

OSCI Exercises

Input & Output

Winter term 2026/27

Manuel Vögele & Benedict Herzog

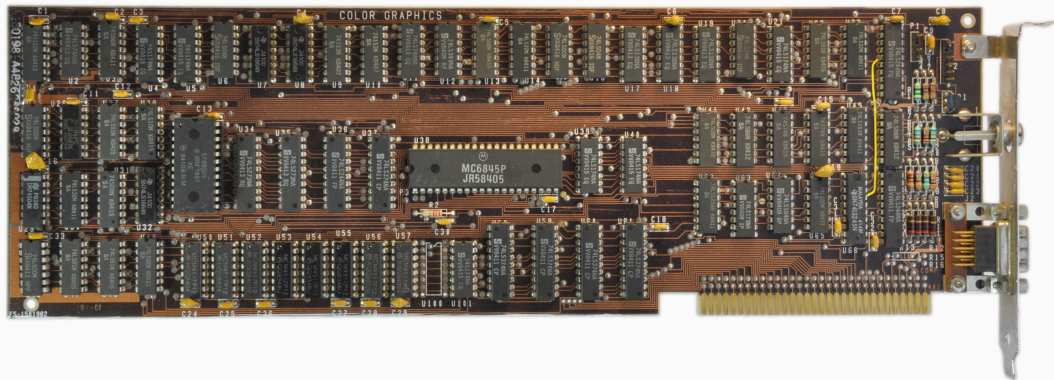
Bochum Operating Systems and System Software Group (BOSS)

RUHR
UNIVERSITÄT
BOCHUM

RUB

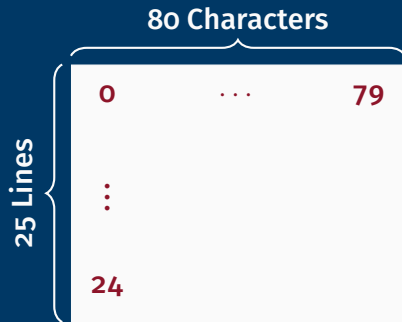


Color Graphics Adapter (CGA)



Quelle: Wikipedia

The CGA Monitor ...



... two bytes per character ...

1st byte: ASCII character

... two bytes per character ...

1st byte: **ASCII character**

[illegible]

... two bytes per character ...

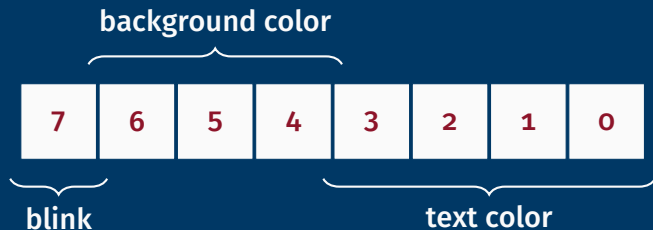
1st byte: **ASCII character**

2nd byte: **style options**

... two bytes per character ...

1st byte: ASCII character

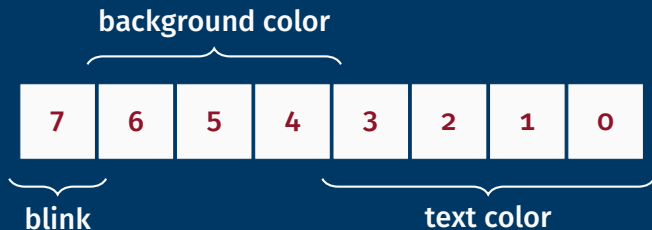
2nd byte: style options



... two bytes per character ...

1st byte: ASCII character

2nd byte: style options



0	black	4	red	8	dark gray	12	light red
1	blue	5	magenta	9	light blue	13	light magenta
2	green	6	brown	10	light green	14	yellow
3	teal	7	light gray	11	light teal	15	white

...mapped to main memory starting at 0xb8000

⋮	
'B'	← 0xb8000
0xf4	← 0xb8001
'a'	← 0xb8002
0x74	← 0xb8003
'r'	← 0xb8004
0x74	← 0xb8005
⋮	

Bar

ar

Refresher: Bitwise Operators

&	AND	0b000000011	&	0b000000101	=	0b000000001
	OR	0b000000011		0b000000101	=	0b000000111
^	XOR (exclusive OR)	0b000000011	^	0b000000101	=	0b000000110
~	negate		~	0b000000101	=	0b11111010
<<	shift left	0b00001100	<<	2	=	0b00110000
>>	shift right	0b00001100	>>	2	=	0b00000011

Refresher: Bitwise Operators

<code>&</code>	AND	<code>0b000000011</code>	<code>&</code>	<code>0b000000101</code>	<code>=</code>	<code>0b000000001</code>
<code> </code>	OR	<code>0b000000011</code>	<code> </code>	<code>0b000000101</code>	<code>=</code>	<code>0b000000111</code>
<code>^</code>	XOR (exclusive OR)	<code>0b000000011</code>	<code>^</code>	<code>0b000000101</code>	<code>=</code>	<code>0b000000110</code>
<code>~</code>	negate		<code>~</code>	<code>0b000000101</code>	<code>=</code>	<code>0b11111010</code>
<code><<</code>	shift left	<code>0b00001100</code>	<code><<</code>	<code>2</code>	<code>=</code>	<code>0b00110000</code>
<code>>></code>	shift right	<code>0b00001100</code>	<code>>></code>	<code>2</code>	<code>=</code>	<code>0b00000011</code>
<code>x = (1 << n)</code>	Set the n^{th} bit in x					
<code>x &= ~(1 << n)</code>	Clear the n^{th} bit in x					

Addressing single bits

- Bitmasks and bit operators

Addressing single bits

- Bitmasks and bit operators
- bit fields in C/C++

```
struct cga_attrb {  
    unsigned char fg : 4,  
    bg : 3,  
    bl : 1;  
};
```

```
struct cga_attrb a;  
a.fg = 4;    // red  
a.bg = 7;    // light gray  
a.bl = 1;    // blink
```

