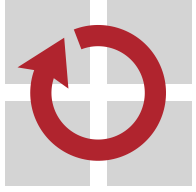
Übung zu Betriebssysteme

Ein- und Ausgabe

25. & 27. Oktober 2017

Andreas Ziegler
Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

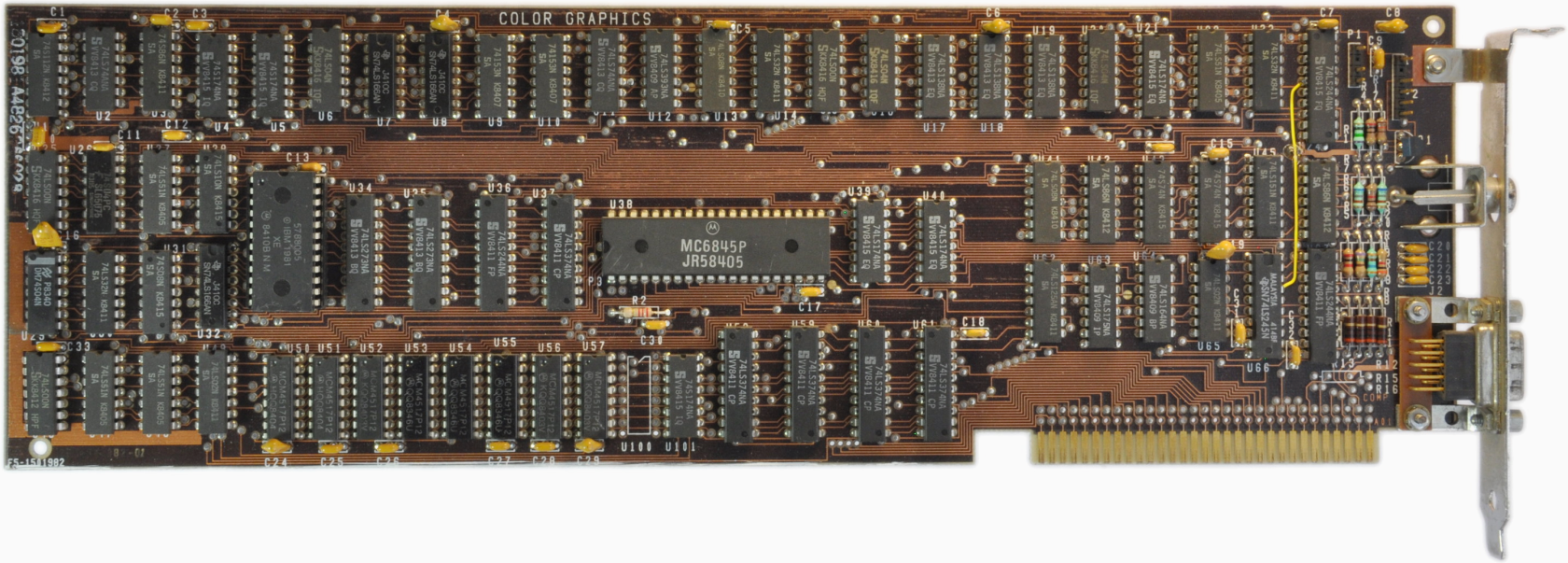


Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

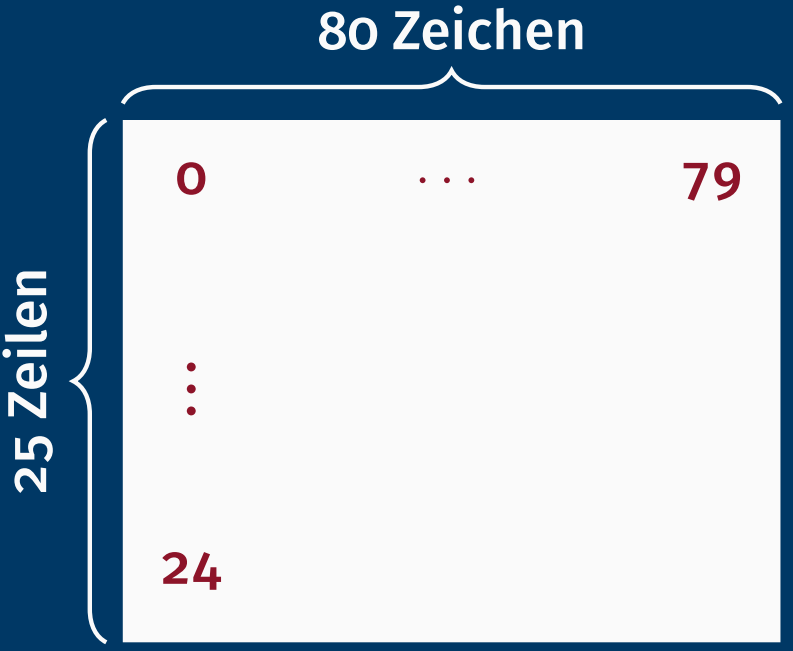
TECHNISCHE FAKULTÄT

Color Graphics Adapter



Quelle: Wikipedia

Der CGA Bildschirm ...



