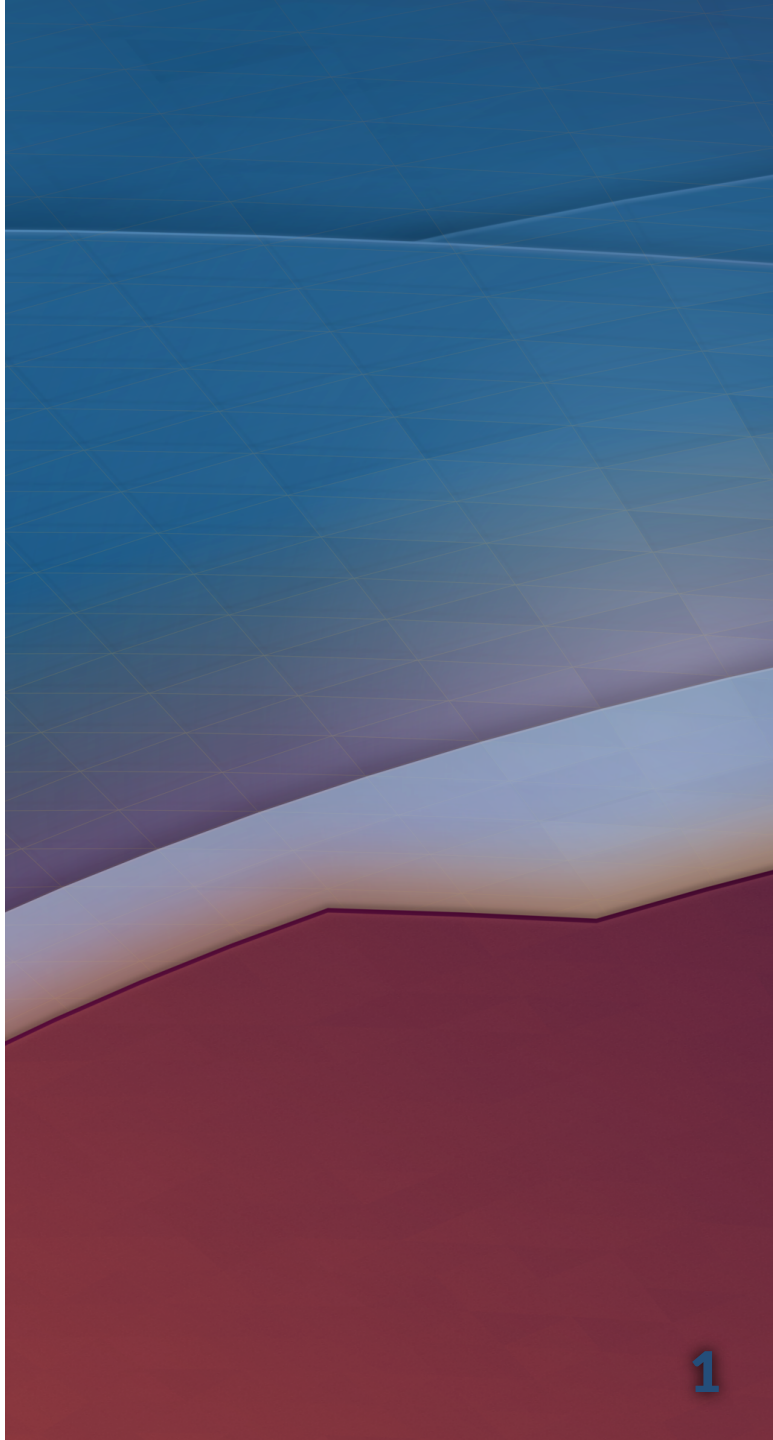
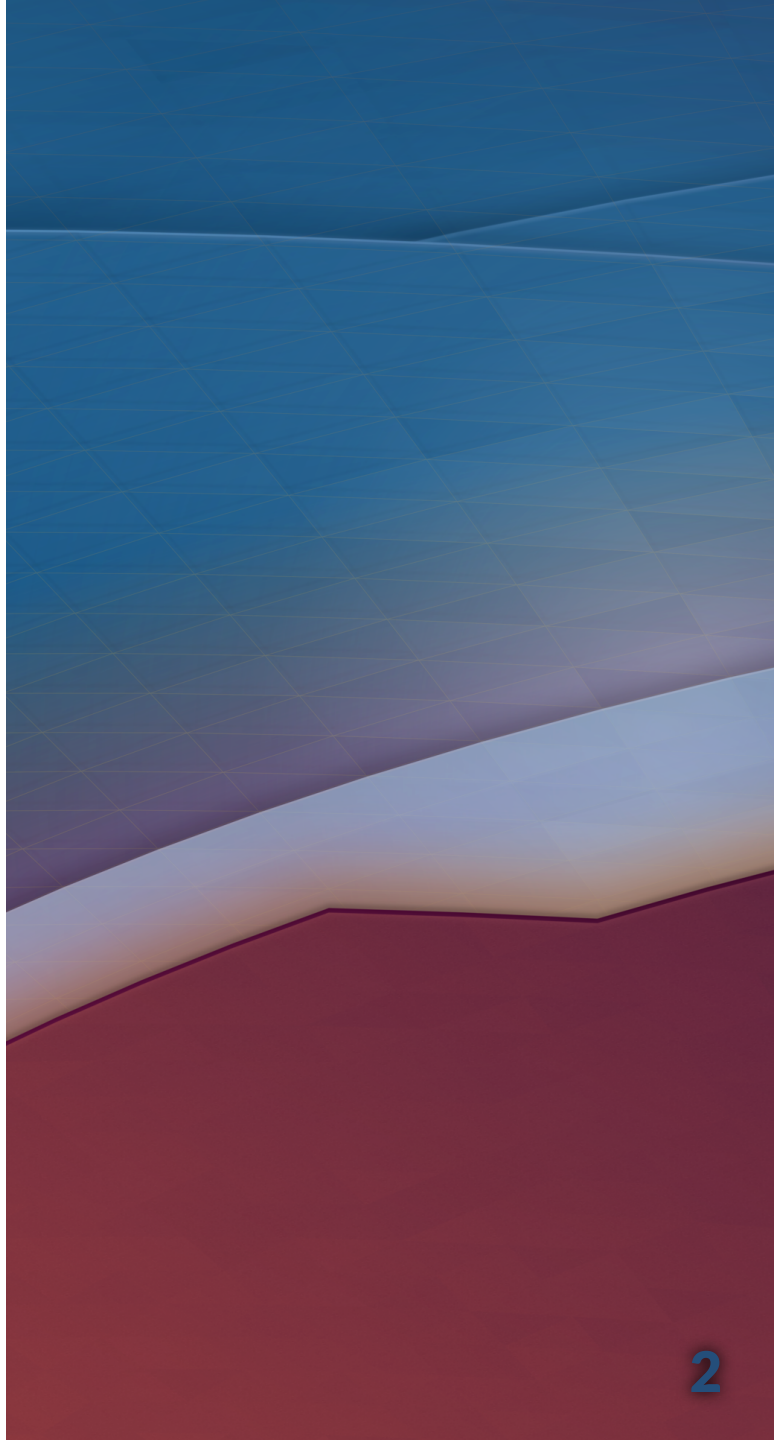
Tutorat 3