Addressing single bits

- Bitmasks and bit operators
- bit fields in C/C++

```
struct cga_attr {  
    unsigned char fg : 4,  
    bg : 3,  
    bl : 1;  
};
```

```
struct cga_attr a;  
a.fg = 4; // red  
a.bg = 7; // light gray  
a.bl = 1; // blink
```

Layout with GCC on x86:



Detour: struct packing

```
struct Test {  
    char foo[3];  
    int bar;  
    bool baz;  
};  
  
cout << sizeof(Test);
```

Detour: struct packing

```
struct Test {  
    char foo[3]; // 3 bytes  
    int bar;     // 4 bytes  
    bool baz;    // 1 byte  
};  
  
cout << sizeof(Test);
```

Detour: struct packing

```
struct Test {  
    char foo[3]; // 3 bytes  
    int bar;      // 4 bytes  
    bool baz;     // 1 byte  
};  
  
cout << sizeof(Test); // "12"
```

Detour: struct packing

```
struct Test {  
    char foo[3]; // 3 bytes  
    int bar;     // 4 bytes  
    bool baz;    // 1 byte  
} __attribute__((packed));  
  
cout << sizeof(Test); // "8"
```

Detour: struct packing

```
struct Test {  
    char foo[3]; // 3 bytes  
    int bar;      // 4 bytes  
    bool baz;     // 1 byte  
} __attribute__((packed));  
  
static_assert(sizeof(Test) == 8, "Unexpected_size");
```

Cursor

Cursor_

Either track the cursor position in software...

... or read it from the hardware

... or read it from the hardware

- CGA has 18 control registers sized 8 bits each

0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14	Cursor (high)	rw
15	Cursor (low)	rw
16	Light Pen (high)	r
17	Light Pen (low)	r

... or read it from the hardware

- CGA has 18 control registers sized 8 bits each
- Cursor position is stored in registers 14 (high) and 15 (low)

0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14	Cursor (high)	rw
15	Cursor (low)	rw
16	Light Pen (high)	r
17	Light Pen (low)	r

... or read it from the hardware

- CGA has 18 control registers sized 8 bits each
- Cursor position is stored in registers 14 (high) and 15 (low)
- Only indirect access through index (0x3d4) and data register (0x3d5)

0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14	Cursor (high)	rw
15	Cursor (low)	rw
16	Light Pen (high)	r
17	Light Pen (low)	r