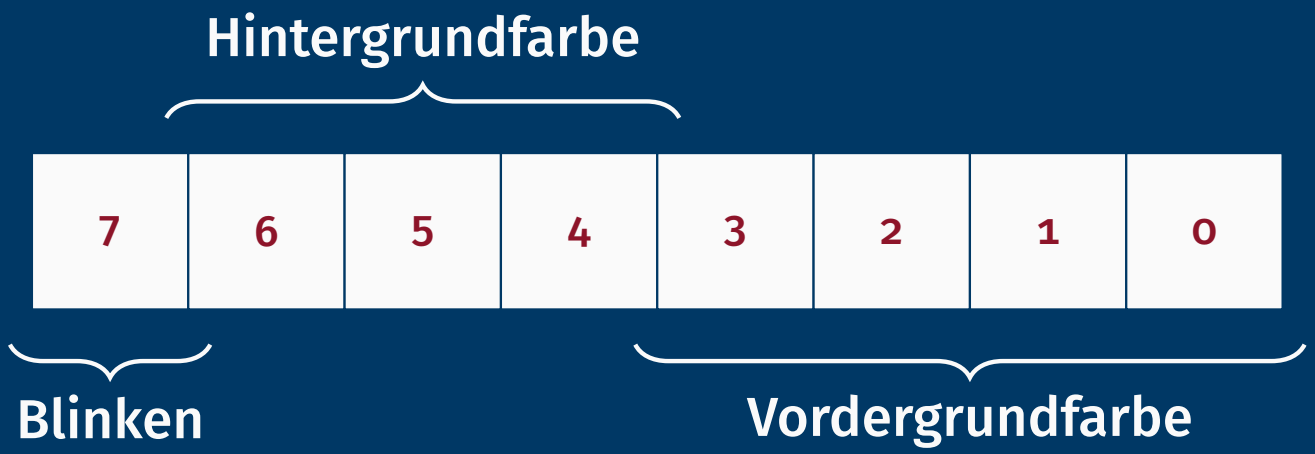
… zwei Bytes pro Zeichen …

1. Byte: ASCII Zeichen
2. Byte: Darstellungsattribut

… zwei Bytes pro Zeichen …

1. Byte: ASCII Zeichen

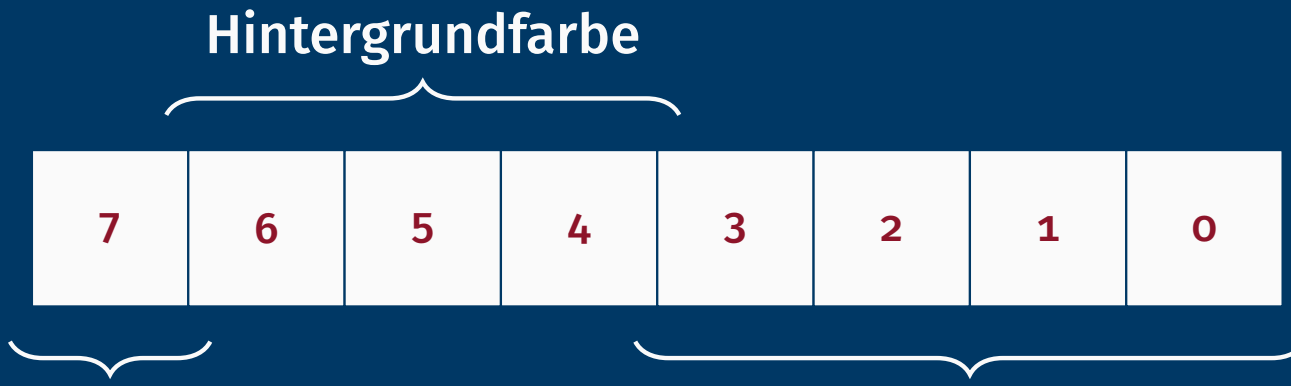
2. Byte: Darstellungsattribut



… zwei Bytes pro Zeichen …

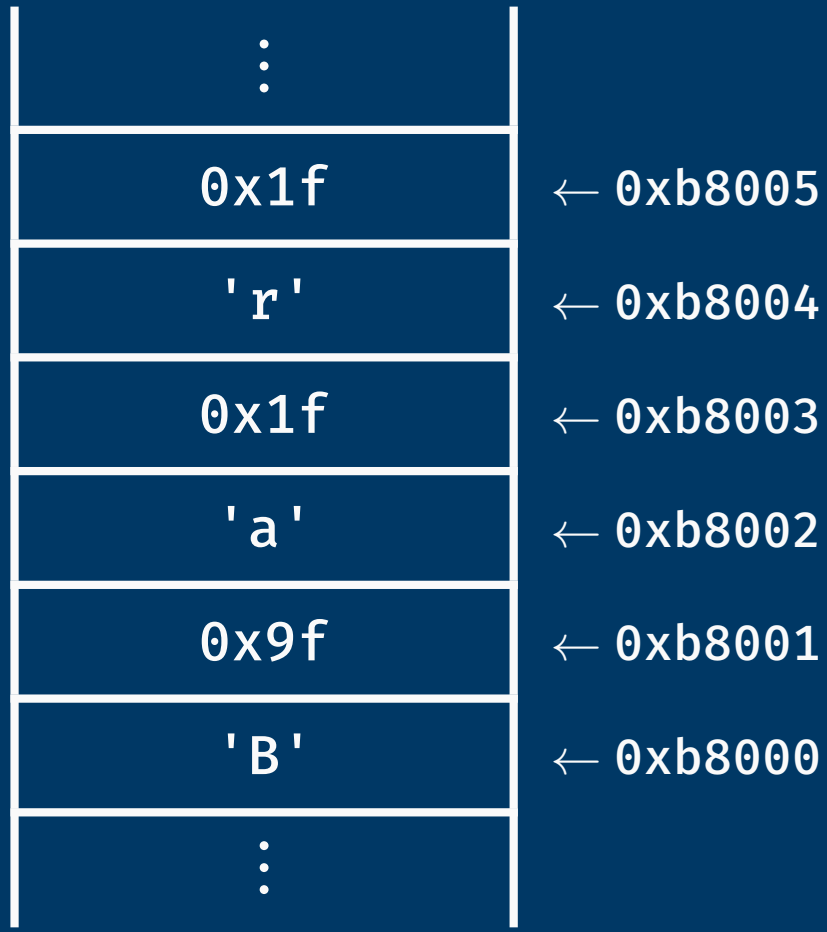
1. Byte: ASCII Zeichen

2. Byte: Darstellungsattribut



0	schwarz	4	rot	8	dunkelgrau	12	hellrot
1	blau	5	magenta	9	hellblau	13	hellmagenta
2	grün	6	braun	10	hellgrün	14	gelb
3	cyan	7	hellgrau	11	hellcyan	15	weiß

…eingeblendet im Arbeitsspeicher ab `0xb8000`



Rekapitulation: Bit~~schubsen~~operationen

`&`	Konjunktion (logisches UND)
`|`	Disjunktion (logisches ODER)
`^`	exklusives ODER
`~`	Einerkomplement (logische Negation)
`<<`	verschieben nach links (Multiplikation mit 2)
`>>`	verschieben nach rechts (Division durch 2)

… entweder
Cursorposition in Software merken …

... oder

Cursorposition aus Hardware auslesen

… oder
Cursorposition aus Hardware auslesen

CGA hat 18 Steuerregister mit je 8 Bit

… oder
Cursorposition aus Hardware auslesen

CGA hat 18 Steuerregister mit je 8 Bit

Position in Register 14 (high) und 15 (low)