IO-Devices



Betriebssysteme, Tutorat 3, Gruppe 6, juergmatth@gmail.com, Universität Freiburg

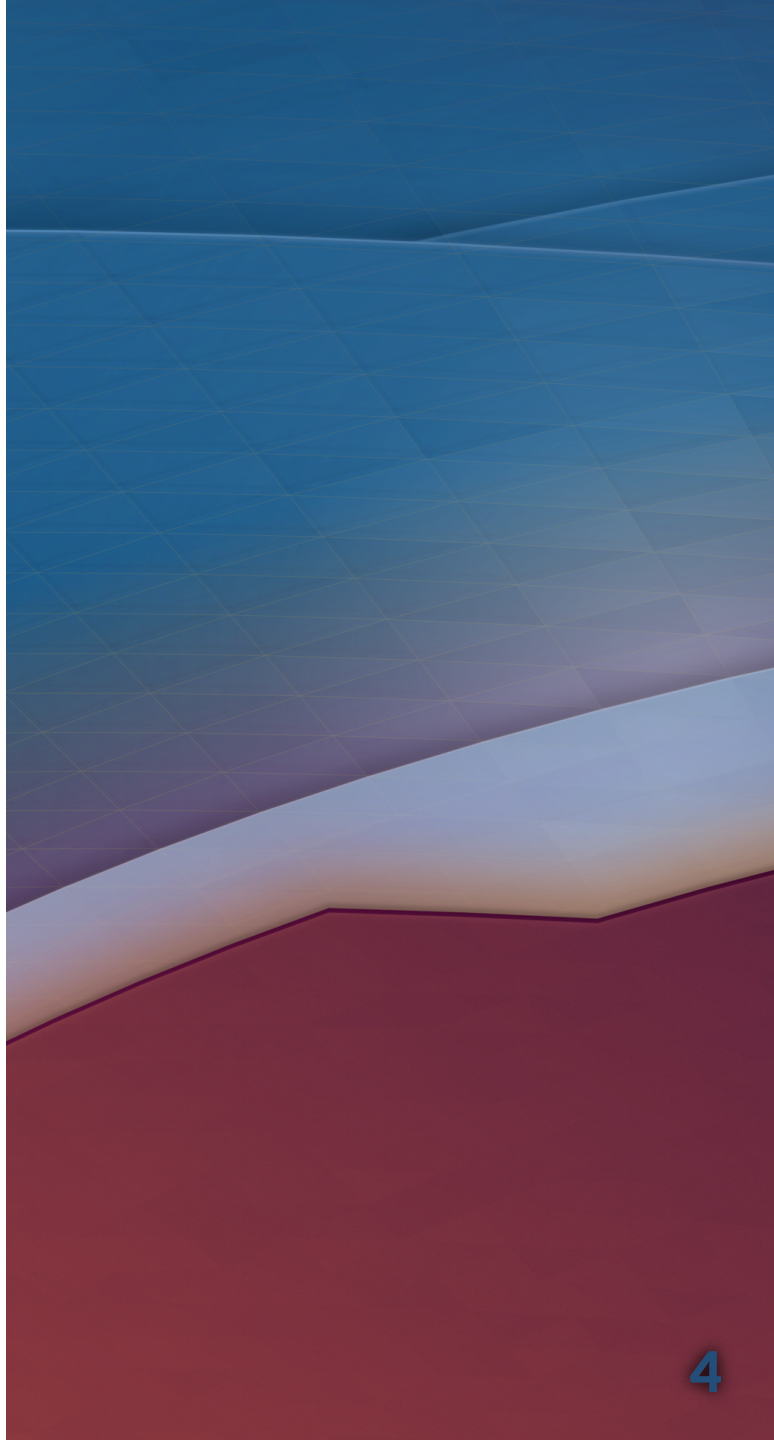
Einstieg



Einstieg
Fakeupdate

- https://fakeupdate.net/

Korrektur

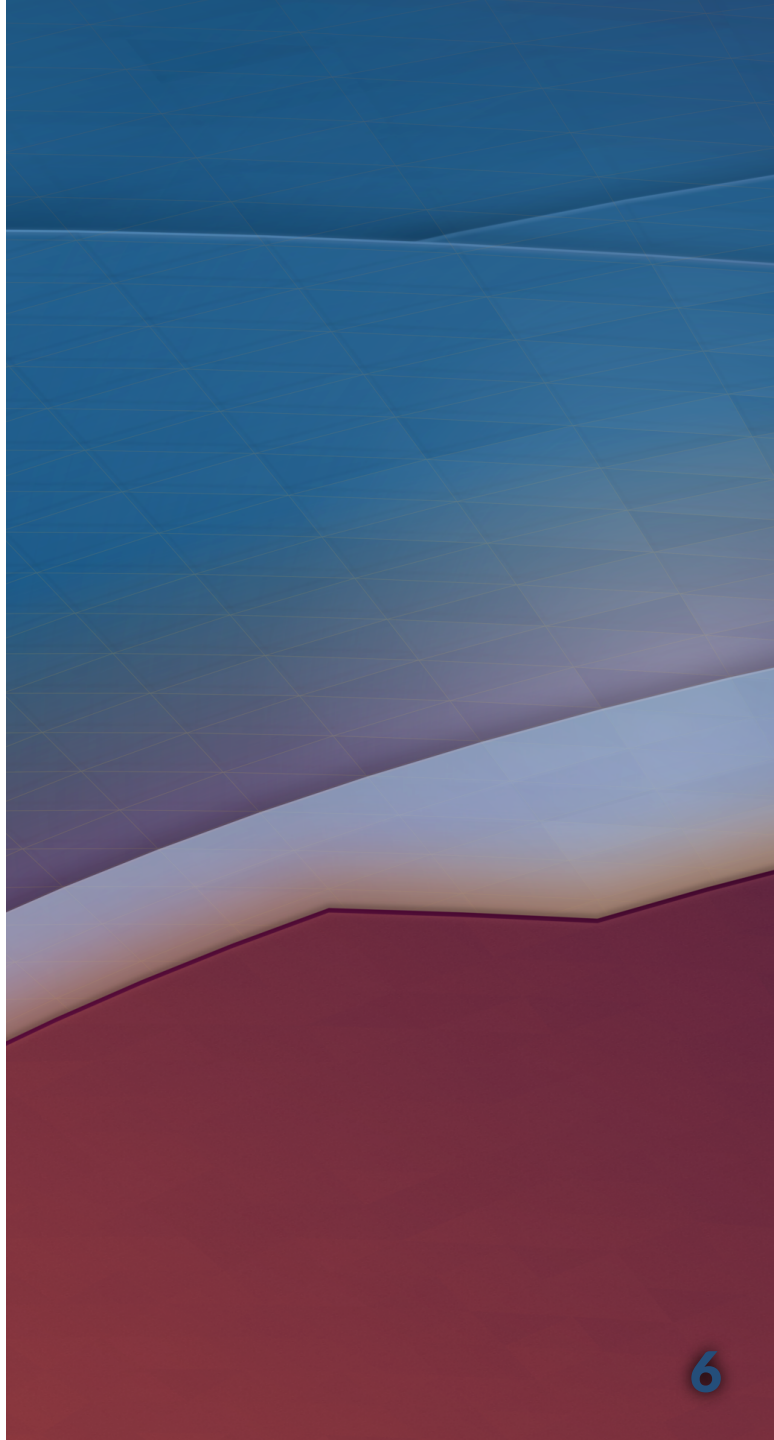


Korrektur

Interessantes und häufige Fehler

- ein paar Fehler bei der **RETI Treiberaufgabe**
- viele kleinere Fehler bei der **push und pop** Aufgabe
- **Aufgabe 3** haben sich viele gespart
- **Usermodus** und **Kernelmodus** hatten einige Fragen
- **Terminal** Bedienung
- die Sache mit `<SP>` und `[SP]`
- **Packagemanager**, unter Linux Sachen installieren
- **kein Feedback:** https://forms.gle/2tGvF4ao5hAVNeRs5

