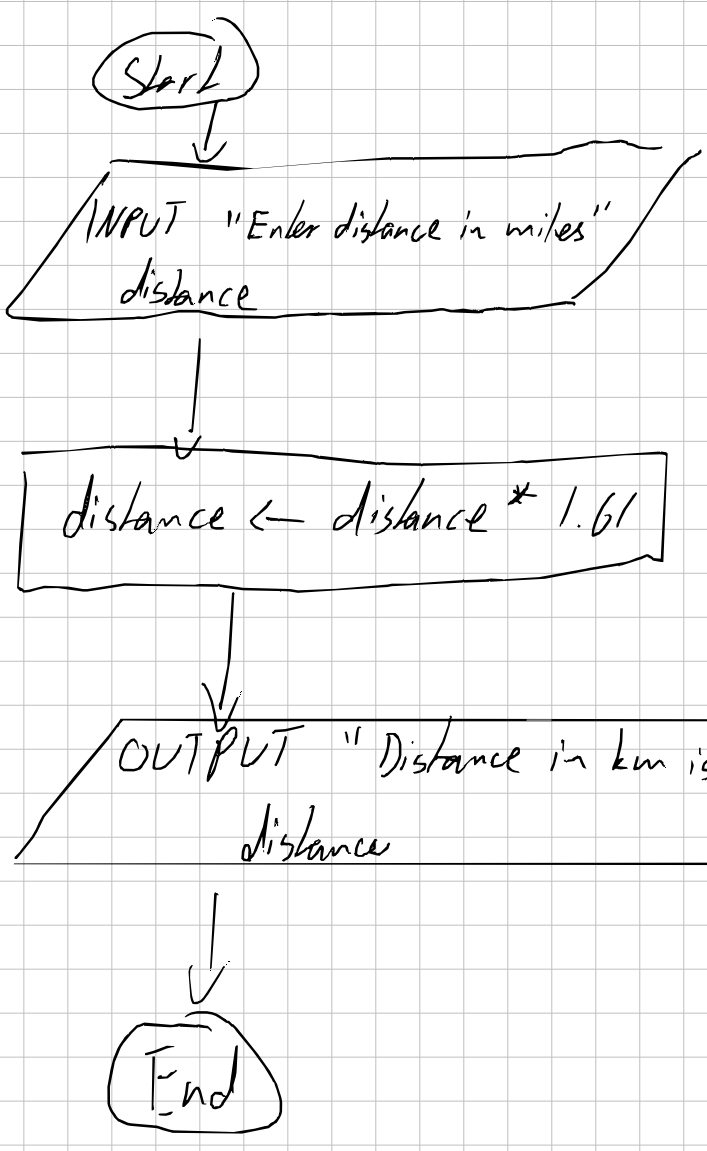


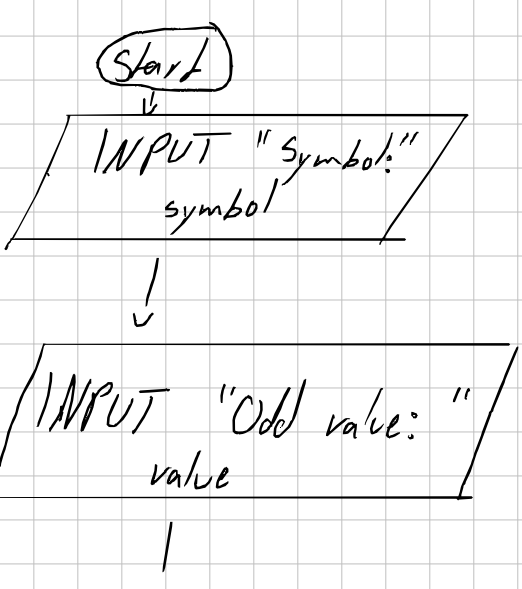
Algorithm Definition:

A sequence of steps to solve a problem in a finite amount of time.

17.01

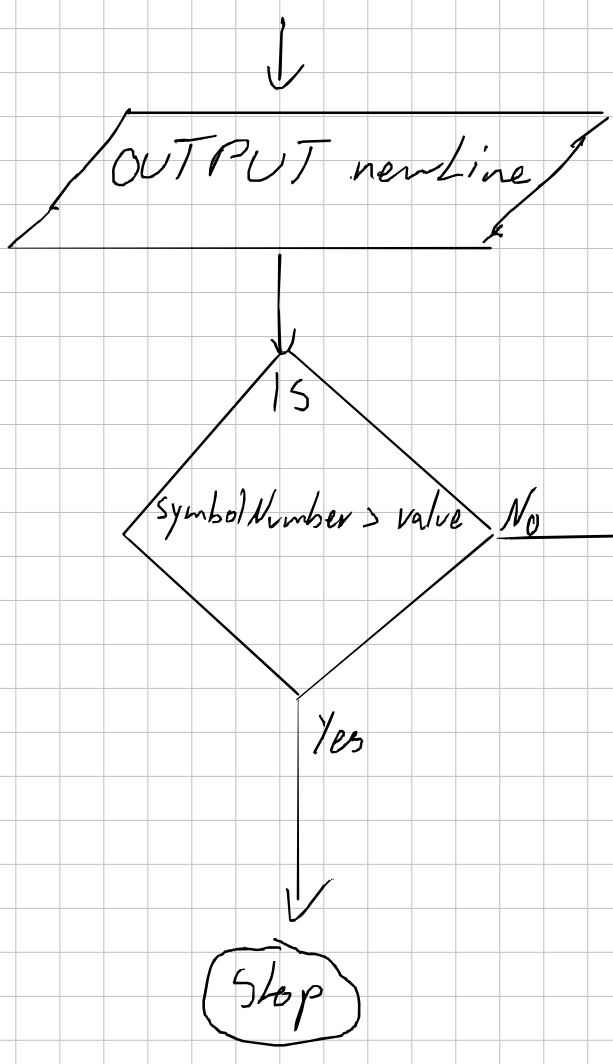
```
DECLARE distance : Integer
INPUT "Enter a distance in miles: " distance
distance ← distance * 1.61
OUTPUT "Distance in km is: " distance
```