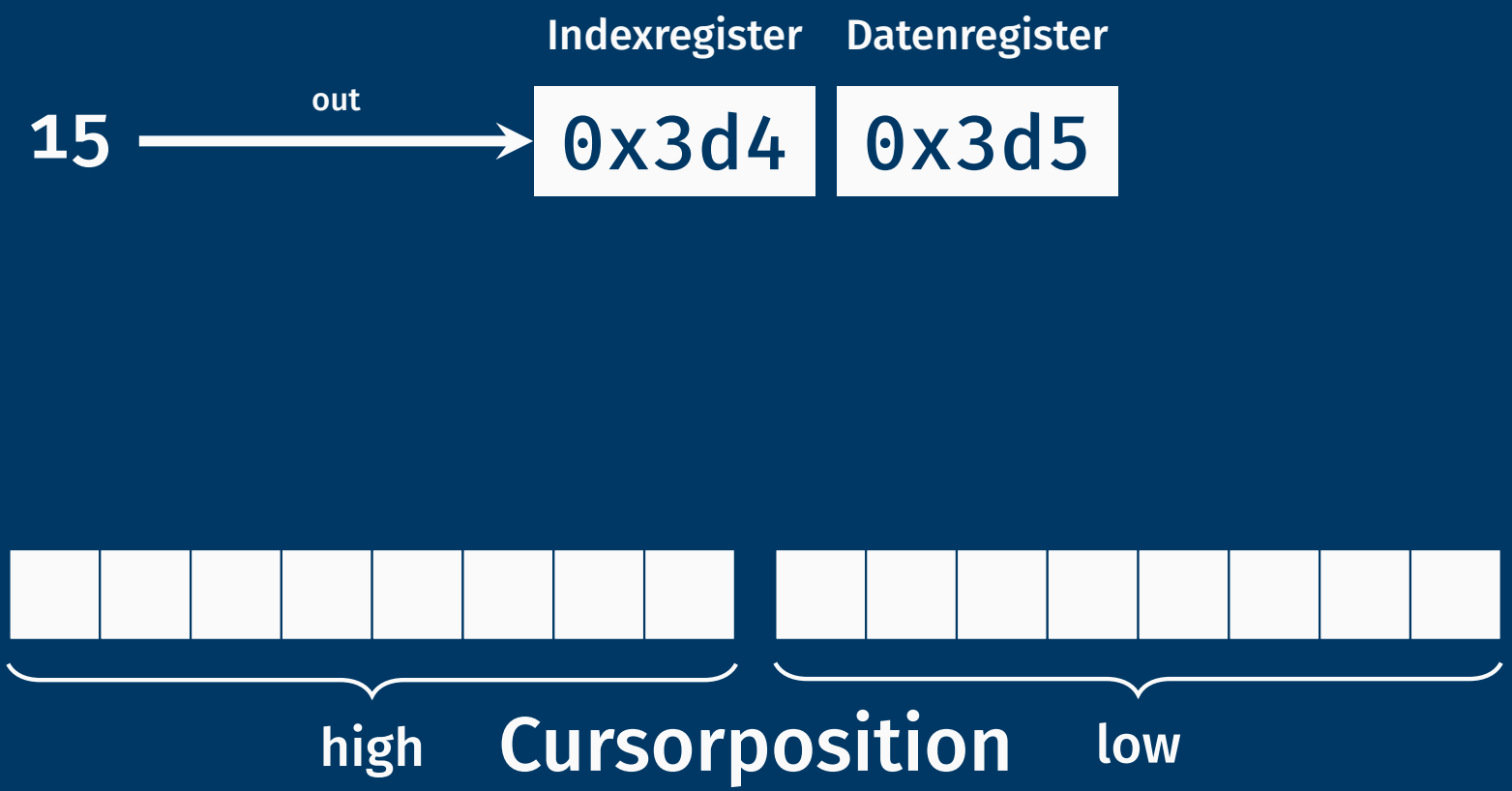
… oder
Cursorposition aus Hardware auslesen