Korrektur



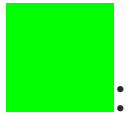
Korrektur

Interessantes und häufige Fehler

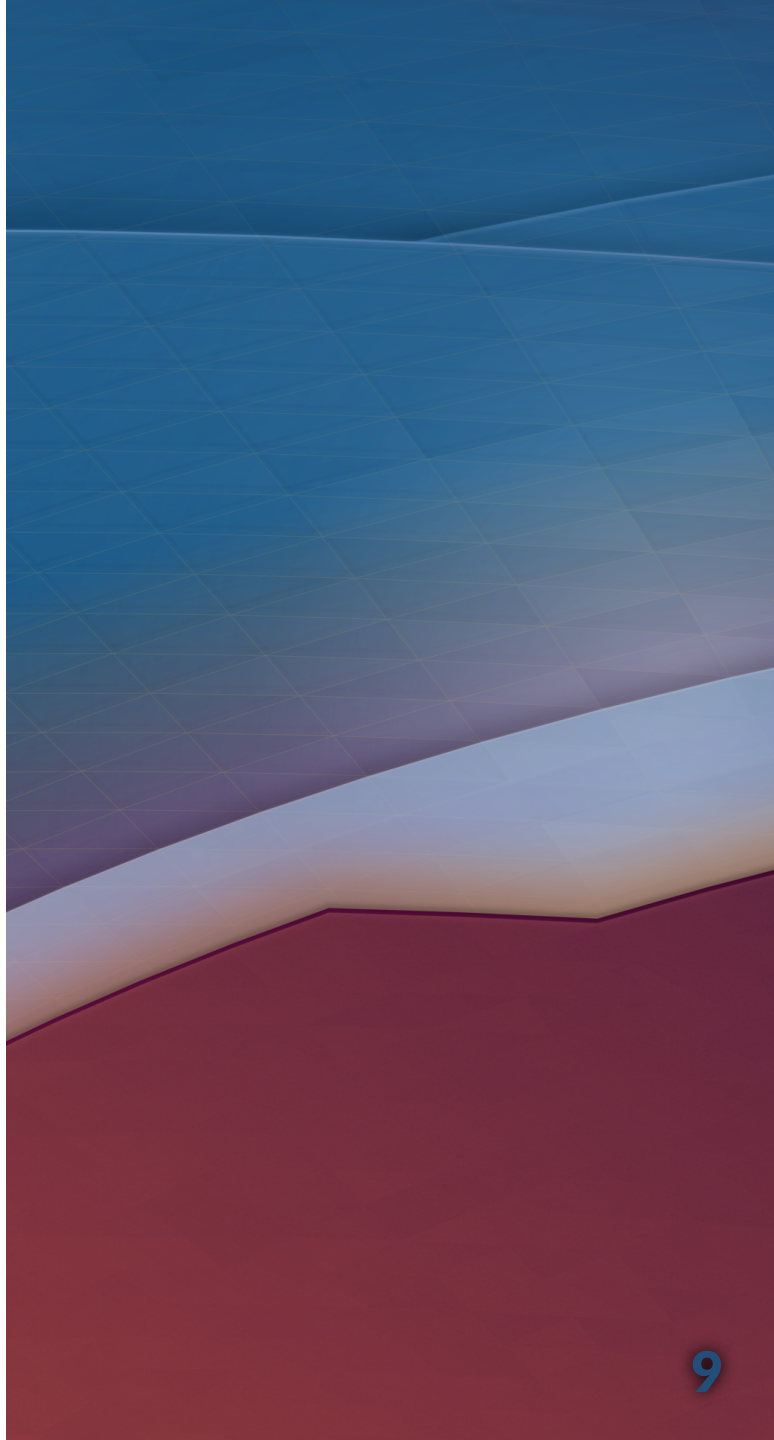
- überschreiben der Daten, die in der b) nach links geshiftet wurden. `OR IN1 1`, **Non-Controlling** Bit `0`
- ~~nach dem **Polling erst shiften~~ → am Ende um 8 Stellen zu viel gehiftet
- `SUBI ACC 2` um `b1` `0` zu setzen
  - einfach `0` setzen geht nicht, weil die anderen **Flags** des **Statusregisters** erhalten bleiben sollen
- neuen 8Bitvektor dranfügen aus **Empfangsregister** `ADD IN1 1`
- der **EPROM** ist **READONLY** → hat keinen Stack
- andere **Flags** des **Statusregister** nicht überschreiben, nur das Bit, was geändert werden soll
- bei `JUMPc i` beschreibt das `i`, wie oft man die Speicherzelle wechselt, und zwar von der Speicherzelle, wo das `JUMPc i` steht aus (wie `<count>j` in **(Neo-)Vim**)

Korrektur

Korrektursystem

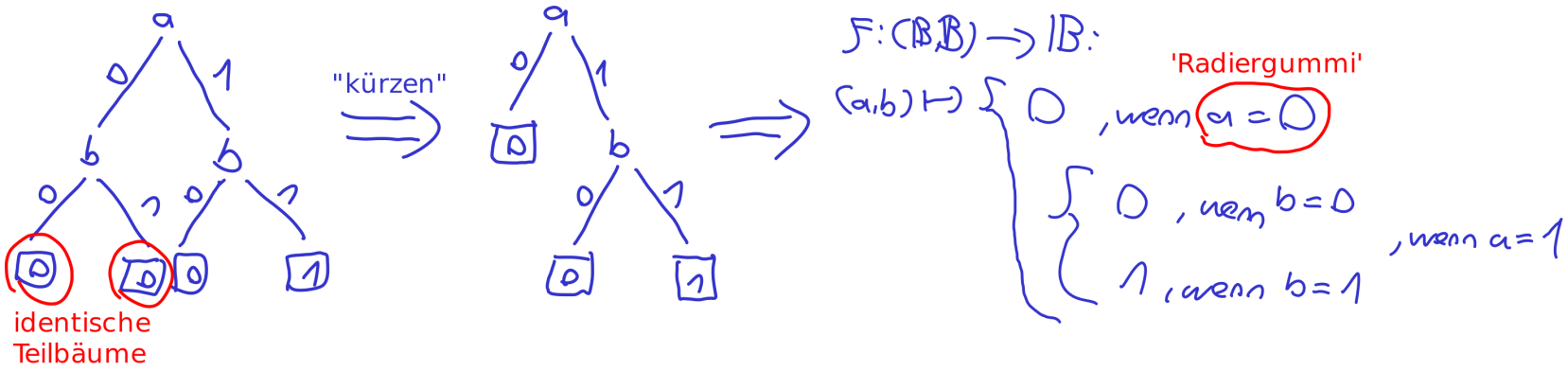
- **Punkte** sind nur zum Vergleich untereinander
- **Ampelsystem:**
  -  : Sehr gut, damit ist man für die Klausur auf der sicheren Seite
  - : Nur ein Hinweis darauf, dass da einige klausurrelevante Sachen vielleicht nochmal vielleicht über das Tutorat nachvollzogen werden sollten
  -  : Nicht ausreichend. Leider zu wenig Arbeitsaufwand investiert

Vorbereitung



Vorbereitung
Bitweise Logiktricks

- **Bestimmte Bits auf `0` setzen u. alle anderen unverändert lassen (*Maskieren*):**
  - `___ 10100111 00101101 10010100 00000100`
    `& 00000000 00000000 00000000 11111111` (***Maske***)
    `___ 00000000 00000000 00000000 00000100`
  - `0` ist **controlling value** zum mit `0`en **überschreiben**
  - **Herleitung über Decision Tree:**



Vorbereitung
Bitweise Logiktricks

- **Bestimmte Bits auf `1` setzen u. alle anderen unverändert lassen (*Union*):**
  - `___ 10100111 00101101 10010100 01100101`
    `| 00000000 00000000 00000000 11111111`
    `___ 10100111 00101101 10010100 11111111`
  - `1` ist **controlling value** zum mit `1`**en überschreiben**
- **Test auf bestimmten Bitwert:**
  - **non-controlling value** von `&` bzw. `|` nutzen, um ein bestimmtes Bit **unverändert beizubehalten** und dann aus diesem bzw. dessen Negation zu schlussfolgern, dass da eine `1` bzw. `0` steht
  - mit `JUMP<> i` testen, ob z.B. **Bit 3** von `REG` `1` bzw. `0` ist. Dazu `ACC = REG & 00000100` bzw. `ACC = ~(REG | 11111011)` und dann: `<PC> + [i]` *gdw.* `ACC` $\neq$ `00000000` *gdw.* **Bit 3** ist `1` bzw. `0`