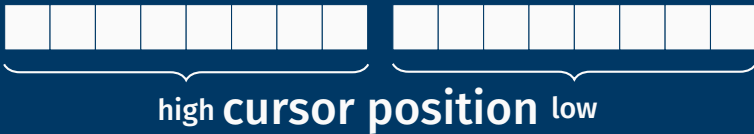
0x3d4

index register



0x3d5

data register





0x3d4

index register

15

0x3d5

data register

0x55

in



high cursor position low



0x3d4

index register

14

0x3d5

data register

0x0a

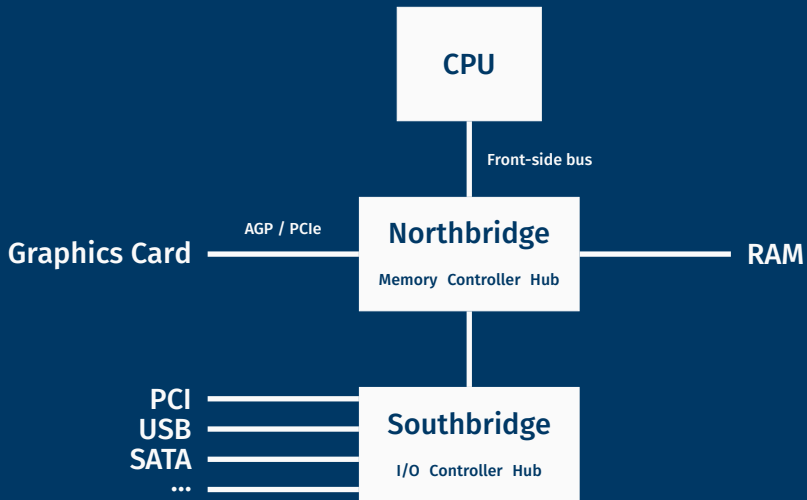
in

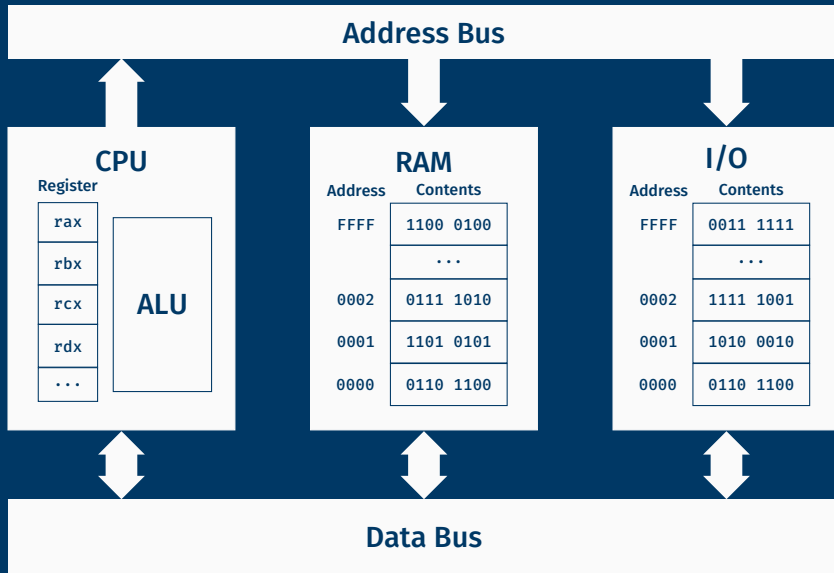


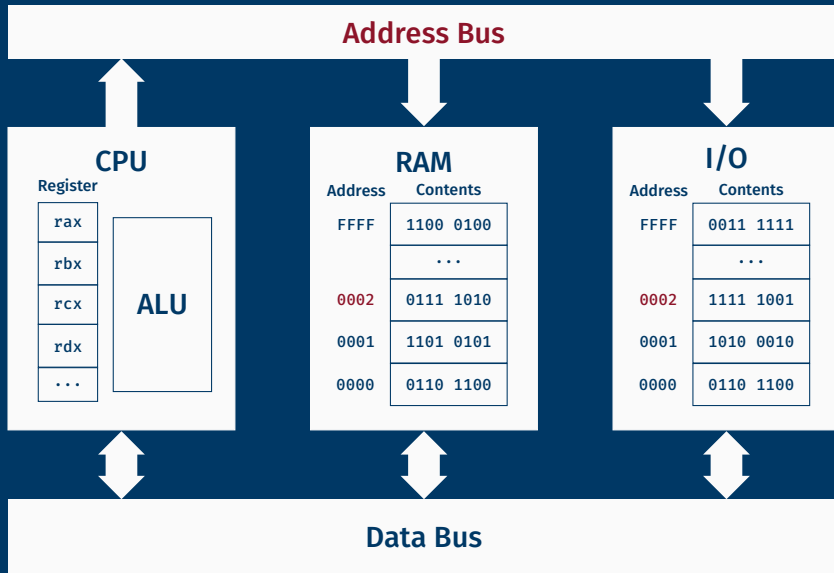
high **cursor position** low

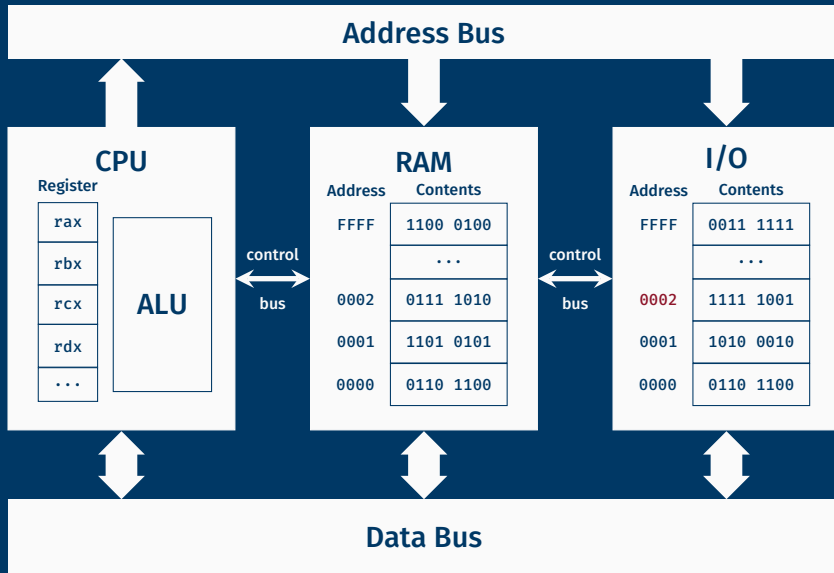
Digression:

Accessing RAM vs. I/O







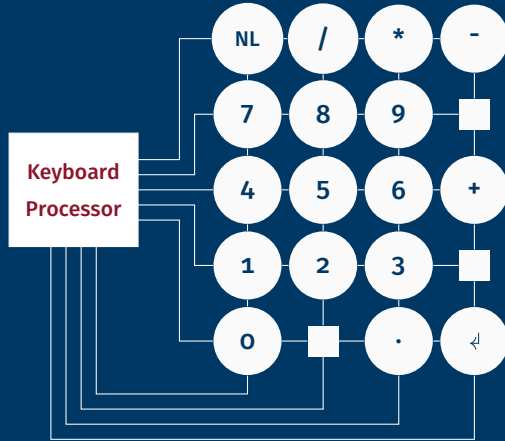


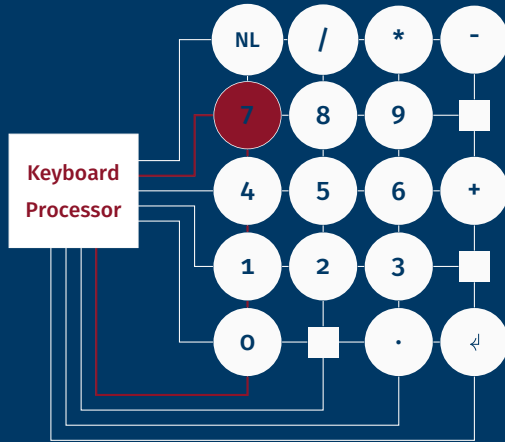
Keyboard

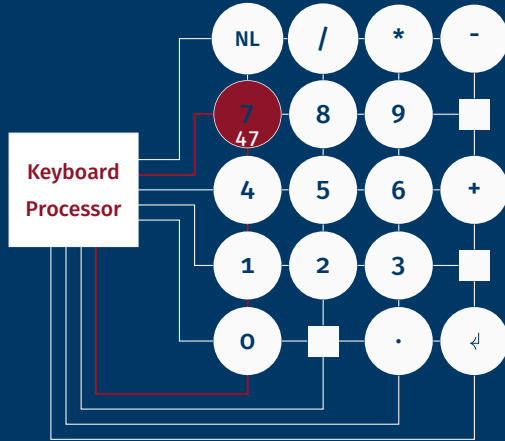


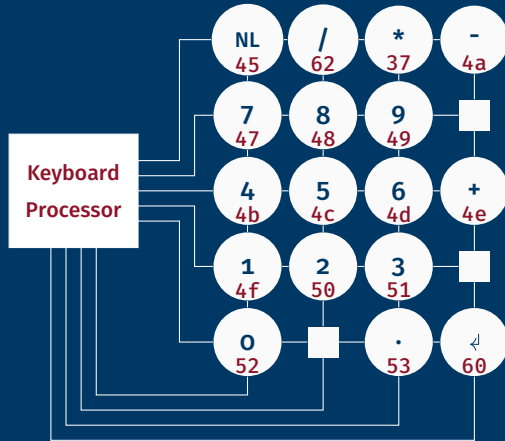
Source: Golem

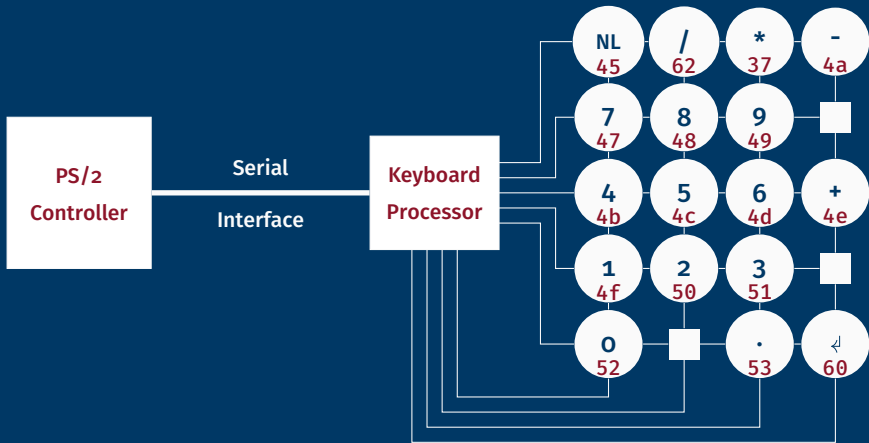


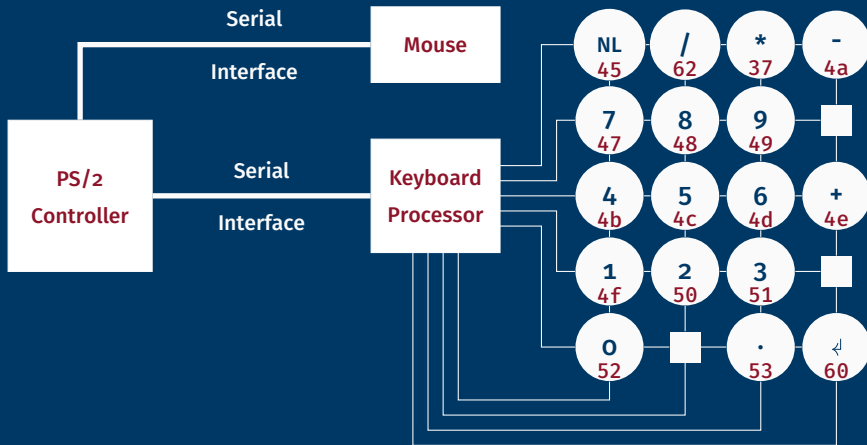












Communication with

- Keyboard (primary PS/2 device)
- Maus (secondary PS/2 device)
- PS/2 Controller (configuration)

Communication with

- Keyboard (primary PS/2 device)
- Maus (secondary PS/2 device)
- PS/2 Controller (configuration)

via I/O ports of the PS/2 controller:

0x60 data register

0x64 control register

Communication with

- Keyboard (primary PS/2 device)
- Maus (secondary PS/2 device)
- PS/2 Controller (configuration)

via I/O ports of the PS/2 controller:

0x60 data register

read output buffer (of the chosen PS/2 device)

write input buffer (of the chosen PS/2 device)

0x64 control register

Communication with

- Keyboard (primary PS/2 device)
- Maus (secondary PS/2 device)
- PS/2 Controller (configuration)

via I/O ports of the PS/2 controller:

0x60 data register

read output buffer (of the chosen PS/2 device)

write input buffer (of the chosen PS/2 device)

0x64 control register

read status register

write command register (of the PS/2 controllers)

SCODE: Key identifier, 7 bit

SCODE: Key identifier, 7 bit

MAKECODE: Key press

BREAKCODE: Key release (Scancode + 0x80)

SCODE: Key identifier, 7 bit

MAKECODE: Key press

BREAKCODE: Key release (Scancode + 0x80)

ASCII: Character rendering, 8 Bit

SCODE: Key identifier, 7 bit

MAKECODE: Key press

BREAKCODE: Key release (Scancode + 0x80)

ASCII: Character rendering, 8 Bit

→ Conversion via (software) decoder

Development environment

Test environment

- Virtual

- QEMU** Emulation in software

- KVM** hardware virtualization

Test environment

- Virtual

 - QEMU** Emulation in software

 - KVM** hardware virtualization

- Bare metal

 - two testing machines

 - PS/2 for mouse and keyboard emulated through BIOS

 - Boot through network (PXE)

 - Access option 1: directly in the lab

 - Access option 2: Through a web interface

 - <https://vnc.boss.rub.de/osci1>

 - <https://vnc.boss.rub.de/osci2>

Test environment

- Virtual

 - QEMU** Emulation in software

 - KVM** hardware virtualization

- Bare metal

 - two testing machines
 - PS/2 for mouse and keyboard emulated through BIOS
 - Boot through network (PXE)
 - Access option 1: directly in the lab
 - Access option 2: Through a web interface
 - <https://vnc.boss.rub.de/osci1>
 - <https://vnc.boss.rub.de/osci2>

The task must be functional on our test machines for task submission

Development Environment

- Recent compiler (for x64)
 - Preferably Gcc (≥ 7)
 - Alternatively LLVM/CLANG (≥ 7)
- Assembler NETWIDE ASSEMBLER (NASM)
- Buildtools (u.a. MAKE, objcopy)
- Emulator QEMU/KVM (for x86_64 guest)
- Debugger GDB
- *Optional*: Python for cpplint
- *Optional*: GNU GRUB & GNU XORRISO for ISO

Important Makefile Targets

make qemu QEMU **without** hardware virtualization

make kvm QEMU **with** hardware virtualization

Important Makefile Targets

make qemu QEMU **without** hardware virtualization

make kvm QEMU **with** hardware virtualization

make qemu-gdb start in QEMU and connect to the integrated GDB stub
→ debugging with gdb

make kvm-gdb the same, but with hardware virtualization

Important Makefile Targets

make qemu QEMU **without** hardware virtualization

make kvm QEMU **with** hardware virtualization

make qemu-gdb start in QEMU and connect to the integrated GDB stub
→ debugging with gdb

make kvm-gdb the same, but with hardware virtualization

make netboot copy into NFS for net boot on bare metal

Suffix **-dbg** for debugging compatible optimizations (-Og & Framepointer)