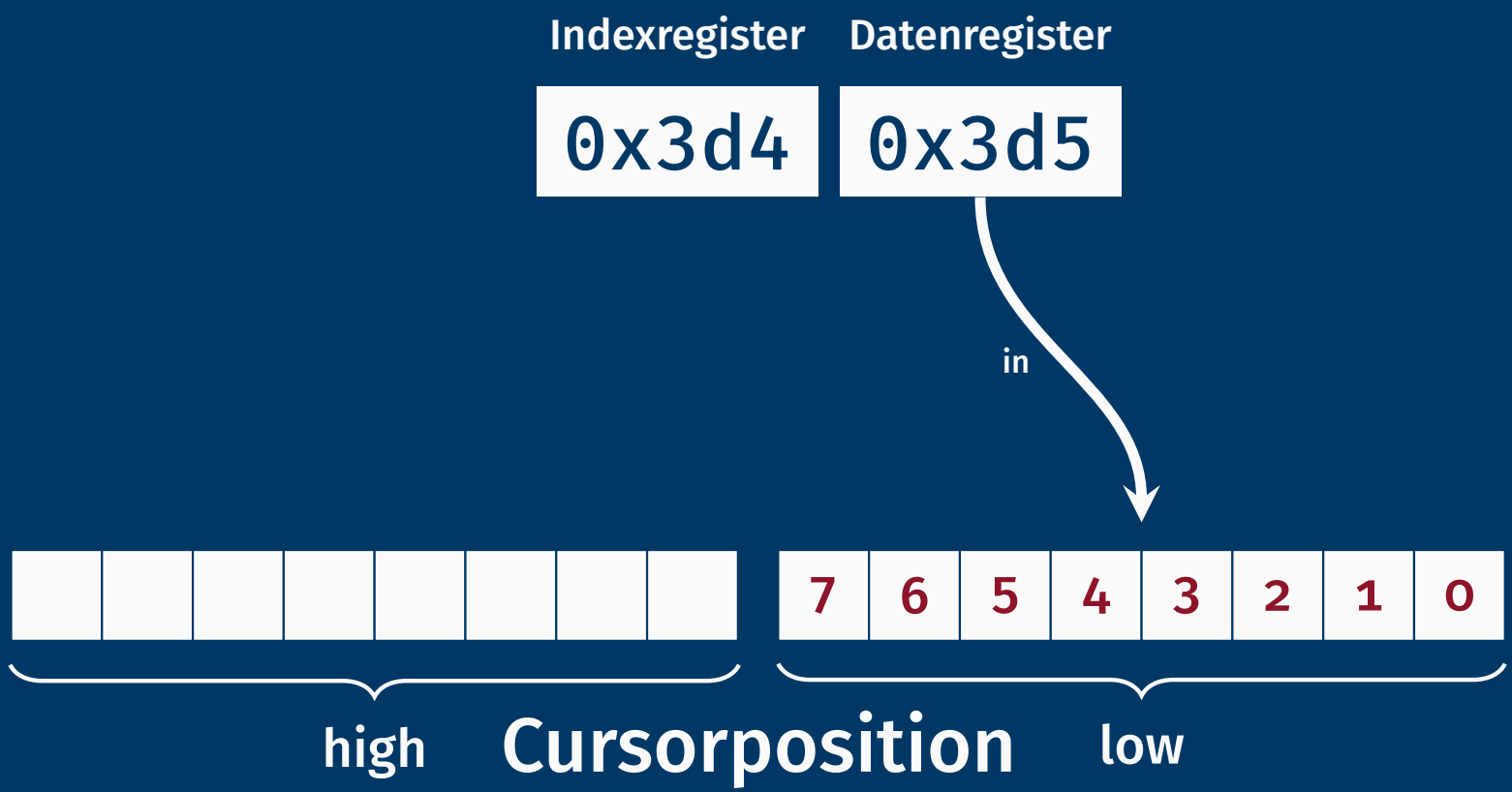
CGA hat 18 Steuerregister mit je 8 Bit

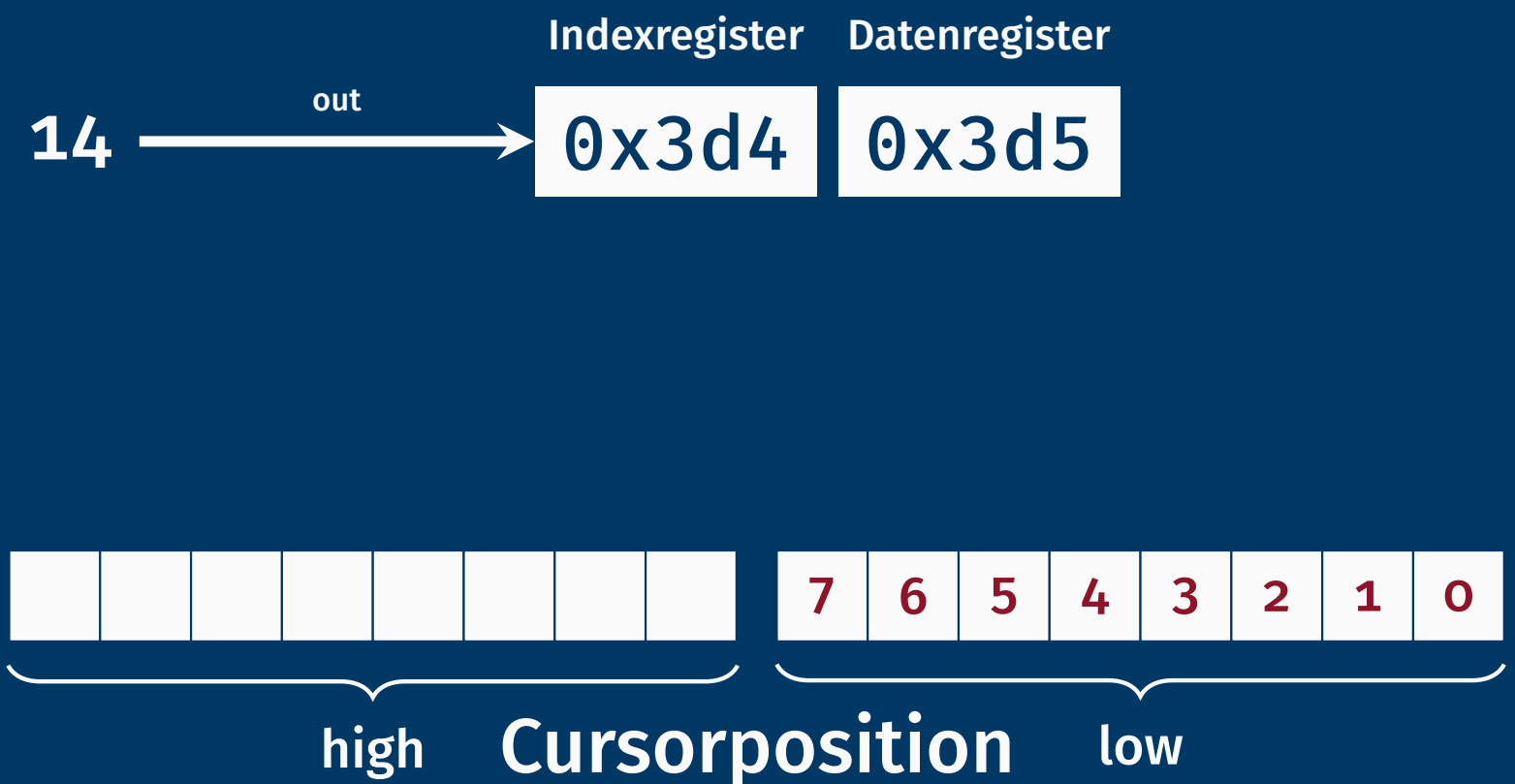