Vorbereitung
Bitweise Logiktricks

- **Bestimmte Bits negieren und alle anderen unverändert lassen (*Differenz*):**
  - `___ 10100111 00101101 10110100 01100101`
    `⊕ 10111100 10101001 00000000 11111111`
    `___ 00011011 10000100 10110100 10011010`
  - **Unterschiede** werden hervorgehoben
  - `1` ist **controlling value** zum **Negieren** von `0` zu `1` bzw. `1` zu `0`
  - `0` ist **non-controlling value** zum **unverändert Beibehalten**
- **Test auf Gleichheit:**
  - mit `JUMP= i` testen, ob zwei Register gleiche Bitworte haben. Dazu `ACC` = `REG1` ⊕ `REG2` und dann: `<PC> + [i]` *gdw.* `ACC` = `00000000` *gdw.* `REG1` = `REG2`

Vorbereitung
Bitweise Logiktricks

- **Bitshiften :**
  - Shiften um **3** Stellen nach **links**
    - `10110 x 1000 = 10110000`
  - Shiften um **3** Stellen nach **rechts**
    - `10110000 / 1000 = 10110`
  - Zahl finden, die **Logarithmus 2** den passenden Wert (hier: **3**) hat bzw. entsprechende Anzahl `0`en hat (hier: **3** `0`en)
    - `log2(8) = 3`, also hat **3** `0`en ➞ passt

“ • Kann man auch als eine Art "**Signextension**" nach links vom **niedrigstelligsten** Bit aus ansehen ”

Vorbereitung

Signed (2er Komplement oder 1er Komplement) und Unsigned

Unsigned (oder Betrag mit Vorzeichen)

- $< x >= x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$
- $[x]_{BV} = (-1)^{x_{n-1}} \cdot (x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0)$
- **Bereich:** 0 bis $+2^n - 1$ oder $-2^{n-1} + 1$ bis $+2^{n-1} - 1$

Signed (2er Komplement)

- $[x]_2 = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$ (weil `1000` - `1` = `111`)
- **Bereich:** -2^{n-1} bis $+2^{n-1} - 1$ (die 0 muss auch iwo hin)

Vorbereitung

Signed (2er Komplement oder 1er Komplement) und Unsigned

Signed (1er Komplement)

- $[x]_1 = -x_{n-1}(2^{n-1} - 1) + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$
- **Bereich:** $-2^{n-1} + 1$ bis $+2^{n-1} - 1$ (es gibt **2** Kodierungen für die `0`)

Vergleich Kodierung von Unsigned, Signed im 1er und 2er Komplement

x	000	001	010	011	100	101	110	111
$[x]_{BV}$	0	1	2	3	0	−1	−2	−3
$[x]_2$	0	1	2	3	−4	−3	−2	−1
$[x]_1$	0	1	2	3	−3	−2	−1	0

Vorbereitung

Signed (2er Komplement oder 1er Komplement) und Unsigned

Kodierung Bedeutungen

- **Höchstwertiges** Bit ist **Sign Bit**, `1` für **negativ**, `0` für **positiv**
- `<i>` **unsigned** und `[i]` **signed**
- **Little Endian**=niedrigstwertiges Bit zuerst, **Big Endian**=höchstwertiges Bit zuerst

Vorbereitung

Signed (2er Komplement oder 1er Komplement) und Unsigned

Interessante Zahlen für 2er Komplement

- `_0`: `0000 0000 ... 0000`
- `-1`: `1111 1111 ... 1111`
- **Negativste Zahl:** `1000 0000 ... 0000`
- **Positivste Zahl:** `0111 1111 ... 1111`

Signed Negation (2er Komplement)

- $\bar{x} + 1 = -x$ (1er Komplement Negation + 1, da $x + \bar{x} = 1111 \ldots 111_2 = -1$)

Vorbereitung

Signextension

- **von 8 Bit auf 16 Bit:**
  - `+2` : 0 `000 0010 => 0000 0000` 0 `000 0010`
  - `−2` : 1 `111 1110 => 1111 1111` 1 `111 1110`
- **unsigned** wird mit `0`en extendet
- das **Sign Bit** wird nach **links** dupliziert

Vorbereitung

Merkhilfe RETI Befehlssatz

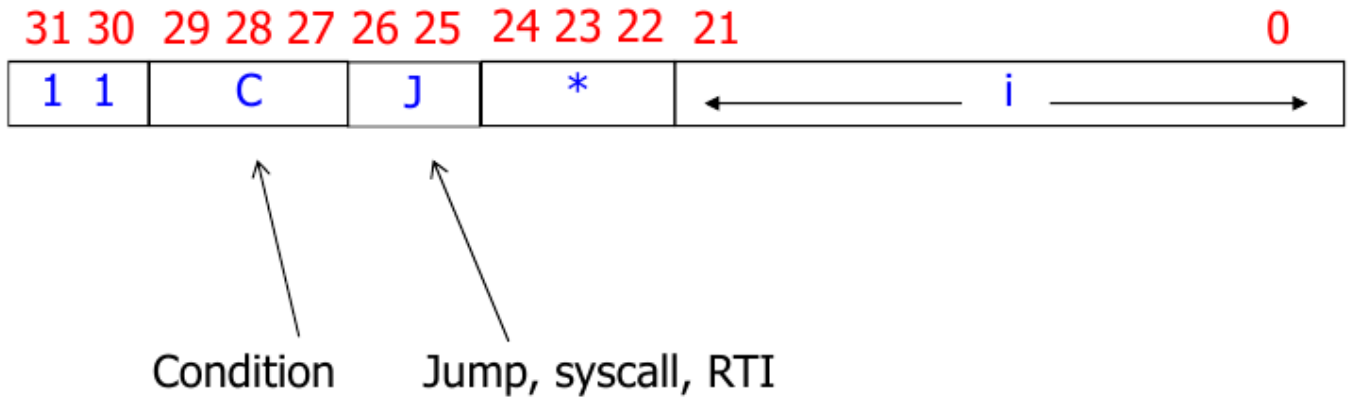
- **to** `X` = **from** `X`
- **Compute:** *calc* `D` `OP` `S` *to* `D`, *calc* `D` `OP` `M(<i>)` *to* `D`, *calc* `D` `OP` `[i]` *to* `D`
- **Load:**
  - `LOAD` *to* `D` *from* `M(<S>)` und `LOADI` *to* `D` *directly from* `i`
  - `LOADIN` *from* `M(<S>+[i])` *to* `D`
- **Store:**
  - `Store` *from* `D` *to* `M(<S>)` und `move` *from* `D` *directly to* `S`
    - es gibt kein `STOREI`, da die erweiterte RETI und vor allem der **SRAM** nicht dazu konzepiert sind, dass **zwei Argumente** beide auf den Speicher zugreifen
  - `STOREIN` *to* `M(<S>+[i])` *from* `D`

Vorbereitung

Merkhilfe RETI Befehlssatz

- **Jump:** `JUMPc i` *gdw.* `ACC c 0`
  - mache `JUMPc i` *gdw.* `3 < 4` *gdw.* `3 - 4 < 0`
- **Kodierung der Condition:**

C	Bedingung c
0 0 0	nie
0 0 1	>
0 1 0	=
0 1 1	≥
1 0 0	<
1 0 1	≠
1 1 0	≤
1 1 1	immer



<=>
011 ist '>'

<=>
101 ist '≠'