Suffix **-noopt** to turn off optimizations (otherwise -O3)

Suffix **-opt** for aggressive optimizations (-Ofast and LTO)

Suffix **-verbose** activates additional output (DBG_VERBOSE)

Additional Makefile Targets

make iso create a bootable ISO image

make qemu-iso boot the iso image in QEMU

make usb-*dev* create a bootable USB drive in *dev* (e.g. *sdb* - **be careful!**)
(root required)

Additional Makefile Targets

make iso create a bootable ISO image

make qemu-iso boot the iso image in QEMU

make usb-dev create a bootable USB drive in **dev** (e.g. *sdb* - **be careful!**)
(root required)

make solution-# Run the solution of task **#** with hardware virtualization

make lint check the code for code style violations

make help display available targets

Working with QEMU/KVM

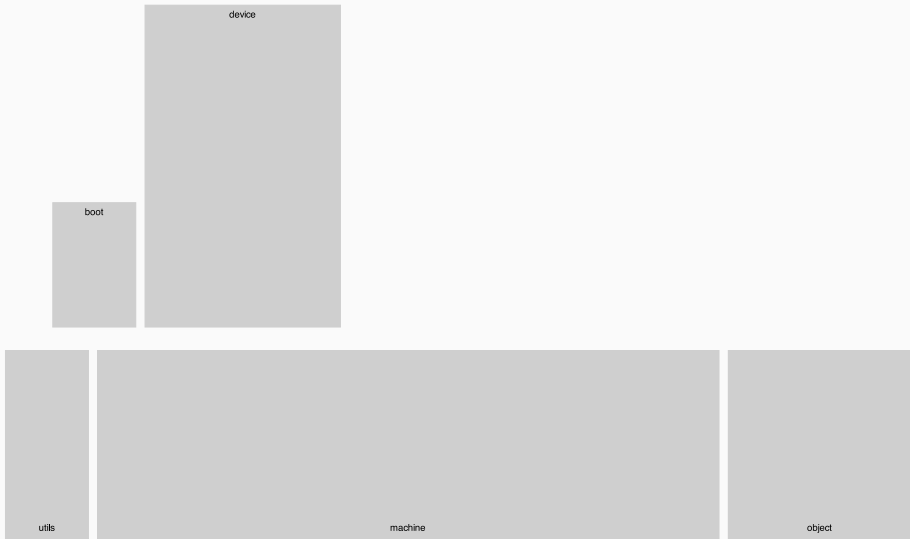
- local execution with **GTK** frontend: **left** **Ctrl** + **Alt** keys
 - **release** mouse via **Ctrl** + **Alt** + **g**
 - Toggle fullscreen via **Ctrl** + **Alt** + **f**
 - Exit via **Ctrl** + **Alt** + **q**
- lokale execution with the (old) **SDL** frontend: **right** **Ctrl** + **Alt** keys
 - **release** mouse via **Ctrl** + **Alt**
 - Toggle fullscreen via **Ctrl** + **Alt** + **f** umschalten
 - To exit, switch to monitor mode with **Ctrl** + **Alt** + **2** followed by **q**
- **Curses** frontend (e.g. when using SSH): **left** **Alt** key
 - To exit, switch to monitor mode with **Alt** + **2** followed by **q**
 - Return to output with **Alt** + **1**