Position in Register 14 (high) und 15 (low)

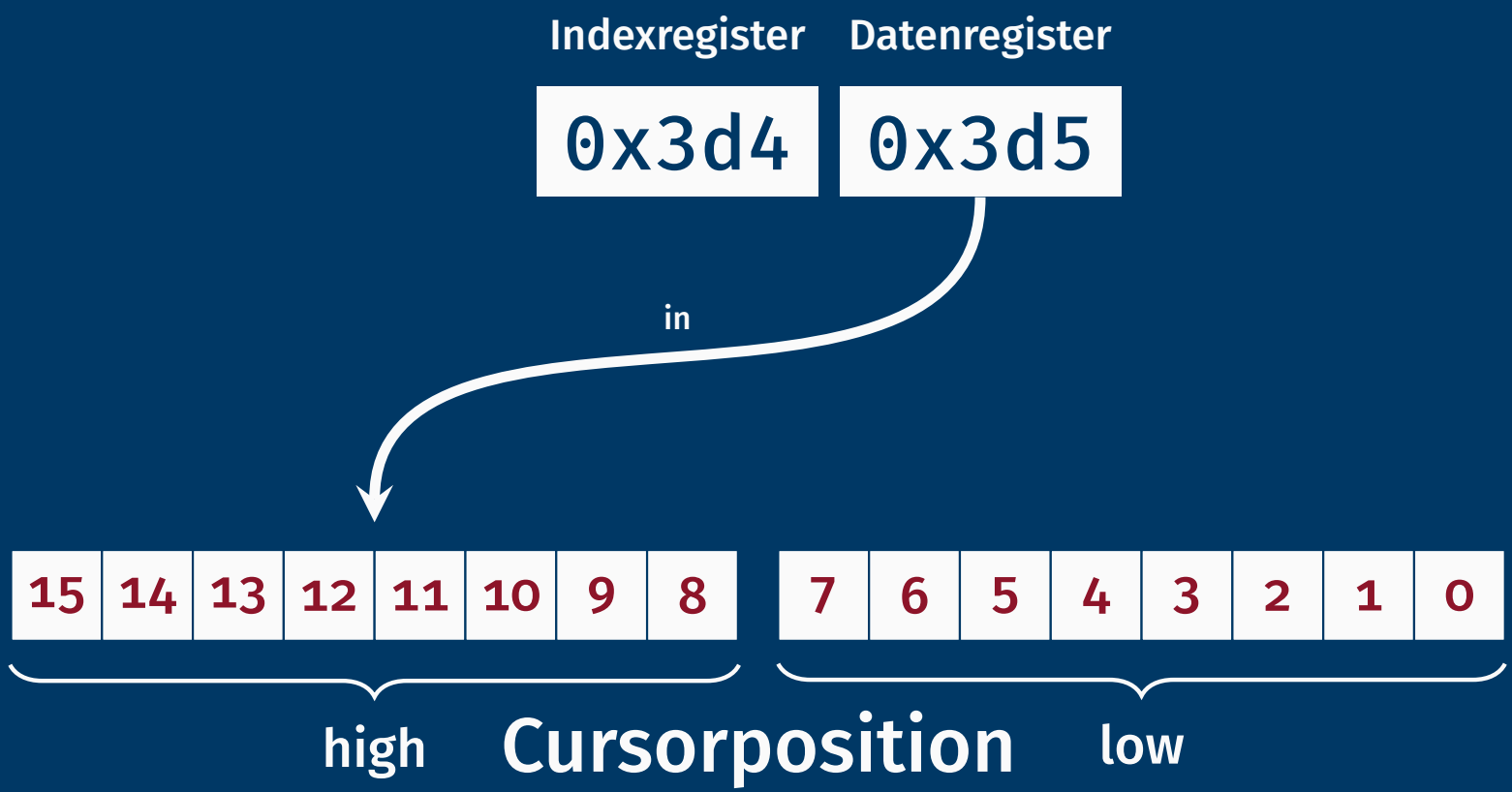
Nur indirekter Zugriff über Index- (`0x3d4`)
und Datenregister (`0x3d5`)