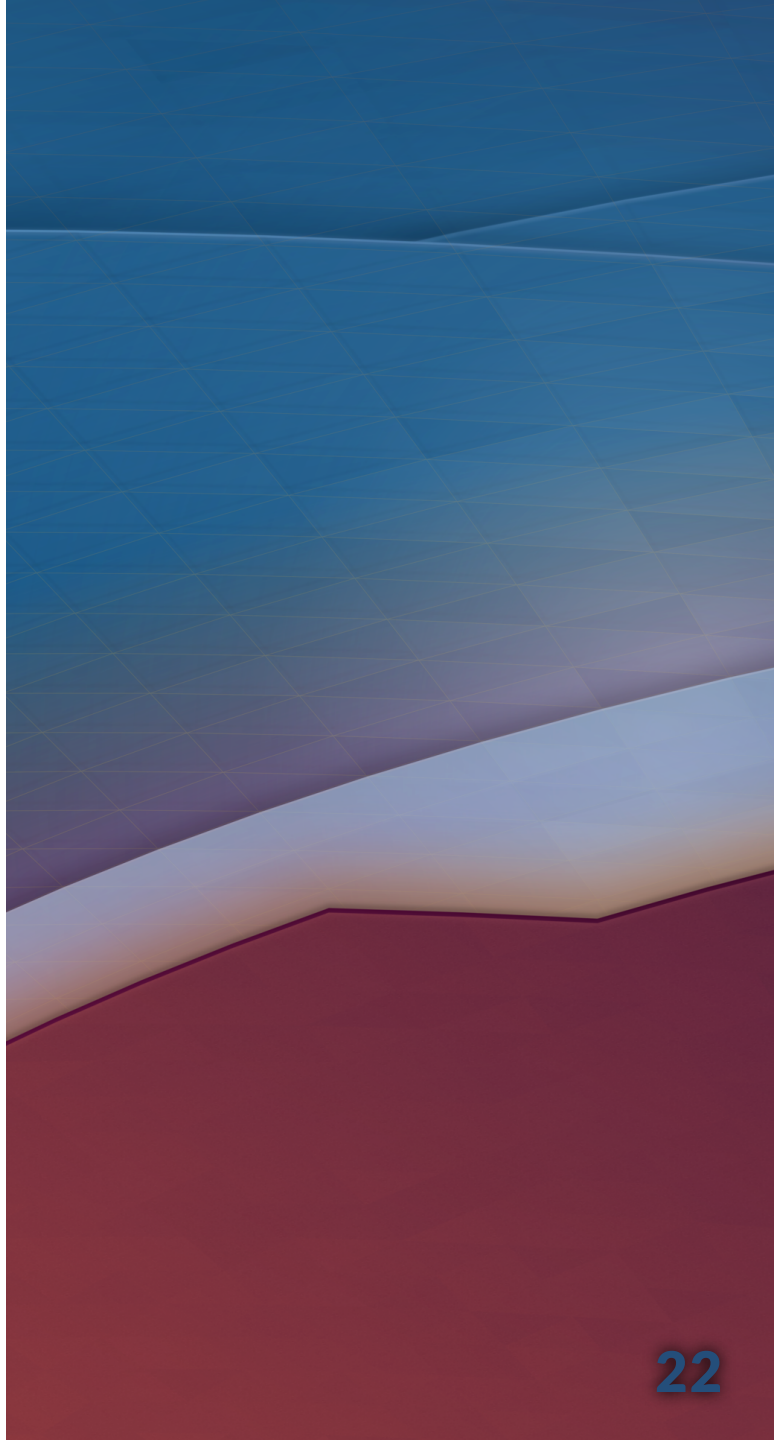
<=>
010 ist '='

Vorbereitung

Datensegmentregister

- Solange im `DS` die Bits `30` und `31` mit dem gewollten **Präfix** besesetzt sind muss man sich keine Sorgen machen
- **Verändern kann man die beiden Bits durch:**
  - durch `LOADI DS 0` z.B. mit `0`en überschrieben durch **Signextension**
  - wenn man durch **Multiplikation** andere Bitwerte an Stelle `31` und `30` shiftet
  - oder wenn man den `DS` mit einem anderen **Register** oder **SRAM Inhalt** überschreibt, die `32` Bit lang sind

Übungsblatt



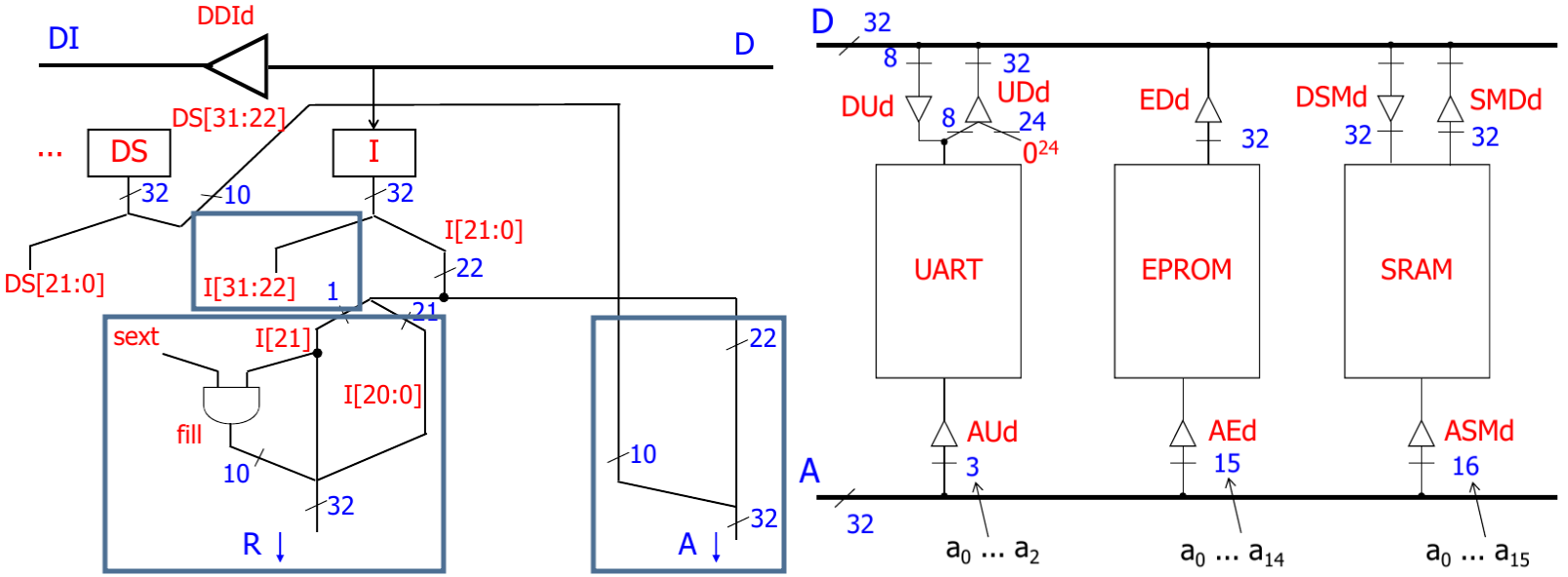
Übungsblatt

Aufgabe 1

- **auf verschiedene Register der UART zugreifen:** `__000000 00000000 00000XXX`
  - **R0:** `XXXXXXXX`, Senderegister (**Senden** an Peripheriegerät)
  - **R1:** `XXXXXXXX`, Empfangsregister (**Empfangen** vom Peripheriegerät)
  - **R2:** `X,X,X,X,X,X,b1,b0`, Statusregister (**Little Endian**)
    - `R2[0] = b0`: `senderegister_befuehlbar`
    - `R2[1] = b1`: `empfangsregister_befuehlt`
  - **R3-7:** `XXXXXXXX`
- **Vorgefertige Adressen im EPROM:** `r/s/t = 00XXXXXX XXXXXXXX XXXXXXXX XXXXXXXX`
  - **UART Konstante:** `EPROM[r] = 01000000 00000000 00000000 00000000`
  - **SRAM Konstante:** `EPROM[s] = 10000000 00000000 00000000 00000000`
  - **`LOADI PC 0` als Konstante:** `EPROM[t] = 01110000 00000000 00000000 00000000`

Übungsblatt
Aufgabe 1

- **Adressbus bekommt:** `DSDSDSDS DSIIIIII IIIIIIII IIIIIIII`
- **Kontrollogik bekommt:** `IIIIIIII II______ ________ ________`
- **Rechter Datenbus bekommt:** `SSSSSSSS SSIIIIII IIIIIIII IIIIIIII`
  - *Signextension* `S`, *Instruktionsregister* `I`, *Datensegmentregister* `DS`