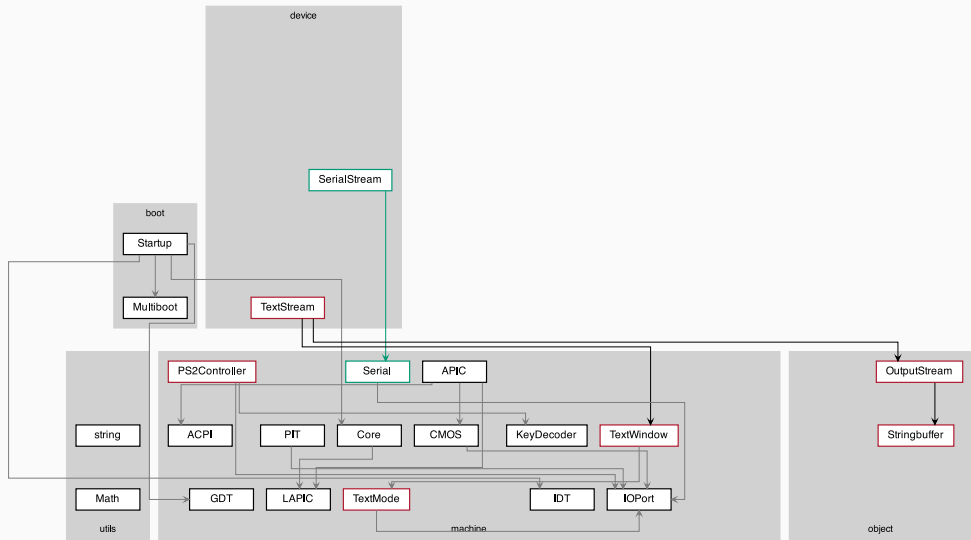
Task 1

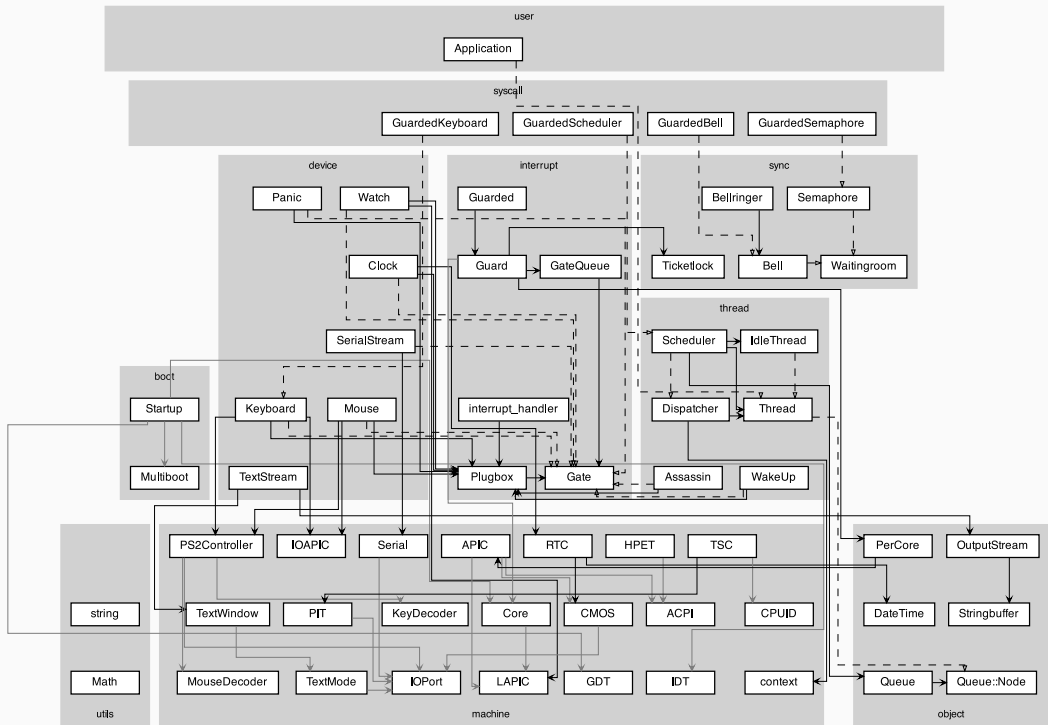
- Getting familiar with the **development environment**

- Getting familiar with the **development environment**
- Improve your **C++ skills**

- Getting familiar with the **development environment**
- Improve your **C++ skills**
- **Low-level** programming
 - Output via **CGA text mode**
 - Input via the **keyboard**
 - *Optional:* **Serial Interface**







**Submission of task 1
until Monday, 12th November**

What is important to us?

- Complete implementation of the tasks
- Well-tested implementation (including edge-cases)
- Clean source code (with comments)

What is important to us?

- Complete implementation of the tasks
- Well-tested implementation (including edge-cases)
- Clean source code (with comments)
- Working independently

What is important to us?

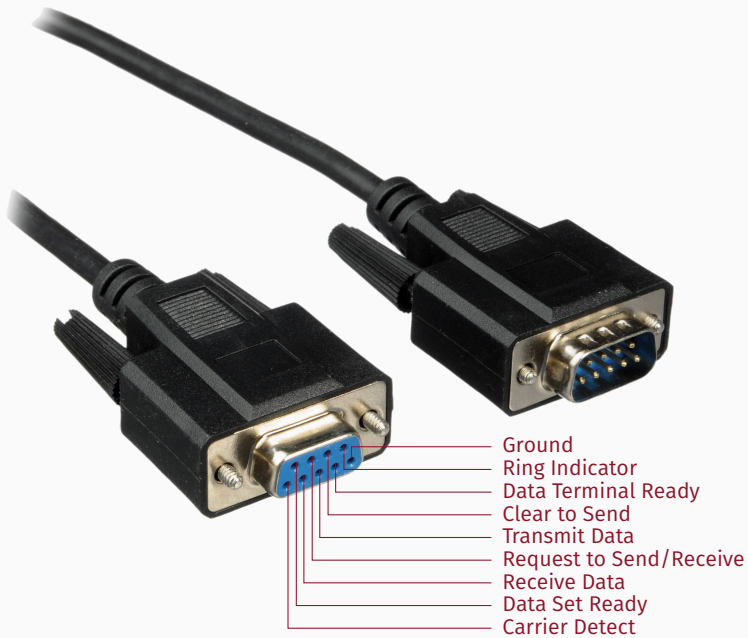
- Complete implementation of the tasks
- Well-tested implementation (including edge-cases)
- Clean source code (with comments)
- Working independently
- Timely submission

Questions?