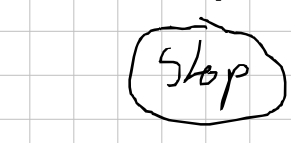




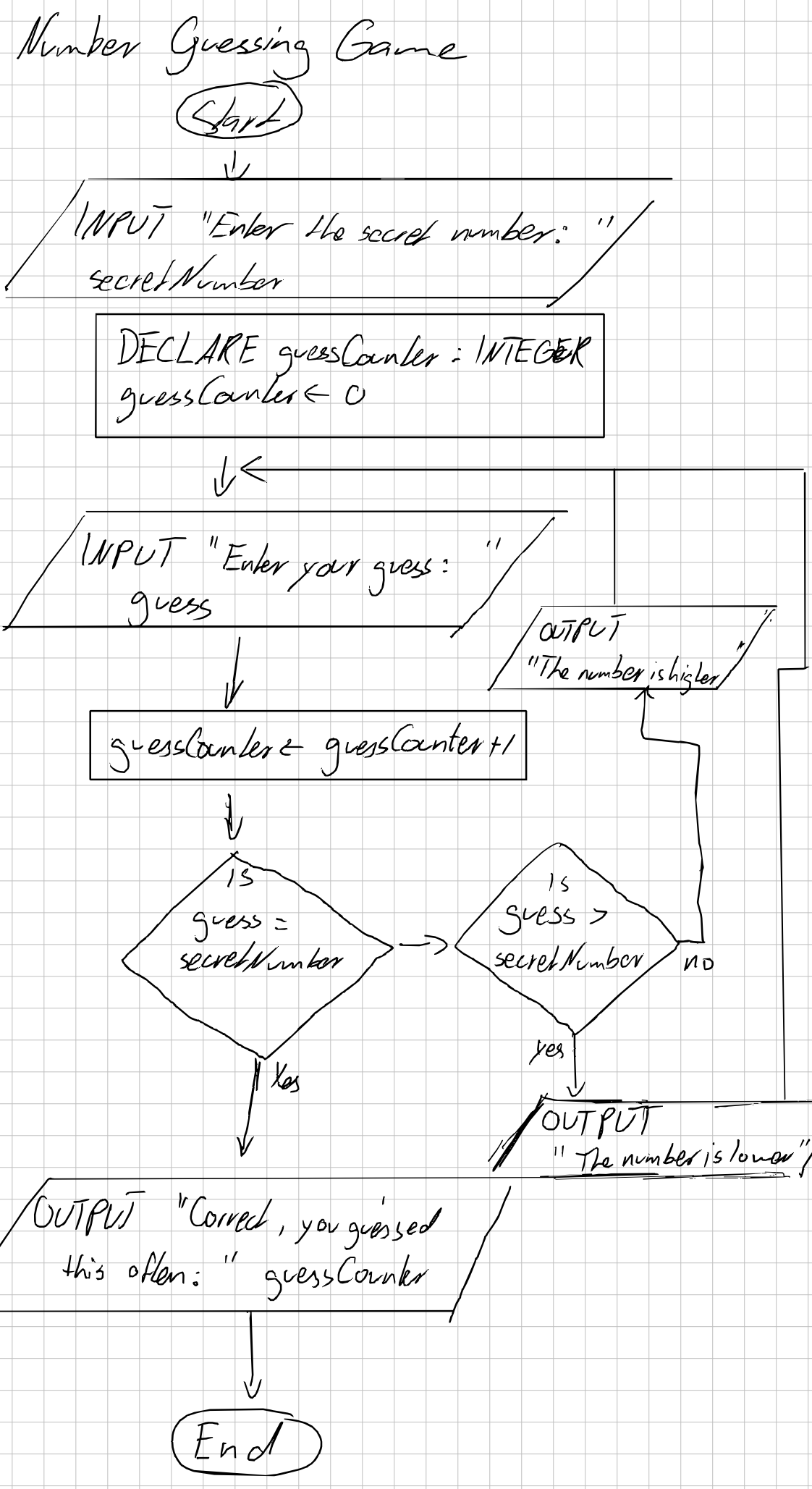
1

```
DECLARE symbolNumber: INTEGER
symbolNumber ← 1
DECLARE newLine : STRING
DECLARE increment : INTEGER
increment ← 0
newLine ← ((symbolNumber - 1) / 2 - increment) * " "
newLine ← newLine + symbol * symbolNumber
symbolNumber ← symbolNumber + 2
increment ← increment + 1
```





Number Guessing Game



Exam-Style Questions

5	9	1	6	...
10	9	6	7	...

$(5 \times 10 + 5 \cdot 9 + 1 \cdot 8 + 6 \cdot 7 \cdots) \text{ MOD } 11 = y$

$11 - y =$ check digit

if $11 - y = 10$ then checkdigit = 'X'

10.1 Data Types & Records

Primitive Data Types

=> Basic pre-defined datatypes provided by programming languages

- also known as 'atomic' data types

Common Primitive Data Types:

Integer	42
Real / Float	3.141
Char	'H'
Boolean	True
(String	"Hello") debatable

Structured Data Types

=> Data types consisting of primitive data types

Date	06/08/2007
Sring	"I love cake"

↳ string consists of many characters, date of integers

↳ (There is some debate around strings being structured or primitive data types, e.g. imagine an empty array)

Record Type

=> A data structure that holds a collection of logically connected data of various data types

- e.g. the data type 'person' might consist of birthday, height, weight, hair_color
- in Python the 'class' method is used

Pseudocode:

```
TYPE BookType
  DECLARE Title, ISBN : STRING
  DECLARE Year_of_publication : INTEGER
  DECLARE Price: REAL
ENDTYPE

DECLARE new_book : BookType

new_book.Title ← "Computer science
new_book.ISBN ← "978..."
new_book.year_of_publication ← 2015
new_book.Price ← 44.95

OUTPUT new_book.Title
```

10.2 Arrays & Algorithms

=> An array is an ordered list of variables that all share the same datatype, stored under a single identifier

List* vs Array

- a list can store a mix of datatypes, an array can't
- Arrays store data more efficiently
- Arrays aren't native to python, lists are

*note, Lists sometimes refer to 1D arrays but can also mean mixed-data-type-lists

Array Technical Terms

data type for all elements is required

DECLARE new_array [0 : 10] OF INTEGER

- new_array: array identifier
- [0 : 10]: lower bound : upper bound
- the bound is inclusive, meaning [0:10] creates 11 values

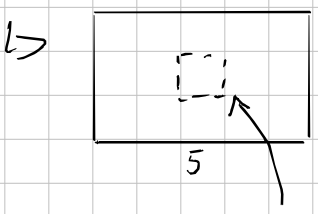
new_array [2] ← 42

- [2]: array index

2D Arrays

> DECLARE new_array [1 : 3, 1 : 5] OF BOOLEAN

- [1 : 3]: rows
- [1 : 5]: columns



as we defined start to be at 1 [1 : 3 ...]

> new_array [2, 3] ← TRUE

Algorithms

Linear Search

=> Compare each element of an array with a value until the matching value is found

[1, 5, 7, 9, 0, 42]

=0? no =0? no =0? =0? no =0 yes! found at index = ...