Übungsblatt

Aufgabe 1

- **zu UART wechseln:**

```
LOADI DS __010000 00000000 00000000
MULTI DS __000000 00000100 00000000
```

- **zu SRAM wechseln:**

```
LOADI DS __100000 00000000 00000000
MULTI DS __000000 00000100 00000000
```

- **zu EPROM wechseln:**

```
LOADI DS __000000 00000000 00000000
```

- `LOADI DS 0` füllt wegen **Signextension** die ersten 10 Bits autoamtisch mit `0`en auf, daher braucht man nur einem einzigen Befehl für den **EPROM**

Übungsblatt
Aufgabe 1

- **Versenden:**

```
if (senderegister_befuehlbar == 1) {  // R2[0] == 1
  write_data(R0);
  R2[0] = 0;
}
// else: warten, denn die UART versendet gerade noch Inhalt von R0 ans
// Peripheriegerät
```

- **Empfangen:**

```
if (empfangsregister_befuehlt == 1) {  // R2[1] == 1
  read_data(R1);
  R2[1] = 0;
}
// else: warten, denn die UART ist noch beim Fühlen des Registers, die UART
// wird sobald sie fertig ist R2[1] = 0; auf 1 setzen
```

Übungsblatt
Aufgabe 1a)

- **C-Code:**

```
polling_loop(int new_instruction) {
  uart_selektieren()
  while (empfangsregister_befuehlt == 0) {  // R2[1] == 0
    // warten, denn die UART ist noch beim Fühlen des Registers, die UART
    // wird sobald sie fertig ist R2[1] = 0; auf 1 setzen
  }
  new_instruction[7:0] = R1;  // IN1[7:0] = R1
  R2[1] = 0;
}
```

- `while (1) {if (empfangsregister_befuehlt == 1) { }}`
  - ➞ `while (!(empfangsregister_befuehlt == 1)) { }`
  - ➞ `while (empfangsregister_befuehlt == 0) { }`

Übungsblatt
Aufgabe 1b)

- **C-Code:**

```
void instruction_loop(int new_instruction) {  // IN1 = 0
  int counter = 4;  // IN2 = 4
  while (counter > 0) {
    new_instruction << 8;  // IN1 << 8
    polling_loop(&new_instruction) // Code aus Teil a)
    counter--;  // IN2 - 1
  }
}
```

Übungsblatt

Aufgabe 1c)

- **Adresse `a`, um im SRAM nächste Instruction abzulegen:**
  `a = __000000 XXXXXXXX XXXXXXXX` (SRAM hat nur **16 Bit** zur Adressierung)
- `final_command` ist die Instruction `01110000 00000000 00000000 00000000` mit der die Übertragung endet
- **C-Pseudo-Code:**

```
void load_code(int free_address, int final_command) {  // Adresse a
  while (new_instruction != final_command) {
    instruction_loop(&new_instruction)  // Code aus Teil b)
    SRAM[free_address] = new_instruction;  // M(<a>) := IN1
    free_address++;  // a + 1
  }
}
```

- es sind nicht mehr genug **freie Register** da, daher muss die Variable `free_address` mit der Adresse `a` auf dem **Stack** gespeichert werden

Übungsblatt
Aufgabe 2

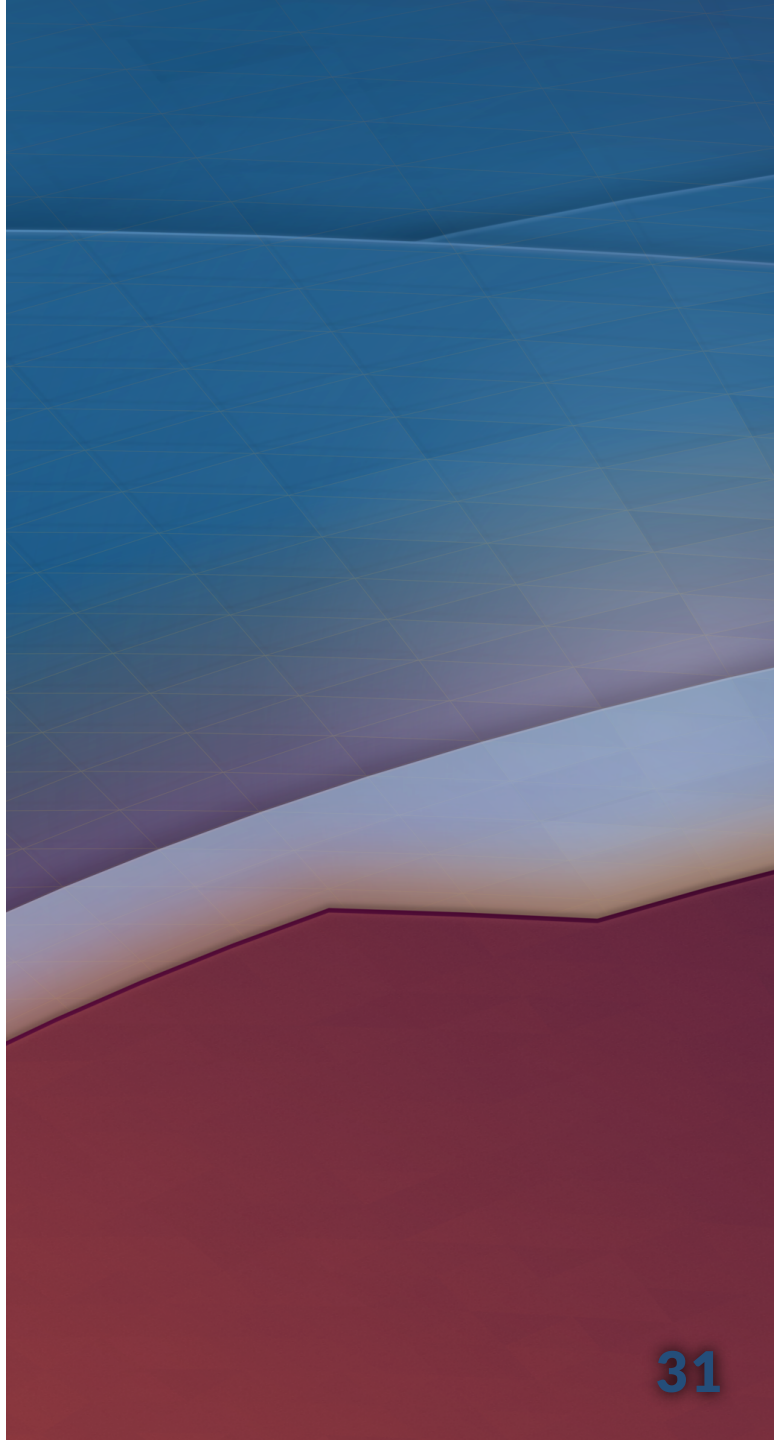
```
LOAD IN1 a // an Adresse a ist 010...0 (32 Bit) gespeichert
LOAD IN2 b // an Adresse b ist 10...0 (32 Bit) gespeichert
```

- Oder mit `MULTI` kann man einen **Bitshift** durchführen und die Bits im IN1 und IN2 Register an die passenden Stellen für den gewünschten **Präfix** shiften

```
LOADIN IN1 ACC 1 // Adresse 1 im UART ansteuern
LOADIN IN2 ACC 1 // Adresse 1 im SRAM ansteuern
```

- **EPROM** bildet eine Schlüsselstelle, weil man mit `LOADI DS 0` immer reinkommt
- funktioniert aufgrund der **Memory Map**, die nur durch Bits `31-30` auf dem **Adressbus** bestimmt wird und nicht ausschließlich vom `DS`

Ergänzungen



Ergänzungen

Packages installieren mit `apt`

updating

- `sudo apt update`: update package lists
- `sudo apt update -y && sudo apt full-upgrade`:

`full-upgrade`	* Installierte Pakete wenn möglich auf eine neuere Version aktualisieren. * Um geänderte Abhängigkeiten zu erfüllen, werden gegebenenfalls auch neue Pakete installiert. * Bei nicht mehr benötigten Abhängigkeiten werden gegebenenfalls auch Pakete entfernt.