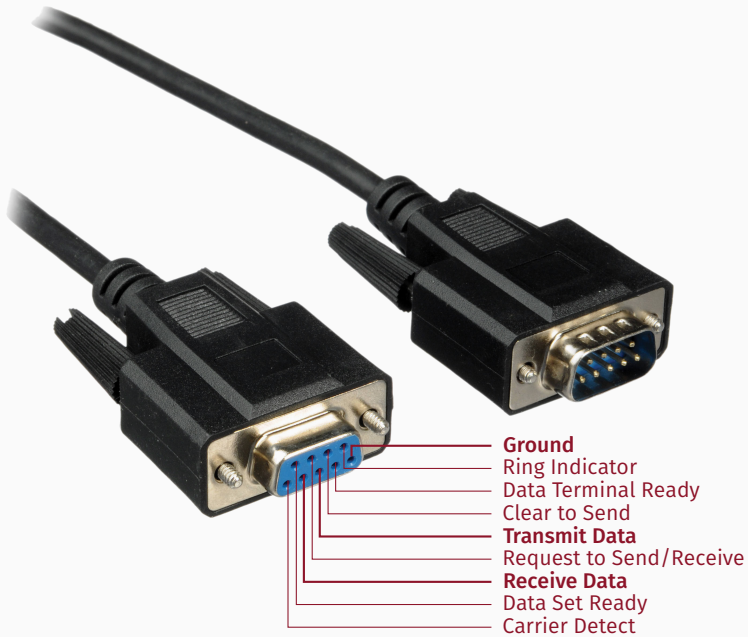
Serial Interface



Source: B&H Photo Video

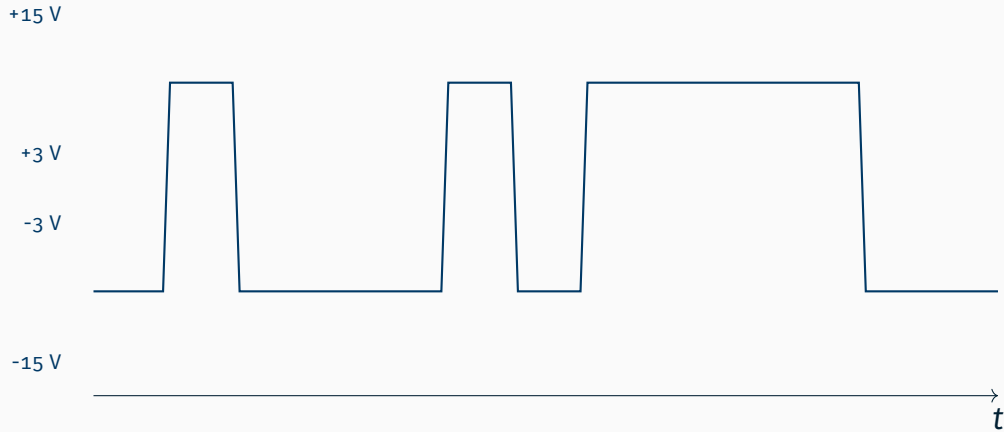


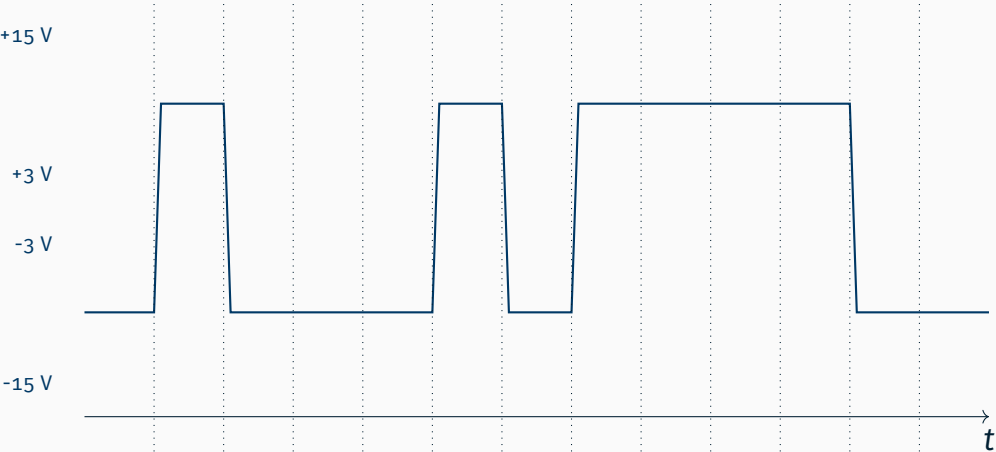
Source: B&H Photo Video



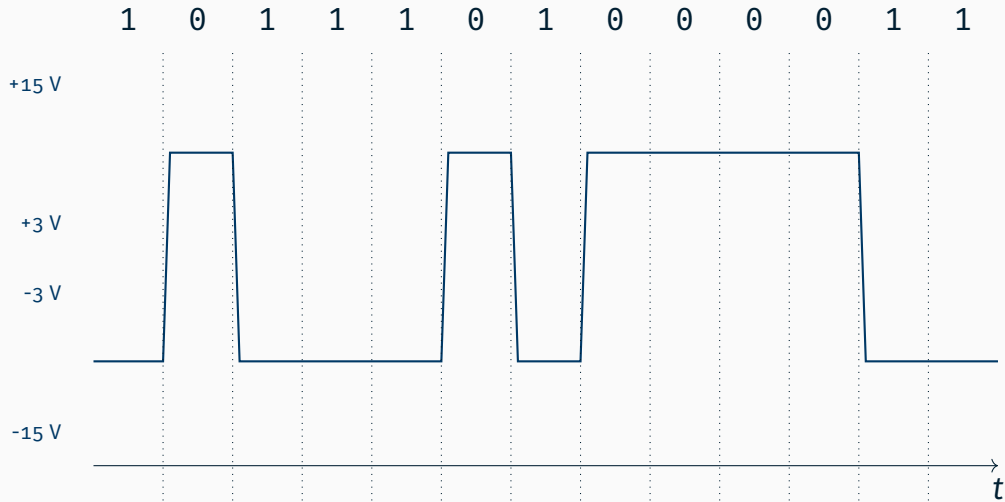
Source: B&H Photo Video

RS232

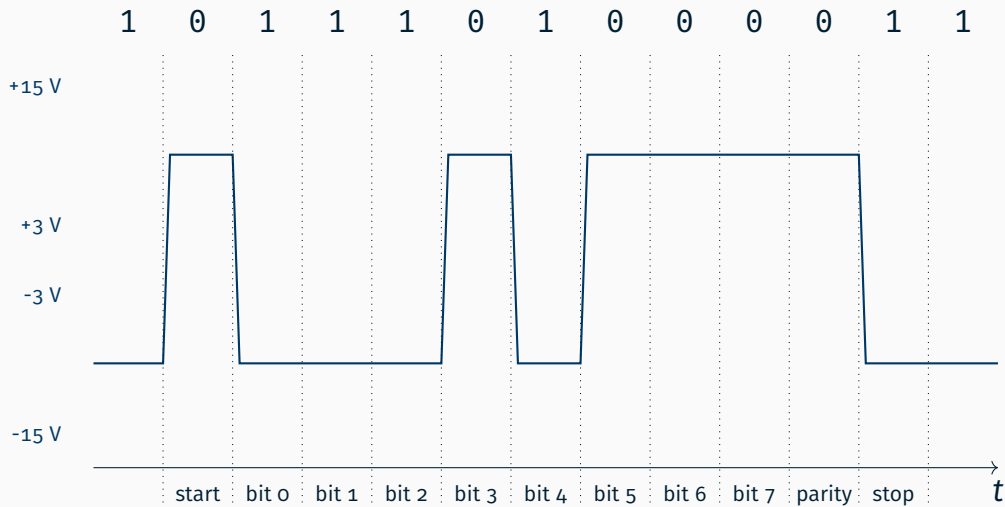




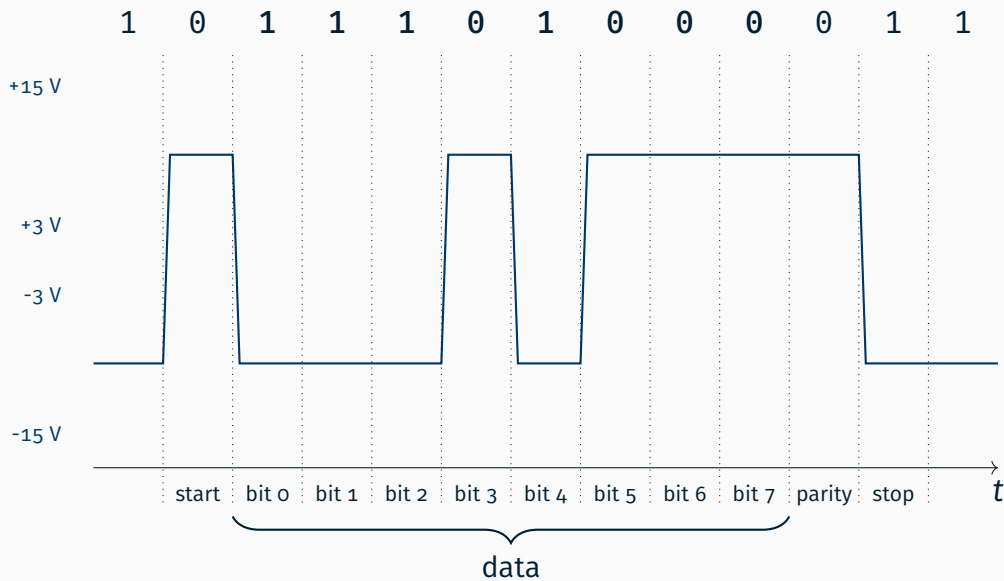
RS232



RS232



RS232



Serial Interface

- Baud rate is divisor of 115 200 Hz
- Communication 8-N-1 with 9 600 Baud is often the default
 - 8 Anzahl der Datenbits
 - N kein Paritätsbit
 - 1 Stopbit
- Current PCs usually have one hardware interface (COM1) at most
- Controller is programmed via I/O ports

Serial Interface (I/O ports)

- Common base addresses

0x3f8 COM1

0x2f8 COM2

0x3e8 COM3

0x2e8 COM4

Serial Interface (I/O ports)

- Common base addresses

 - 0x3f8 COM1

 - 0x2f8 COM2

 - 0x3e8 COM3

 - 0x2e8 COM4

- 12 registers across 8 offset addresses

 - 0 data (receive / send)

 - 1 interrupt enable / divisor low

 - 2 interrupt identification / FIFO-Control / divisor high

 - 3 line control

 - 4 modem control

 - 5 line status

 - 6 modem status

 - 7 scratch

Serial Interface (I/O ports)

- Common base addresses
 - 0x3f8 COM1
 - 0x2f8 COM2
 - 0x3e8 COM3
 - 0x2e8 COM4
- 12 registers across 8 offset addresses
 - 0 data (receive / send)
 - 1 interrupt enable / divisor low
 - 2 interrupt identification / FIFO-Control / divisor high
 - 3 line control
 - 4 modem control
 - 5 line status
 - 6 modem status
 - 7 scratch
- Details at osdev.org and lowlevel.eu

Terminal



ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '\e')

ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '\e')
- Important commands

\e[Am attributes

- 0 none
- 1 bold
- 2 faint
- 3 italic *
- 4 underline
- 5 show blink
- 6 rapid blink *
- 7 invert
- 8 hide *

* not widely supported

ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '\e')