Bubble Sort

=> Sort numerical values in an array by comparing two numbers next to each other and switch them if the left is smaller. After each pass the array is reduced as the last place is certainly correct

1, 5, 3, 2

ok

[1, 5], 3, 2

1, [5, 3], 2 → 1, 3, 5, 2

1, 3, [5, 2] → 1, 3, 2, (5) we know for sure the five is largest

[1, 3], 2, 5 five is not iterated over anymore

⋮ until a perfect run is completed (all values are in correct places)

10.3 Text Files

=> Text files play the significant role of storing data past the point where the program stops

All syntax on how to read, edit, and append is on Notion.

End-Of-File / End-Of-Line Marker

example.txt:

```
Hello, this is the first line EOL-Marker
This is another line EOL-Marker
This will be the last line EOL-Marker
EOF-Marker
```

```
OPENFILE "test.txt" FOR READ
WHILE NOT EOF("test.txt") DO
...
```

↑ the EOF() function checks if the cursor inside test.txt has already reached the EOF marker

↳ It's better to use pre-condition loops instead of post-condition loops as the file might be empty and the EOF marker is reached immediately

---

13.08

1.
```
DECLARE board :ARRAY[0:2, 0:2] OF CHAR
: game occurs, all fields are filled
OPENFILE "lastGame.txt" FOR WRITE
DECLARE newLine : STRING
FOR row ← 0 TO 2
    newLine ← ""
    FOR COLUMN ← 0 TO 2
        newLine ← newLine & board[row, column]
    WRITEFILE "lastGame.txt" newLine
CLOSEFILE "lastGame.txt"
```

2.
```
OPENFILE "lastGame.txt" FOR READ
DECLARE board :ARRAY[0:2, 0:2] OF CHAR
DECLARE newLine : STRING
FOR row ← 0 TO 2
    READFILE "lastGame.txt" newLine
    FOR Column ← 0 TO 2
        board[row, column] ← MID(newLine, column+1, 1)
CLOSEFILE "lastGame.txt"
```

10.4 Abstract Data Types (ADTs)

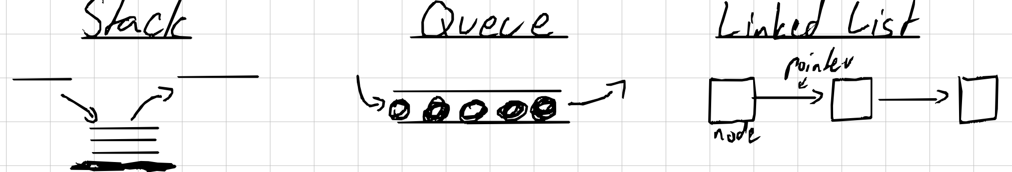
=> Describe a data structure and a collection of operations that can be used on data with the datatype

- operators often include:
  - create instance
  - find element
  - insert new element
  - delete element

  and sometimes: - show all data

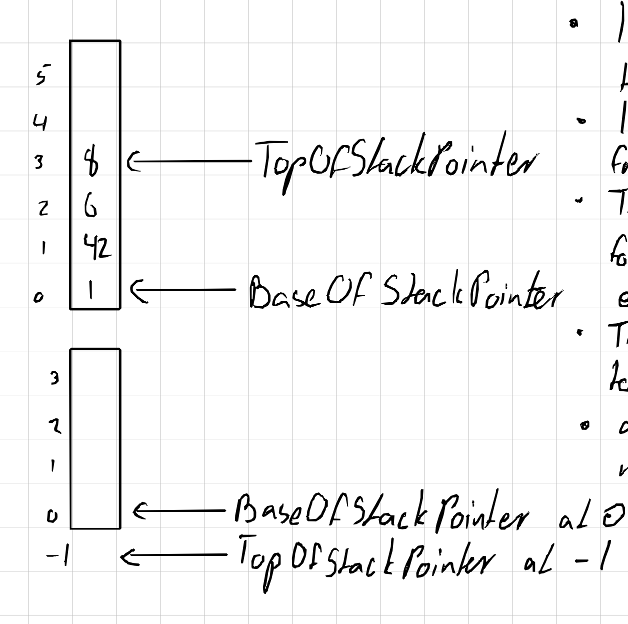
Common Abstract Data Types

Stack	Queue	Linked List
Last-in-first-out (LIFO)*	First-in-first-out (FIFO)*	A list where one item points to the next
e.g. stack of CDs	e.g. toilet queue	e.g. browser link history



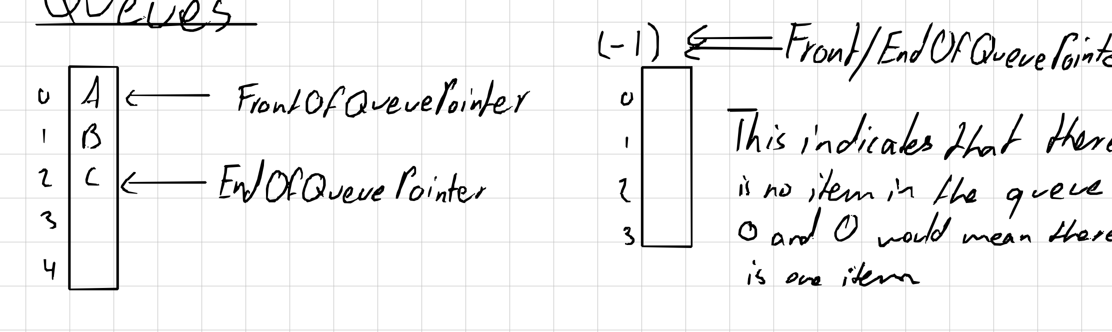
*LIFO = The last (most recent) item in is the first one out
*FIFO = The first item in is the first item out

Stacks



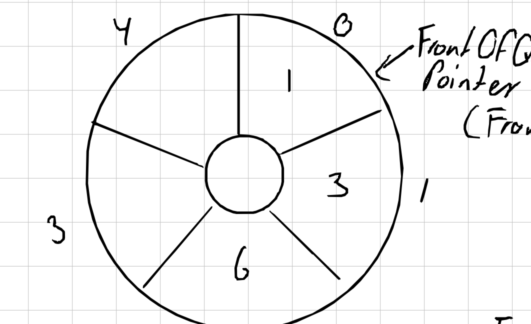
- Items can be pushed (added) to the stack
- Items can be popped (removed) from the stack
- The TopOfStackPointer always follows the position of the newest element
- The BaseOfStackPointer always points to the first slot
- as the BaseOfStackPointer always remains at zero, the top of stack pointer will have to move to -1 to show an empty stack. Zero doesn't work