- `sudo apt update -y && sudo apt full-upgrade qutebrowser`: update a program

“ • `full-upgrade` is the recommended way over `upgrade` ”

Ergänzungen

Packages installieren mit `apt`

installing

- `sudo apt update -y && sudo apt install gcc -y`: install package from repo
- `sudo apt update -y && sudo apt install ./foo_1.0_all.deb -y`: install local package

removing

- `sudo apt update -y && sudo apt purge gcc -y`: uninstalls package, es werden alle Konfigurationsdateien gelöscht
- `sudo apt update -y && sudo apt autoremove -y` uninstalls all packages, that are not needed anymore and have no dependencies to other packages

“ • `purge` is the recommended way over `remove` ”

Ergänzungen

Packages installieren mit `apt`

searching

- autocomplete application name, e.g. `sudo apt install openjdk`, double tab
- `apt list gcc`: lists als packages with which fit the search term
- `apt list gcc --installed`: only list packages that are installed
- `apt show gcc`: shows desciption of package matching the search term
- `apt search gcc`: lists alls packages which the search term in their discription or name

“ • glob-pattern or regex as search pattern ”

Ergänzungen

Packages installieren mit `apt`

other

- `sudo apt download emacs`: download `.deb`-package
- `sudo apt install alacritty -y`: no `y` each time
- `sudo do-release-upgrade`: upgrade **Distro** to a newer release

“
- instead of confirming with y, once can also just spam enter
- access packages over `/var/cache/apt/archives`

”

Ergänzungen

Packages installieren mit `apt`

comparisson to apt-get

Vergleich apt/apt-get

	apt install	apt-get install	apt upgrade	apt-get upgrade	apt full-upgrade	apt-get dist-upgrade
installierte Pakete wenn möglich auf eine neuere Version aktualisieren		ja		ja		ja
ggf. Installation neuer Pakete		ja	ja		nein	ja
ggf. Löschung unnötig gewordener Abhängigkeiten		nein		nein		ja
installiert ein lokales Paket und dessen Abhängigkeiten	ja	nein		–-		–-

Ergänzungen

Packages installieren mit `pacman`

Synchronising with the repositories

- `sudo pacman -Sy`: As new packages are added to the repositories you will need to regularly synchronise the package lists. This will only download the package lists if there has been a change (sudo apt update)
- `sudo pacman -Syy`: Occasionally you may want to force the package lists to be downloaded

Updating software

- `sudo pacman -Su`: perform an update of software already installed (sudo apt upgrade)
- `sudo pacman -Syu`: check whether the package lists are up-to-date at the same time

Ergänzungen

Packages installieren mit `pacman`

Searching for software

- `pacman -Ss ^hunspell`: searching a package by name in repos. Supports Regex
- `pacman -Qs hunspell`: searching package locally
- `pacman -Q`: list all packages installed on computer
- `pacman -Qeq`: self installed programs (e), only the program names, not the version number (q)
- `pacman -Qen`: packages self installed from main repos (n)
- `pacman -Qem`: packages self installed from aur (m)
- `pacman -Qdt`: orphans, unneeded dependencies

Find out where package installed

- `pacman -Ql handbrake`: look up where application gets installed

Ergänzungen

Packages installieren mit `pacman`

Installing software

- `sudo pacman -S gimagereader-gtk`: install package from repo
- `sudo pacman -U /var/cache/pacman/pkg/rofi-1.6.1-1-x86_64.pkg.tar.zst`: install local package

Removing software

- `sudo pacman -Rns dmenu`: remove a package (R), dependencies (s) and configuration files (n)
- `sudo pacman -Rns $(pacman -Qtdq)`: if at a later date you want to remove all orphan packages and configuration files for packages that you removed some time ago
- `sudo pacman -Sc`: remove unused packages and repos from cache

Ergänzungen

Packages installieren mit `pacman`

Misc

- If a package in the list is already installed on the system, it will be reinstalled even if it is already up to date. This behavior can be overridden with the `--needed` option.

Prinzip

- capital letter at beginning
- `S`: sync with repository in some way
- `Q`: search locally
- `R`: remove

Ergänzungen

Packages installieren mit `pacman`

Edit configuration files

- `sudo nvim /etc/pacman.conf`

```
# Misc options
#UseSyslog
Color
#TotalDownload
# We cannot check disk space from within a chroot environment
CheckSpace
#VerbosePkgLists
ILoveCandy
```

- `sudo nvim /etc/pacman.d/mirrorlist`

Anmerkungen

- **PAC**kage **MAN**ager
- always make `sudo pacman -Syu` before installing new software

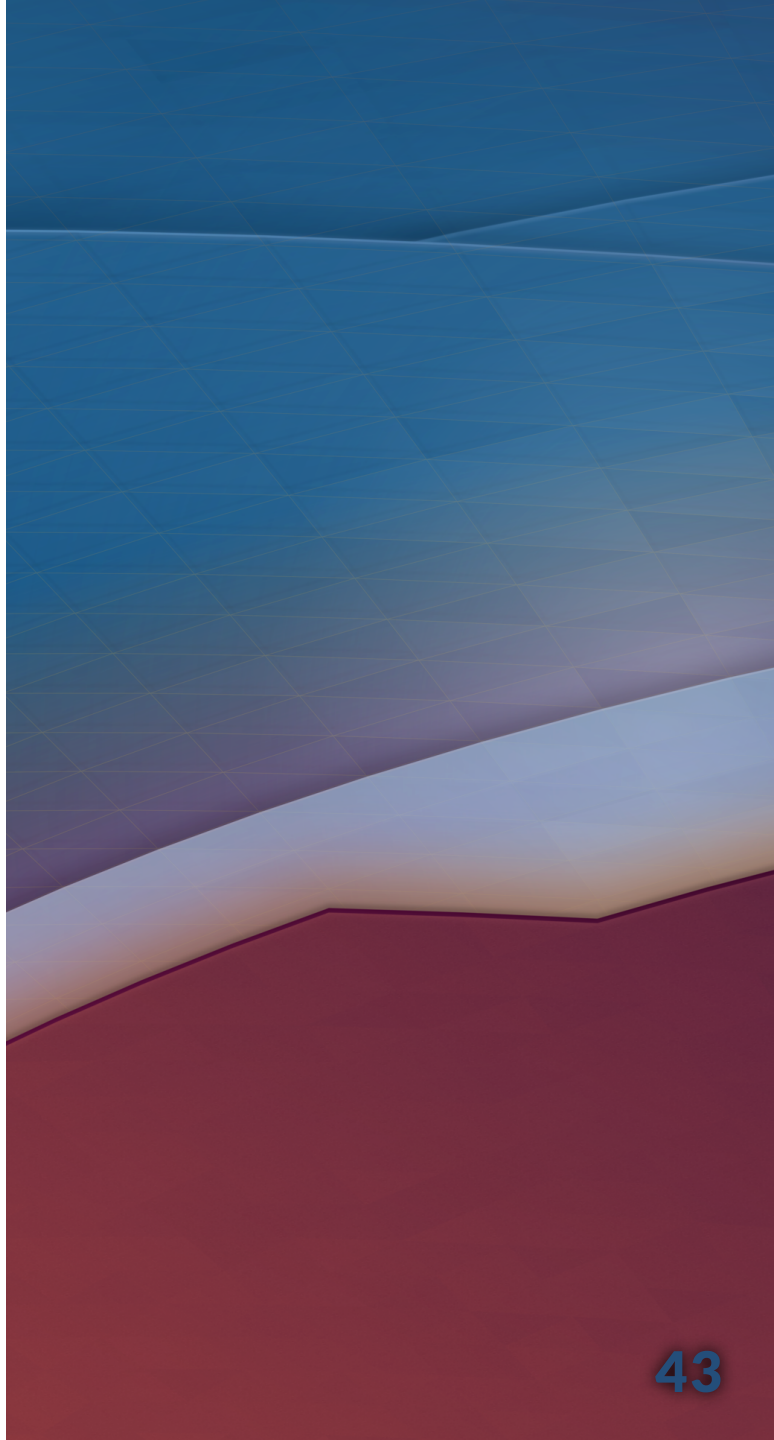
Ergänzungen

Packages installieren mit `pacman`

Yay

- commands are the same as in `pacman`
- adds search in the **AUR (Arch User Repository)**: https://aur.archlinux.org/ (**Duckduckgo:** `!au`)
- `yay polybar` erlaubt auswahl an packages, die z.B. Discord im Namen haben