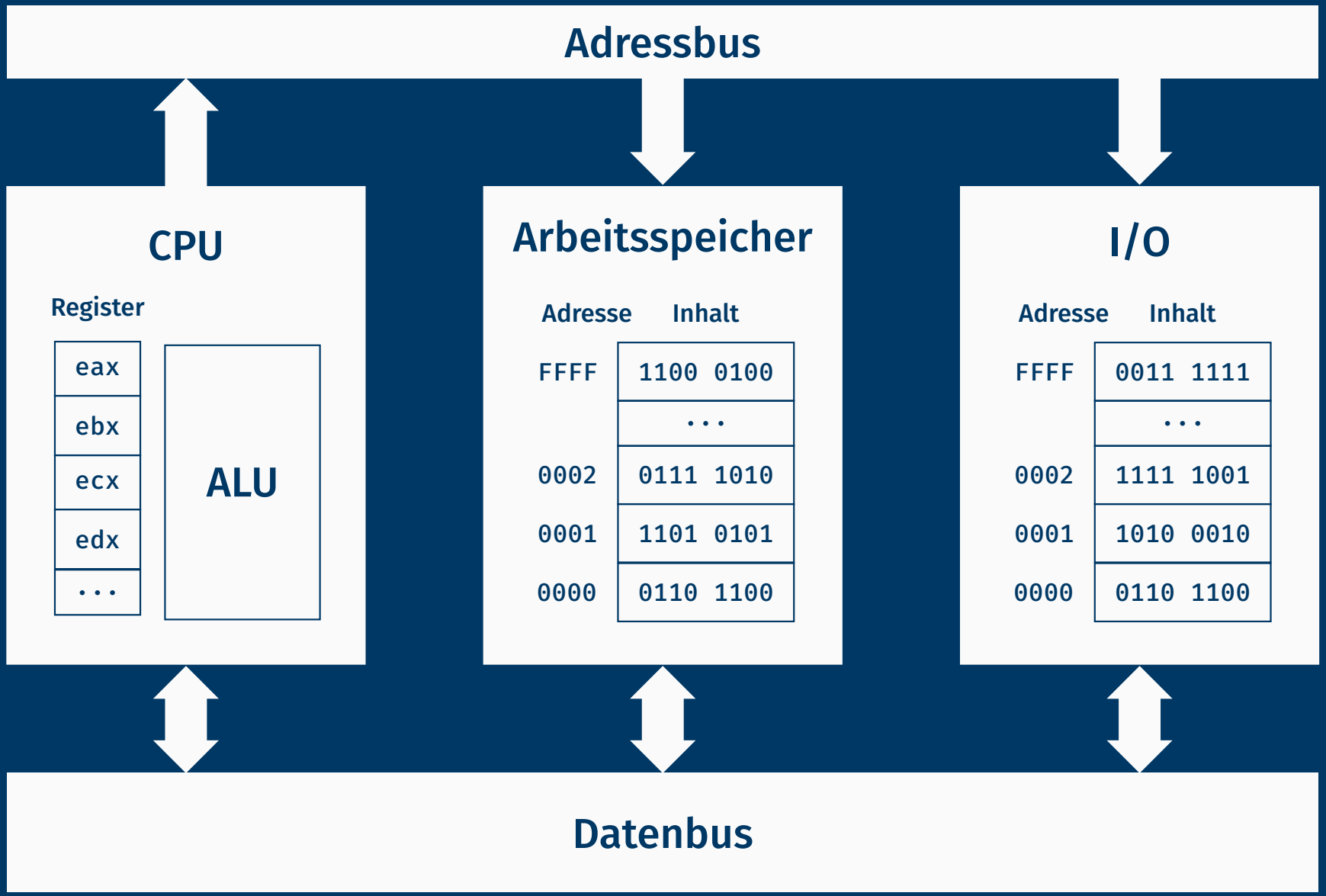
Indexregister	Datenregister
0x3d4	0x3d5

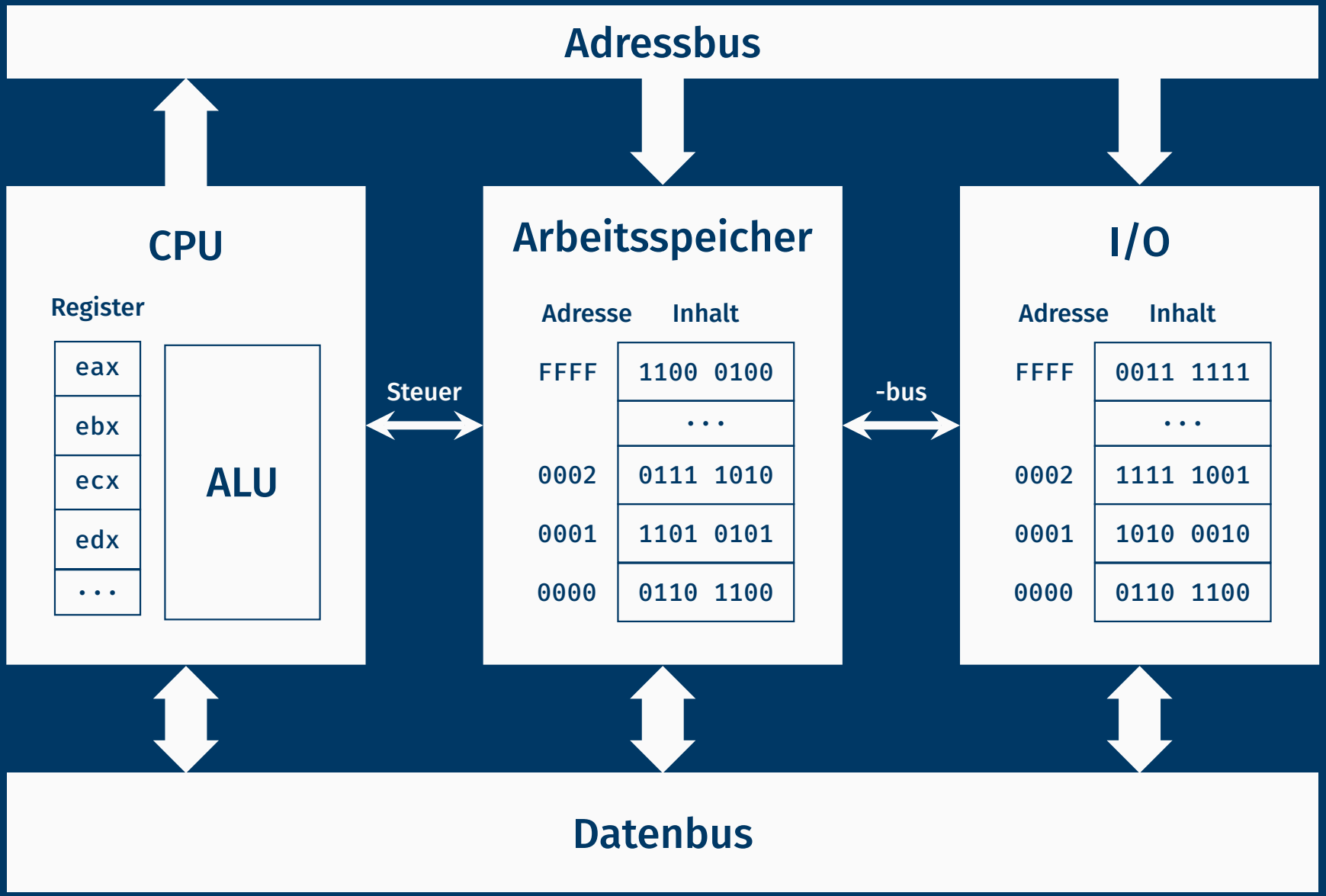

Indexregister
Datenregister
15
out
0x3d4
0x3d5
high
Cursorposition
low


Indexregister
Datenregister
0x3d4
0x3d5
in
7
6
5
4
3
2
1
0
high
Cursorposition
low


Indexregister
Datenregister
14
out
0x3d4
0x3d5
7 6 5 4 3 2 1 0
high
Cursorposition
low


Indexregister
Datenregister
0x3d4
0x3d5
in
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
high
Cursorposition
low

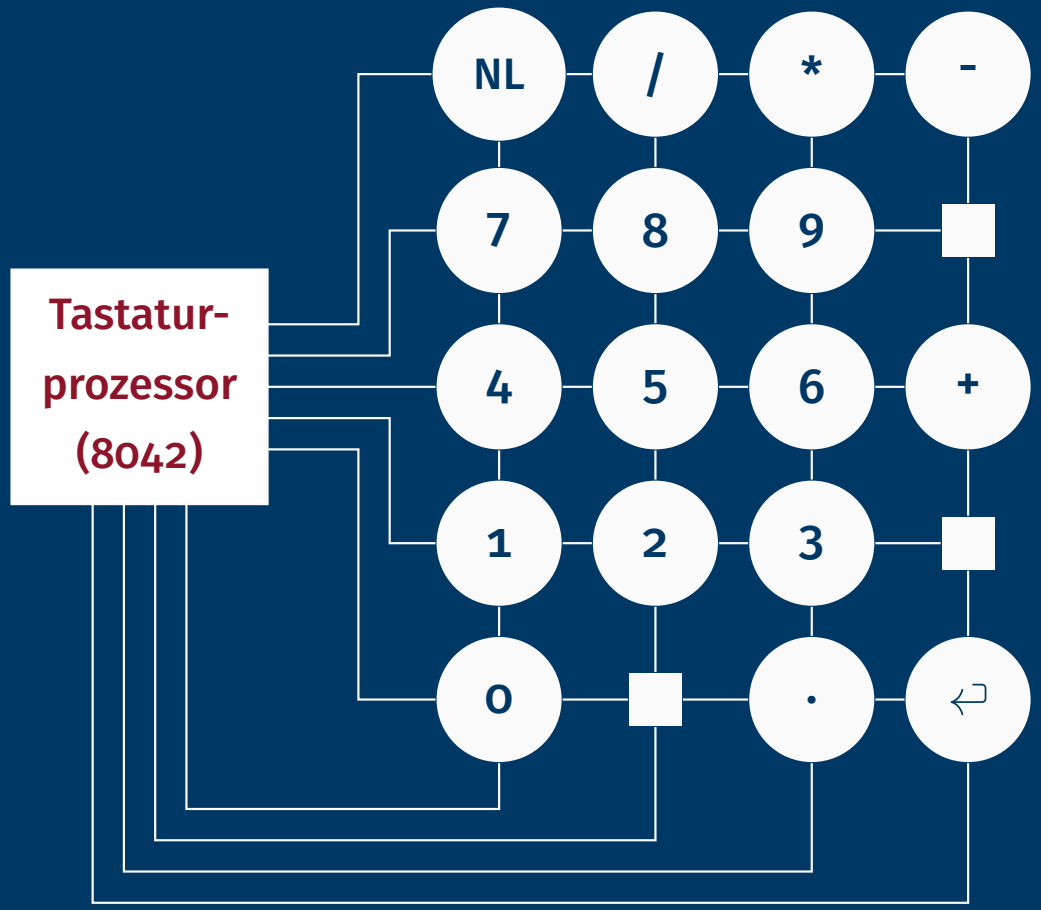

Adressbus
CPU
Register
eax
ebx
ecx
edx
...
ALU
Arbeitsspeicher
Adresse
Inhalt
FFFF
1100 0100
...
0002
0111 1010
0001
1101 0101
0000
0110 1100
I/O
Adresse
Inhalt
FFFF
0011 1111
...
0002
1111 1001
0001
1010 0010
0000
0110 1100
Datenbus