Queues



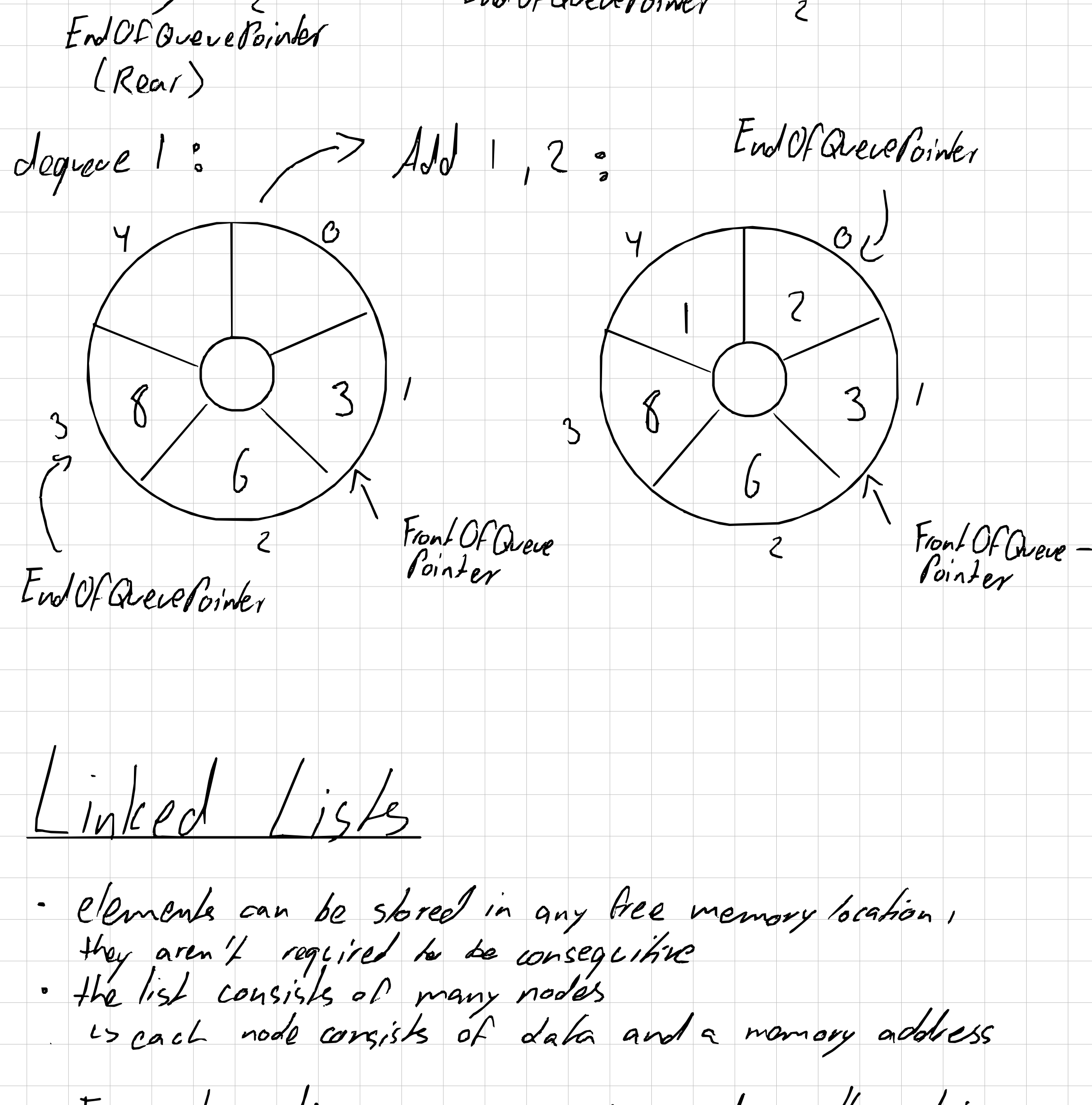
This indicates that there is no item in the queue, 0 and 0 would mean there is one item

when item leaves front:



- Items are enqueued and dequeued

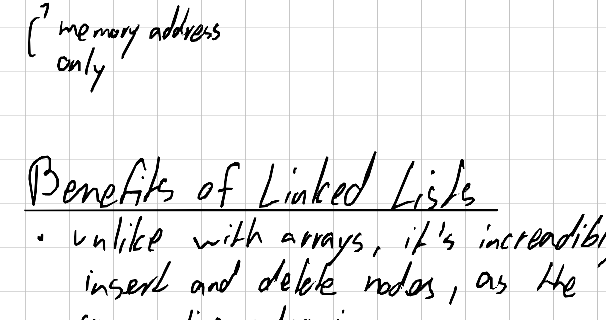
Circular Queues



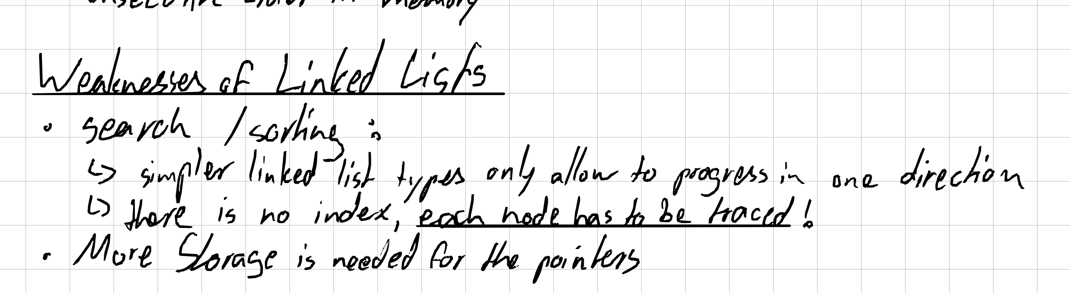
Linked Lists

- elements can be stored in any free memory location, they aren't required to be consecutive
- the list consists of many nodes
  - ↳ each node consists of data and a memory address

Example node:



* a node usually contains much more than just one variable
  ↳ usually contains record