- Important commands

\e[Am attributes

\e[3Cm Foreground color

\e[4Cm Background color

0 black

1 red

2 green

3 yellow

4 blue

5 magenta

6 cyan

7 white

9 *default color*

ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '`\e`')
- Important commands
 - `\e[A`m attributes
 - `\e[3C`m Foreground color
 - `\e[4C`m Background color
 - `\e[Y;X`H Set cursor position

ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '\e')
- Important commands
 - \e[A~~m~~ attributes
 - \e[3~~C~~m Foreground color
 - \e[4~~C~~m Background color
 - \e[~~Y~~;~~X~~H Set cursor position
 - \e[6n Read cursor Position
 - \e[~~Y~~;~~X~~R Response (cursor position)

ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '`\e`')
- Important commands
 - `\e[A`m attributes
 - `\e[3`Cm Foreground color
 - `\e[4`Cm Background color
 - `\e[Y;X`H Set cursor position
 - `\e[6`n Read cursor Position
 - `\e[Y;X`R Response (cursor position)
 - `\ec` Reset

ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '`\e`')
- Important commands
 - `\e[A`m attributes
 - `\e[3C`m Foreground color
 - `\e[4C`m Background color
 - `\e[Y;X`H Set cursor position
 - `\e[6n` Read cursor Position
 - `\e[Y;XR` Response (cursor position)
 - `\ec` Reset
- You can test this on the command line

```
o1 voegele:~$ echo -e "\ec\e[47m\e[1mF\e[31moo\e[0m"
```

ANSI Escape Sequences

- Control codes for the cursor position and text attributes
- Start with ESCAPE character (27. ASCII character, '`\e`')
- Important commands

`\e[A`m attributes

`\e[3`**C**m Foreground color

`\e[4`**C**m Background color

`\e[Y;X`H Set cursor position

`\e[6`n Read cursor Position

`\e[Y;X`R Response (cursor position)

`\e`c Reset

- You can test this on the command line

```
o1 voegele:~$ echo -e "\ec\e[47m\e[1mF\e[31moo\e[0m"
```

See also <http://www.termsys.demon.co.uk/vtansi.htm>

https://en.wikipedia.org/wiki/ANSI_escape_code