Vertiefungen



Vertiefungen

Addition binär und dezimal

```
  011011 (27)          17718
+ 011101 (29)        +  6524
  11111                 11 1
  ======               ======
  111000 (56)          24242
```

```
00 + 00 = 00          00 + 00 (+ 01) = 01
00 + 01 = 01          00 + 01 (+ 01) = 10
01 + 00 = 01          01 + 00 (+ 01) = 10
01 + 01 = 10          01 + 01 (+ 01) = 11
```

Vertiefungen

Subtraktion binär und dezimal (nicht empfohlen, dient Vergleich mit nächster Folie)

```
(1)
  0111000 (56)          24242
- 0011011 (27)        - 17718
   11111                 11 1
  =======               ======
  0011101 (29)           6524
```

```
10 - 00 = 10                    10 - 00 (- 01) = 01
10 - 01 = 01                    10 - 01 (- 01) = 00
11 - 00 = 11                    11 - 00 (- 01) = 10
11 - 01 = 10                    11 - 01 (- 01) = 01
```

Vertiefungen

Subtraktion binär und dezimal (funktioniert immer, egal was für Vorzeichen Zahlen haben)

```
(2)
   0111000 (56)
 + 1100101 (27) (0011011 negiert und +1)
  11
   =======
   0011101 (29)
```

- **Zweierkomplement Negation:** `11011 -> 011011 -> 100100 -> 100101`
  - `0`en hinzufügen bis **Minuend** und **Subtrahend** beide gleiche Länge haben und Platz für ihr **Vorzeichenbit** ist und dieses korrekt gesetzt ist
  - **1er Komplement Negation** und `+1` nicht vergessen für den **Subtrahenden**

Vertiefungen

Multiplikation binär und dezimal

```
1101 x 1001 (13 * 9)          1304 x 12
       1101                          48
      0000                     +    0
     0000                      + 36
    1101                       +12
   ========                    ======
    1110101 (117)               15648
```

- **Verschiebung** ist aufgrund der `0`en, die hier ausgelassen sind

Vertiefungen
Division binär

```
  1110101 / 1011 (117 : 11) = 1010 (10) Rest: 111 (7)
- 1011|||
 =====|||
    111||
-     0||
   ====||
    1110|
-   1011|
   =====|
      111
-       0
     ====
      111
```

Vertiefungen
Division dezimal

```
15658 / 12 = 1304,833...
12|||
==|||
 36||
 36||
 ==||
  05|
   0|
  ==|
   58
   48
    ...
```

Vertiefungen
Division dezimal

```
==
10|0  oder Rest: 10
 9 6
 ===
   40
   36
   ==
    40
    36
    ==
     4...
```

Vertiefungen

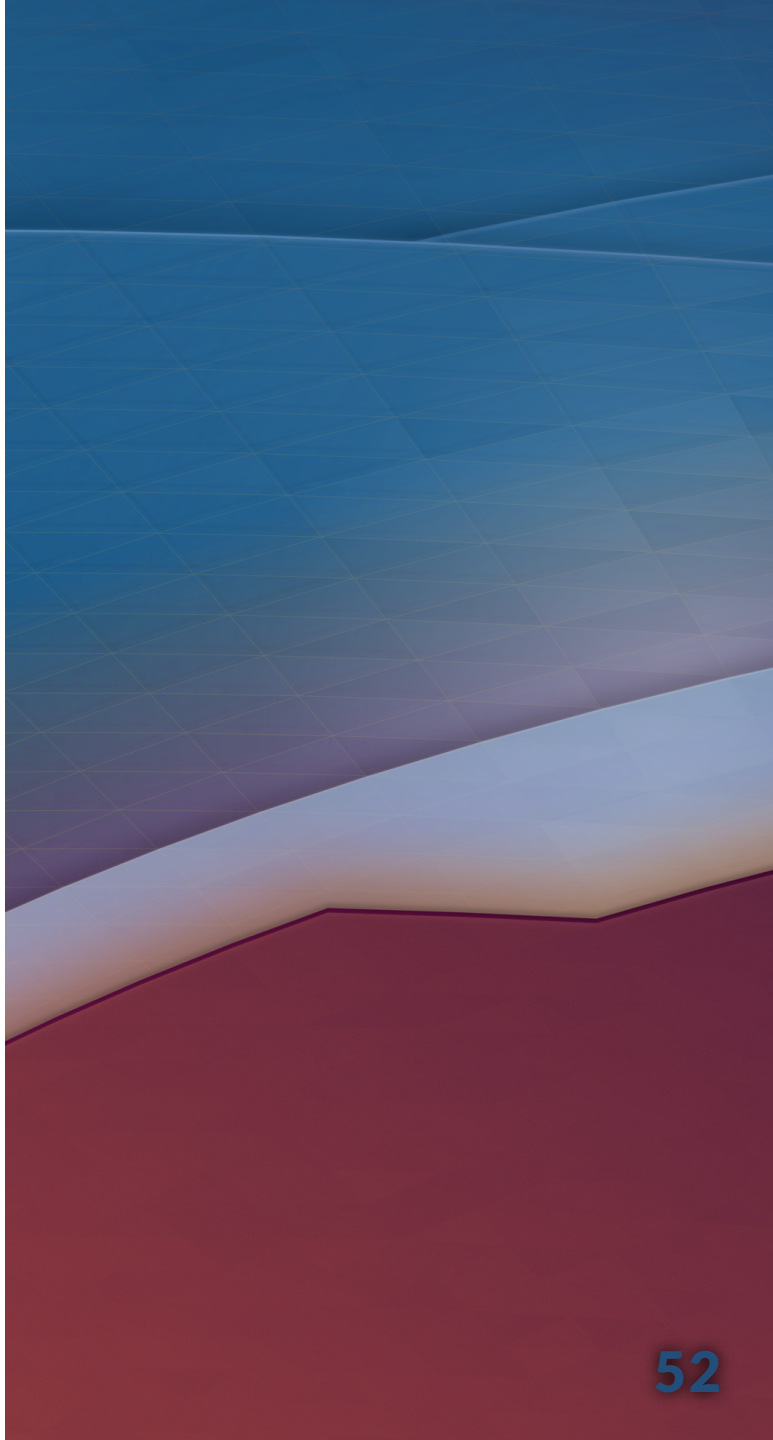
Division binär

- bei **binärer Division** gibt es nur **2 Zuständen** (`1` oder `0`), dementsprechend wird entweder die Zahl so übernommen (Zahl · `1`) oder die Zahl ist `0` (Zahl · `0`)

Division allgemein

- nach jeder Addition ein Zahl runterholen, bis keine mehr runtergeholt werden kann $\rightarrow$ dann Ende (bei **ganzzahliger Division**). Was unten stehen bleibt ist der **Rest**
- bei Division mit Nachkommastellen, 0en runterbringen, bis einmal **kein Rest** mehr rauskommt oder Grenze setzen bis zu der man weiter macht $\rightarrow$ dann Ende
- ist der **Dividend** trotz runtergebrachter weiter Stelle (weil einmal kein Rest übrig blieb) immernoch kleiner als der **Divisor**, so ist der **Quotient** 0, weil nur durch $\cdot 0$ rechnen kann der **Divisor** noch kleiner sein als der **Dividend**

Quellen



Quellen

Wissenquellen

- https://en.wikipedia.org/wiki/Register–memory_architecture

Vielen Dank für eure Aufmerksamkeit!