Benefits of Linked Lists

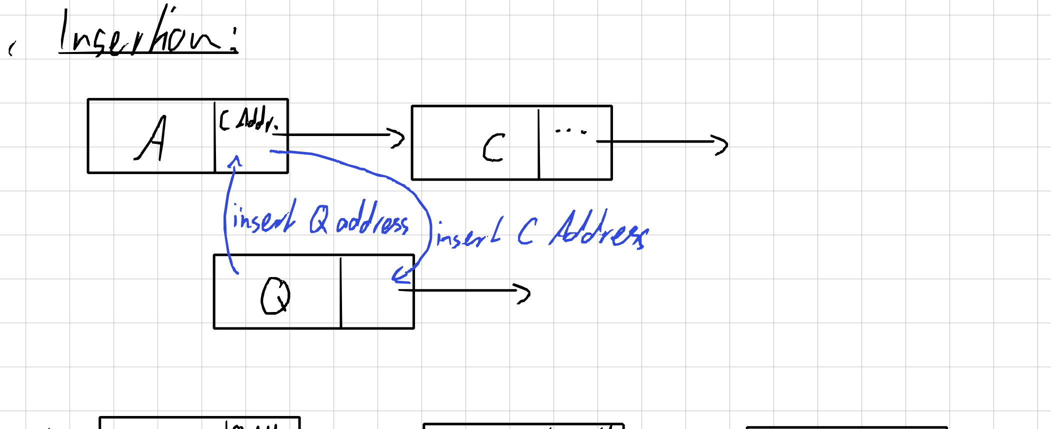
- unlike with arrays, it's incredibly efficient to insert and delete nodes, as the nodes aren't stored in consecutive order in memory

Weaknesses of Linked Lists

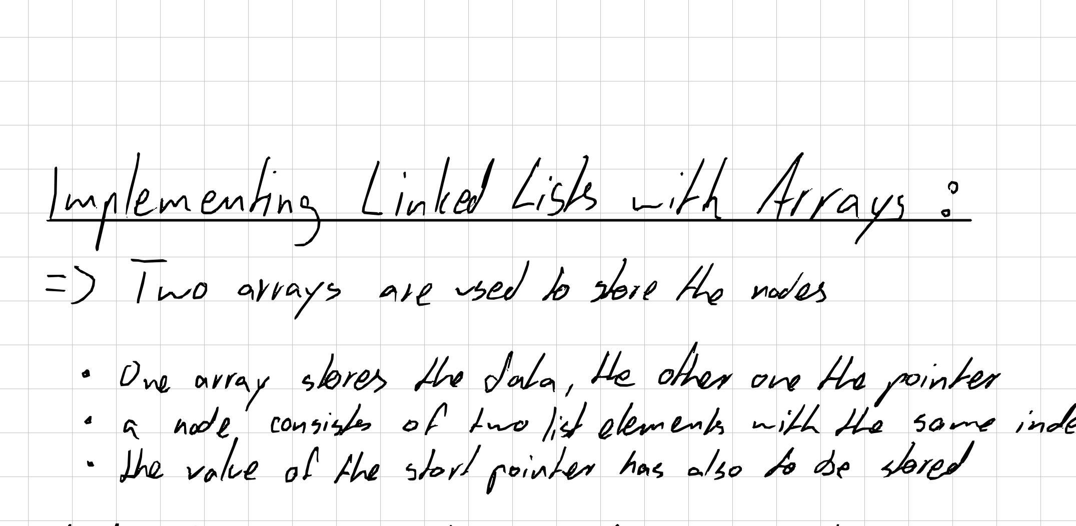
- search / sorting:
  - ↳ simpler linked list types only allow to progress in one direction
  - ↳ there is no index, each node has to be traced!
- More Storage is needed for the pointers

Insertion & Deletion

Deletion:



Insertion:



Implementing Linked Lists with Arrays:

=> Two arrays are used to store the nodes

- One array stores the data, the other one the pointer
- a node consists of two list elements with the same index
- the value of the start pointer has also to be stored

StartPointer: 1

Data	Index	Pointer
"like"	0	2
"I"	1	0
"apples"	2	3
"very"	3	4
"much"	4	-1
	5	
	6	
	7	

-1 acts as null pointer (Ø)

Deletion:
- The removed node remains in the array but it's inaccessible as no pointer is pointing to it.
- The deleted node can still have a pointer itself

=> Free lists should be created with all free (deleted) nodes

Free Lists

=> Unused nodes have to be easy to find so that they can be reused

- all free nodes including the deleted ones will be appended to the free list

Empty Linked List:

StartPointer: -1

FreeListPointer: 0

Data	Index	Pointer
	0	1
	1	2
	2	3
	3	4
	4	5
	5	6
	6	7
	7	8

↳ All elements are part of the free list

Partially filled Linked + Free List:

StartPointer: 0

FreeListPointer: 2

Data	Index	Pointer
42	0	1
420	1	3
	2	4
90	3	-1
	4	5
	5	6
	6	7
	7	-1

Two null pointers (3 and 7)

12.1 Software Development Life Cycle

=> Five fundamental stages
=> Four common models combining those stages

Stages

Analysis
- Investigate issues and old systems (if existent)
- Define problem clearly and precisely
- Plan a solution
- Decide approach: bottom-up: start with small sub-problems
  top-down: start broad (e.g. pseudocode) and refine down

Design
- Plan data structures, algorithms and required variables
  ↳ use identifier tables

Coding
- Decide on a suitable programming language
- Convert design (eg. pseudocode) into code
- Debug until it works (compiles / doesn't crash)

Testing
- write and perform tests

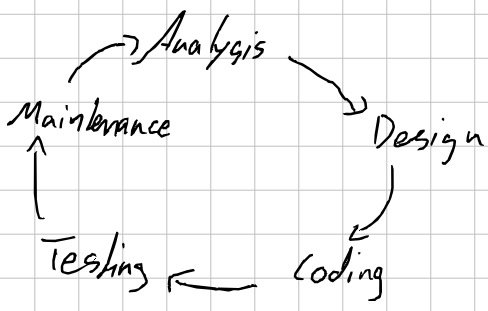
(Discussed later)

Maintenance

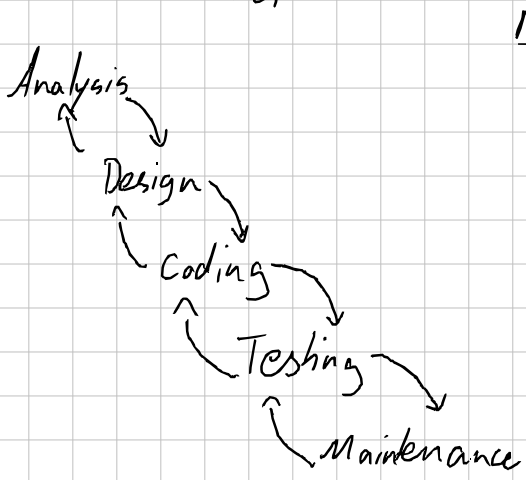
Program Development Life Cycle (PDLC)

=> Consists of many models that clearly define how the five steps of development are iterated over

most simple:



Waterfall Model



Principle:
- Starting with analysis, stages are completed one at a time
- Output from stage is input for the stage below
- more work might be required in the stage above, hence the loops