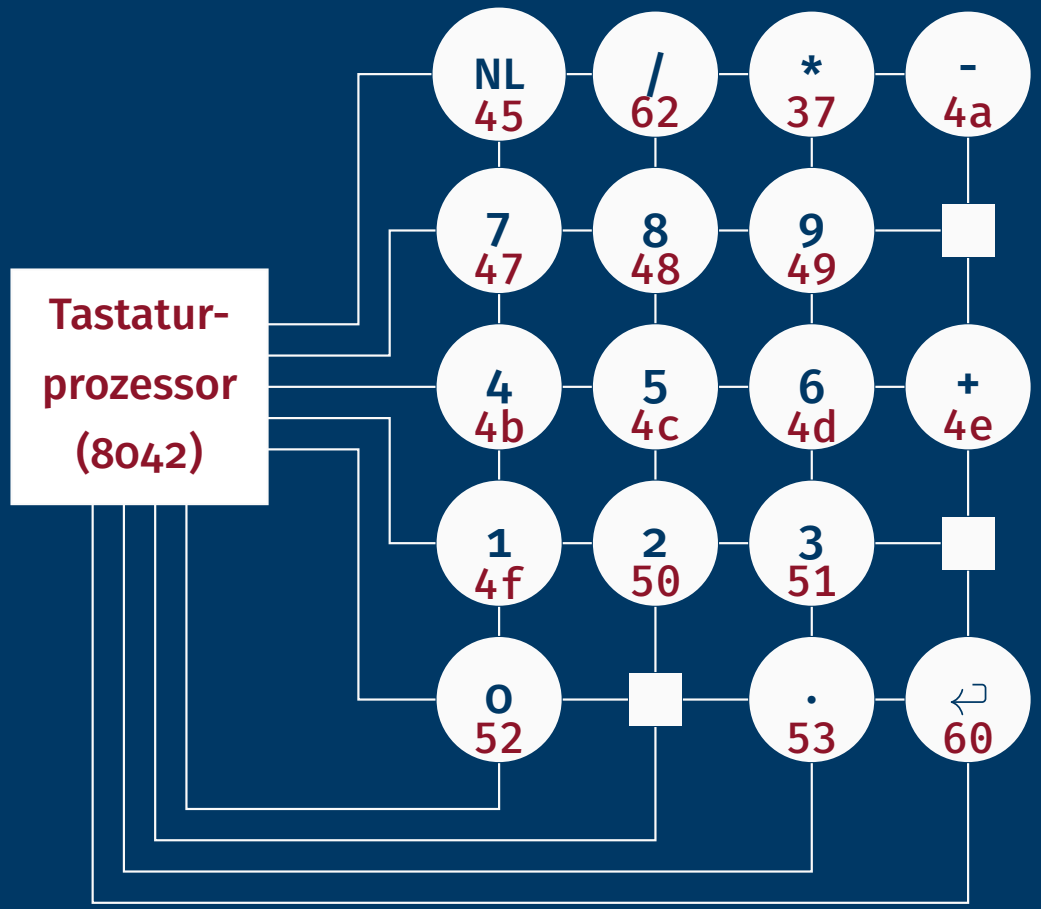

Adressbus
CPU
Register
eax
ebx
ecx
edx
...
ALU
Steuer
Arbeitsspeicher
Adresse
Inhalt
FFFF
1100 0100
...
0002
0111 1010
0001
1101 0101
0000
0110 1100
-bus
I/O
Adresse
Inhalt
FFFF
0011 1111
...
0002
1111 1001
0001
1010 0010
0000
0110 1100
Datenbus