Benifits:
- simple, clearly defined
- easy to manage, predictable outcomes for each stage
- works well for smaller projects
  ↳ requirements need to be well understood

Drawbacks
- software not working until end
- not adaptable to changing requirements or new insights
- hard to measure progress, poor for long projects

Iterative Model

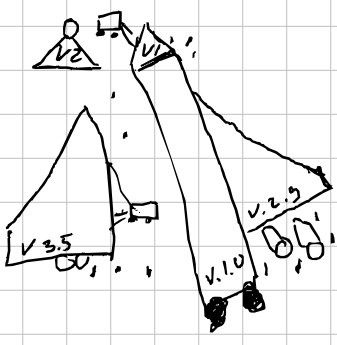
Principle
- start with a small subset of requirements
- after each full iteration add more requirements until system is completed

Benifits:
- early working project
  ↳ early risk identification
  ↳ faster feedback loop
- allows for parallel development
- testing for smaller program parts is easier

Drawbacks
- only large projects really profit
  ↳ simple projects can't be subdevided
- Design issues, as requirements aren't clear from start
- Might need more resources

Rapid Application Development (RAD) Model

Principle
- uses minimal planning
- prototypes for modules are developed and integrated
- The stages of PDLC are iterated over fast and many times
- modules will be changed and refined during development



Benifits:
- very addoptible to changing requirements
- Measurable progress
- Very productive when few people
- Reduces development time
  ↳ reusability of modules

Drawbacks
- only works for systems that can be build in modules
- requires skillfull coders
- mainly suitable for scalable systems
- requires user involvement during development

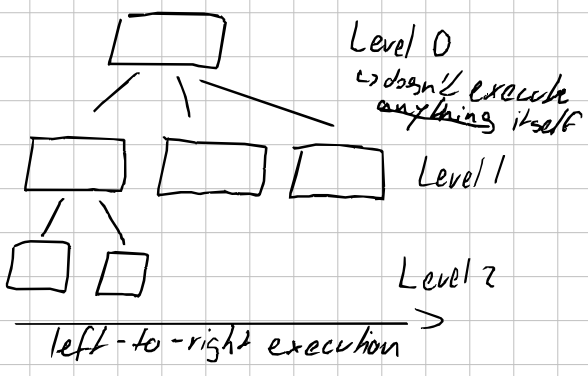
12.2 Structured Charts

=> a graphical representation of the modular structure of a solution

=> shows how modules are connected together through shared variables called "parameters"

Syntax

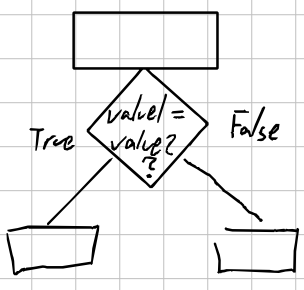
Pyramid structure



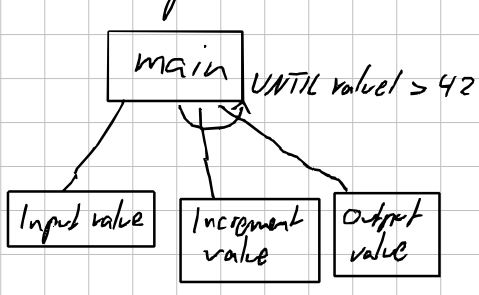
= module

—— = calls module

Conditions

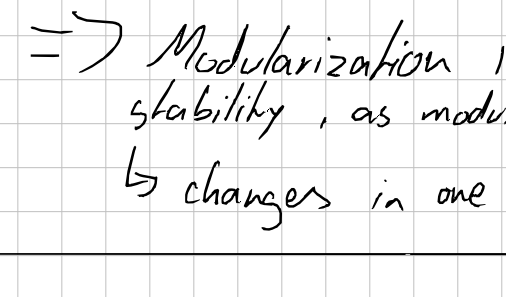


Loops



↷ loops over connections it intersects with

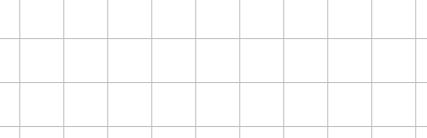
Variables



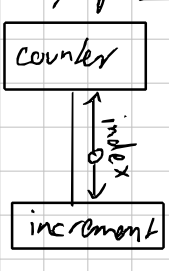
=> variable passing is called "interface"

↳ Interface between modules

Flags (aka Control Flow)



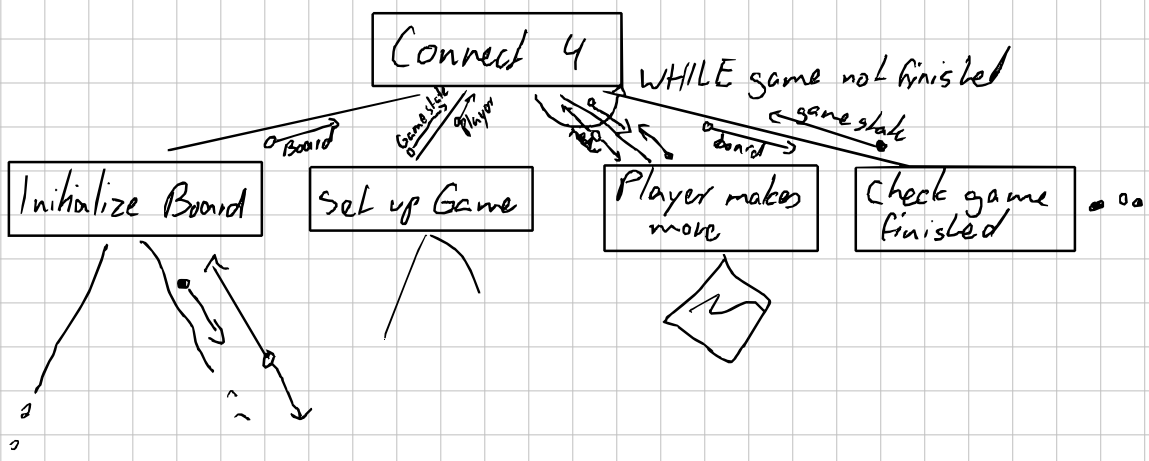
Modify passed value



<variable> = used when variable is a flag (true / false)

<value> = value is passed to module, modified, and then sent back

Full example



See page 292

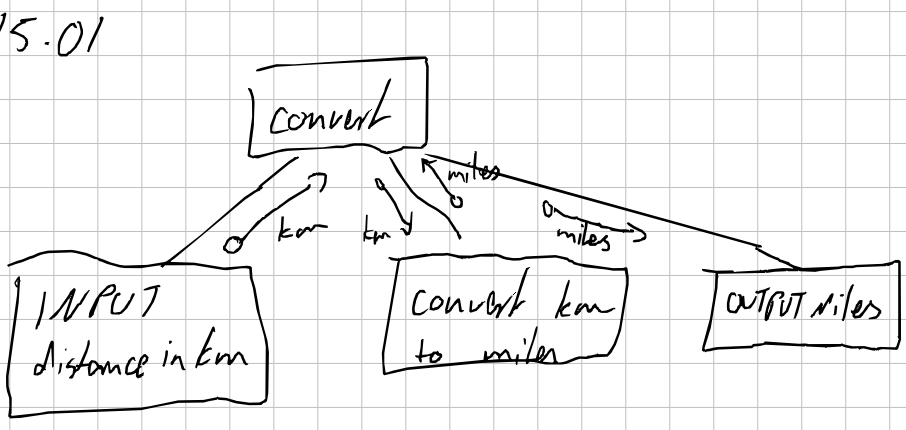
=> There are no stop nodes, all modules are expected to return the process to the top level after completion

=> Modularization improves the program's stability, as modules are self-contained

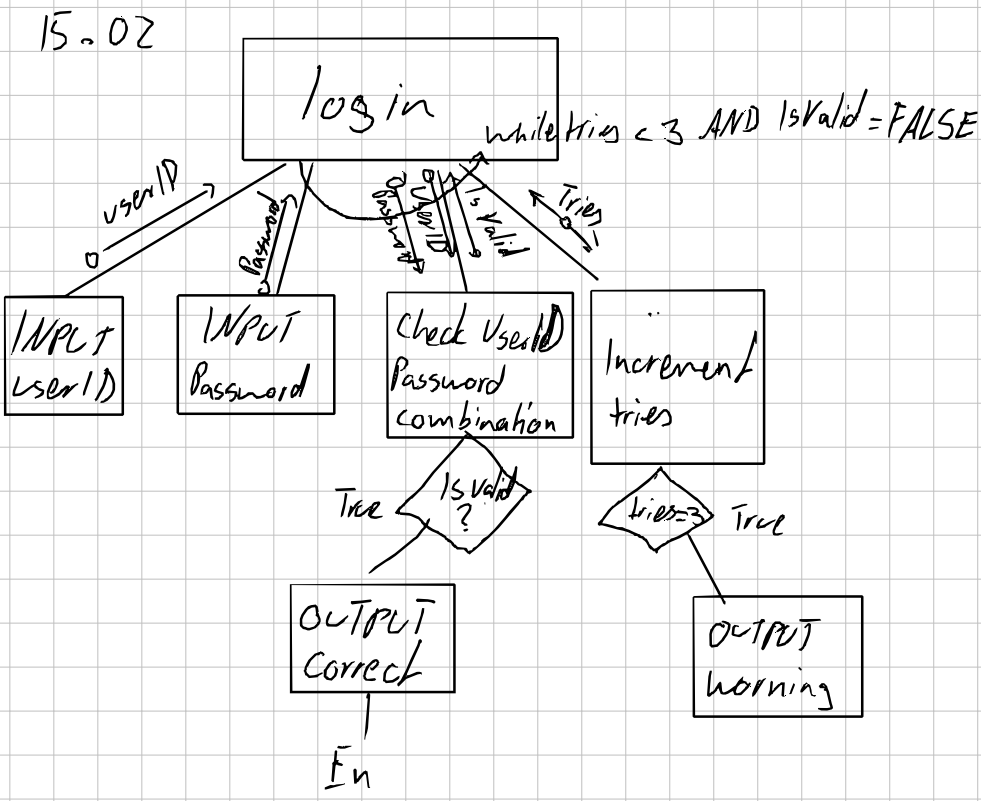
↳ changes in one module won't affect an other

Tasks

15.01



15.02



Exam style

3. a)
- A: Initiate tally array
- B: generate random number
- C: (valueToIncrement) randomNumber
- D: tally
- E: frequency table tally

b)

12.3 State Transition Diagrams

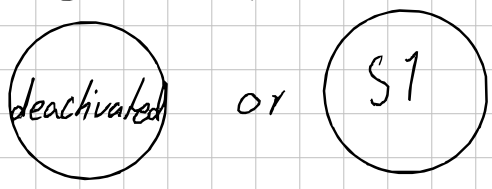
Finite State Machine (FSM)

=> A model where a program can be in any of a finite number of pre-defined states.

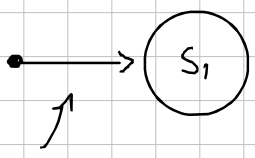
=> A variety of inputs can be entered and will cause a state transition

Syntax:

- A state is denoted by a circle containing a description:

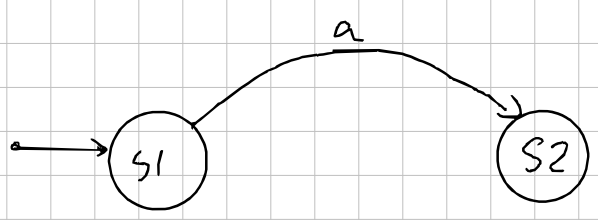


- State at which program starts:



- A state transition will be caused if a program is in a given state and a specific input is given.
  - The transition is indicated by an arrow and the condition is defined above:

where a = e.g. button 1 pushed



- Final states are marked by a double circle. (Optional)
  - Often there are no transitions from a halting state to another, but this isn't a requirement
  - halting states are states that are accepted to be the last state of the program

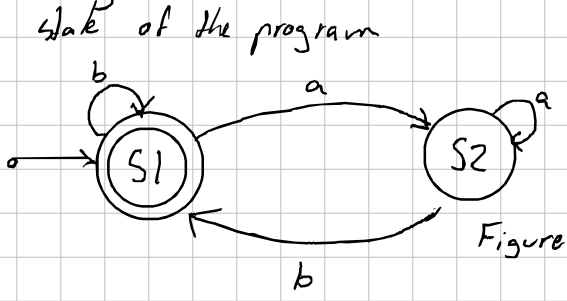


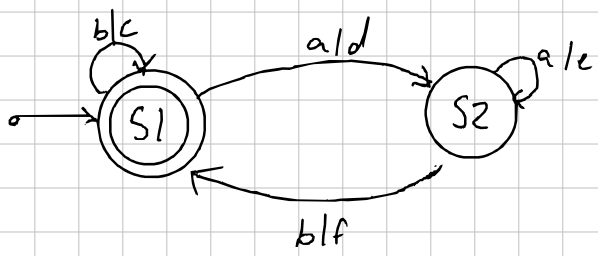
Figure 1

Only S1 is accepted to be the last state of the program here.

Finite State Machines with Output

aka "Mealy Machine"

- The output is noted on the right side of the input condition, seperated by a vertical line



=> When transitioning between states the program can give an output

State Transition Tables

For Finite State Machine:

Input	current state: S1	S2
a	S1	S1
b	S2	S2

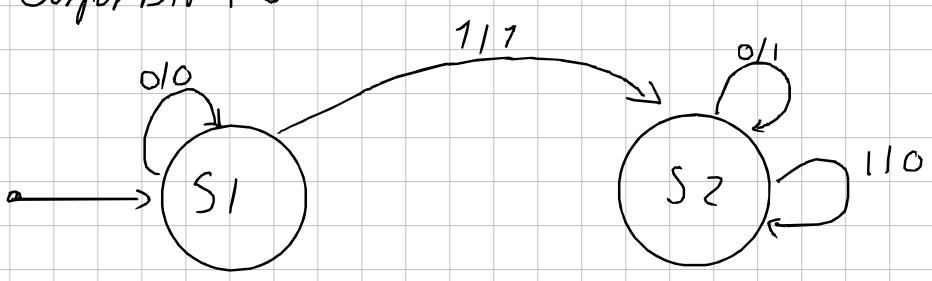
e.g. when the current state = S1 and input = b next state = S2

↖ for "Figure 1", see above

For Mearly Machine:

=> number of columns = number of states x variables

Current State	S1	S1	S2	S2
Input Bit	0	1	0	1
Next State	S1	S2	S2	S2
Output Bit	0	1	1	0



15.01

-6 0110

1010 ✓

Exam Style #4

17.4 Errors & Tests

Types of Errors

Syntax Error:

=> Program statement defies the rules of the programming language

- The compiler or Interpreter won't understand the statement
  -> a compiler can only compile syntax-error-free code
  -> an interpreter will only stop when reaching a syntax error

Run-Time Error:

=> While executing, the program unexpectedly halts or crashes

Causes:
- memory overflow
- access undefined variable
- infinite loop floods RAM, freezes program
- division by zero, etc...

- Run-Time Errors won't be flagged
  ↳ Testing is required to uncover any errors

Logic Error:

=> The program doesn't do / output what is expected, due to a logical design flaw

Causes:
- Bad planning
- Incompetence

- Logic Errors won't be flagged and don't throw errors on execution
  ↳ Requires testing where output is compared with expected output

Types Of Testing

Stub Testing

stub trans. German = Stummel

=> Functions and procedures are empty and contain print messages to acknowledge they were being called

- great for testing top logic in a program / GUI testing

Black-Box Testing

=> Someone without knowledge about the source code looks at the program speifications and designs test data + expected outputs

Test Data
- Three types of test data will be created
  - Normal Data: Explicitly valid data a user might enter
  - Extreme / Boundry Data: checks minimums, maximums, including just-out-of-bound (invalid) values
  - Erroneous / Abnormal Data: Explicitly invalid data, tries to break code

White-Box Testing

=> The programmer creates tests that check every possible path through the code

- Data is selected so that:
  - all sides of the conditions are checked
  - every submodule gets executed

Dry-Running

=> Using a trace table to record all variable states the program is run through, line-by-line

=> All variable changes and outputs are recorded in the trace table

Example:

```
x ← 2
FOR i ← 1 TO 3
    y ← x * i
NEXT i
OUTPUT y
```

x	i	y	Output
2			
	1	2	
	2	4	
	3	6	
			6

↑ spaces are only filled on changes

Rules:
- new row for each iteration
- new row for function / procedure call

Stages Of Testing

Module Testing (my addition)
- tests previously covered are used to check single modules

Integrity Testing

=> Checking if all modules work together by adding one module at a time and seeing if the program works as expected

Alpha Testing

=> Software will be tested in-house by software testers, before it's being released

Acceptance Testing
- done when a program is developed for a specific customer (e.g. company)

=> Customer checks if the software meets all requirements

=> If successful, software will be signed off

Beta Testing

=> Software released to the general public is first released to a limited audience of "beta testers"

=> The released "beta version" will collect feedback and problem reports
  ↳ Most problems will be patched by the time of the official release

Test Strategy

=> A test strategy is required to ensure the product is rigorously tested

Create Test Plan

=> Strategically create a plan with tests to run that will find as many errors as possible and explore wide parts of your system

=> Start with an outline test plan and get very specific in a detailed test plan

Test Data

=> Based on the test plan design data sets of input values and expected outputs in a table

=> More about the three test data categories above under "Black-Box Testing"

How to minimize errors

- use existing libraries
- commonly accepted algorithms and design techniques
  ↳ e.g. object-oriented-programming

Exam Style #1, 2

1. a) i returns "1011"

   ii returns "101"

   b) The first dry run runs successful, as

   11 = 1011 in binary

   The second dry run fails, 10 is 1010, but 101 is returned

   Currently the program stops early, as soon as the last one has been written. Therefore 1010 is shortened to 101.

   Fix: Stop when last place value is written instead

   ~~UNTIL Number = 0~~

   -> UNTIL PlaceValue = 0

2.

			Expected output
Normal	1 5 20	Some normal entries testing a range of likely entered values	/ ////\ ////\ ////\ ////\ ////\
Boundary	0 -1 3000	0: Testing what happens if the lowest valid number is entered -1: Testing a number just out of range 3000: Testing an unexpectedly high value	empty line empty line ////\ ////\ ... ↑ showing out of bounds
Abnormal	'a' 3.5	Testing if program can handle wrong data types. Testing if program is robust enough to deal with reals	empty line empty line

12.5 Program Maintenance

=> Programs often have to be maintained to keep working as intended and to stay up-to-date with constantly changing customer needs

=> There are three common types of maintenance

Corrective Maintenance

=> Unfixed errors will often be discovered way past the initial release in rarely used parts of the code

↳ new errors have to be patched regularly

Adaptive Maintenance

=> The program might be changed or augmented to stay relevant to the customers

=> Adapt to specification changes, e.g. changing external APIs

Perfective Maintenance

=> Program fulfills all needs but is further improved

↳ e.g. more effective algorithm or faster loading speed

Write Maintainable Code

- Use Indentation
- Use descriptive names for variables and subroutines
- write comments explaining and defining sections of code
- comment on any unusual practices you implemen

15.08

1 0 1 0

$1 \times 0 + 1 = 1$

$2 \times 1 + 0 = 2$

$2 \times 2 + 1 = 5$

$2 \times 5 + 0 = 10$

Instead of

~~Denary Value + 2 + Bit Value~~

Denary Value * 2 + Bit Value

15.09

$$x = v_i t + \frac{1}{2} a t^2$$

$$\frac{2x}{a} = t^2$$

$$\uparrow$$

$$y$$