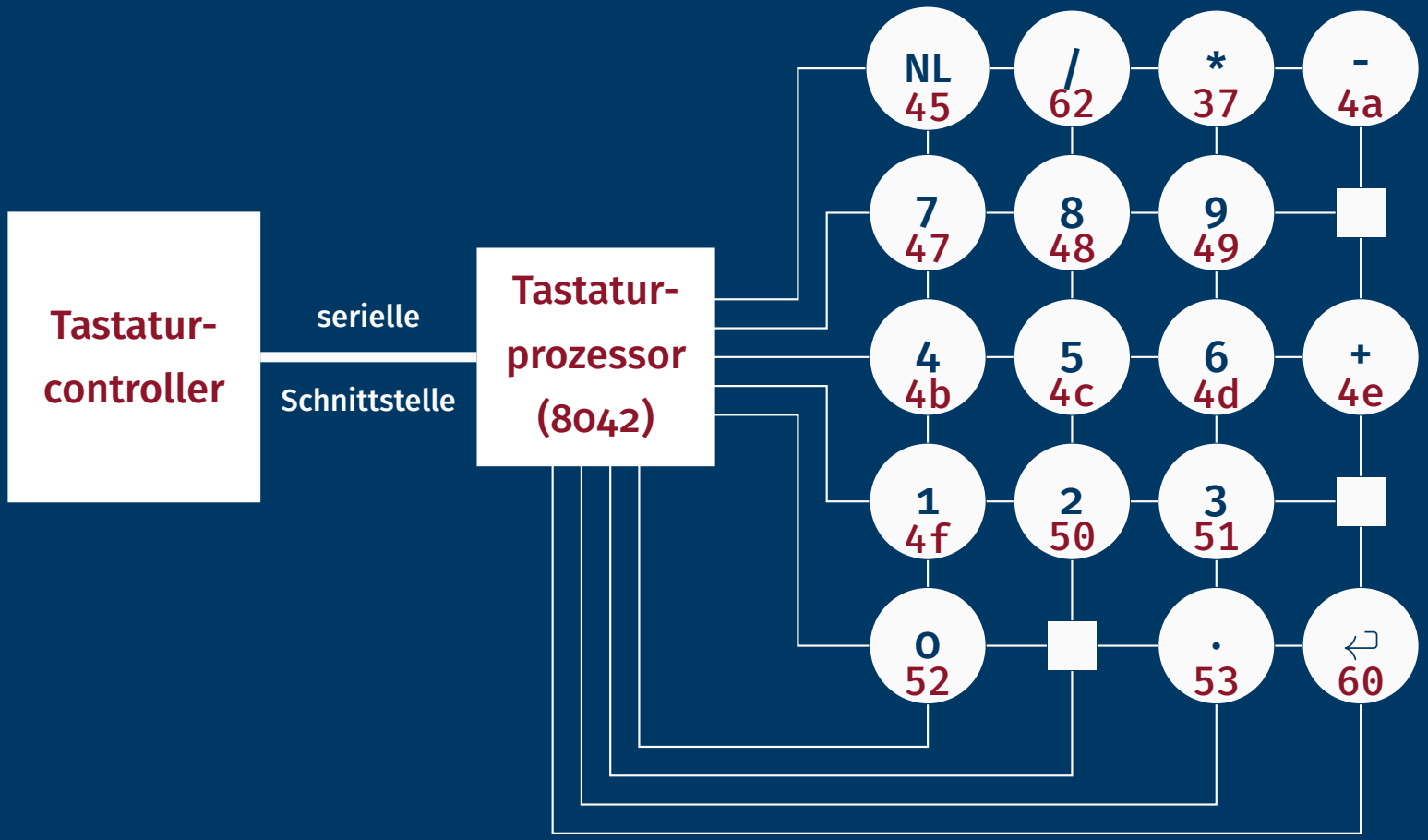
Tastatur

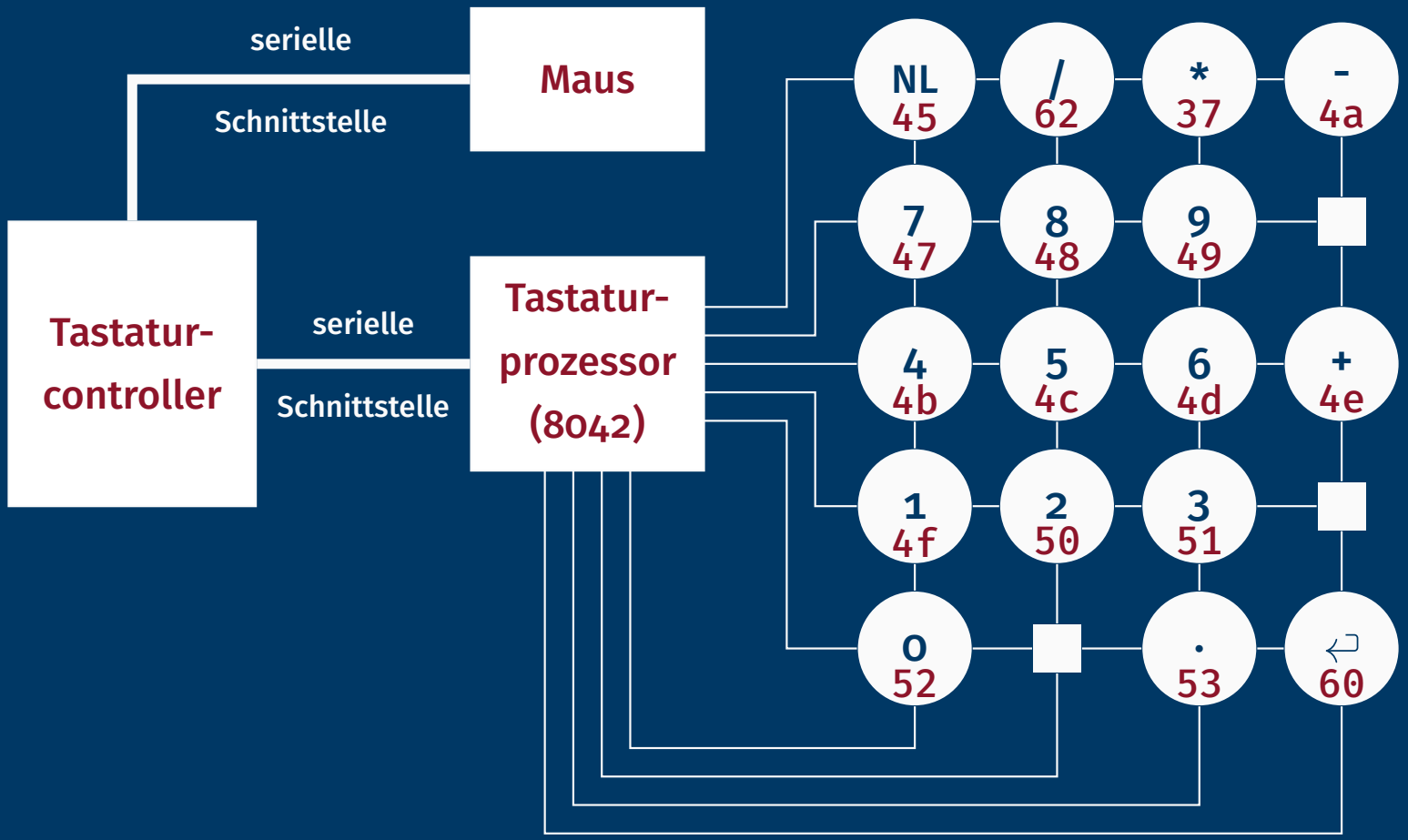
Quelle: Golem


NL
/
*
-
7
8
9
4
5
6
+
1
2
3
0
.


Tastatur-
prozessor
(8042)
NL
/
*
-
7
8
9
4
5
6
+
1
2
3
0
.


Tastatur-
prozessor
(8042)
NL
45
/
62
*
37
-
4a
7
47
8
48
9
49
4
4b
5
4c
6
4d
+
4e
1
4f
2
50
3
51
0
52
.
53
60


Tastatur-
controller
serielle
Schnittstelle
Tastatur-
prozessor
(8042)
NL
45
/
62
*
37
-
4a
7
47
8
48
9
49
4
4b
5
4c
6
4d
+
4e
1
4f
2
50
3
51
0
52
.
53
60


serielle
Schnittstelle
Maus
Tastatur-
controller
serielle
Schnittstelle
Tastatur-
prozessor
(8042)
NL
45
/
62
*
37
-
4a
7
47
8
48
9
49
4
4b
5
4c
6
4d
+
4e
1
4f
2
50
3
51
0
52
.
53
60

ASCII: Darstellung von Zeichen, 8 Bit

Scancode: Tastenkennung, 7 Bit

Makecode: Tastendruck (Scancode + `0x80`)

Breakcode: Taste loslassen

C++ Crashkurs

From: Linus Torvalds
Subject: Re: Compiling C++ kernel module + Makefile
Date: Mon, 19 Jan 2004 22:46:23 -0800 (PST)

On Tue, 20 Jan 2004, Robin Rosenberg wrote:
>
> This is the "We've always used COBOL^H^H^H^H" argument.

In fact, in Linux we did try C++ once already, back in 1992.

It sucks. Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.

The fact is, C++ compilers are not trustworthy. They were even worse in
1992, but some fundamental facts haven't changed:

 - the whole C++ exception handling thing is fundamentally broken. It's
   especially broken for kernels.
 - any compiler or language that likes to hide things like memory
   allocations behind your back just isn't a good choice for a kernel.
 - you can write object-oriented code (useful for filesystems etc) in C,
   without the crap that is C++.

In general, I'd say that anybody who designs his kernel modules for C++ is
either
 (a) looking for problems
 (b) a C++ bigot that can't see what he is writing is really just C anyway
 (c) was given an assignment in CS class to do so.

Feel free to make up (d).

        Linus

Sichtbarkeit

```
class FooBar
{
    private:
        // ...
    protected:
        // ...
    public:
        // ...
}
```

Vererbung

Syntax:

```
class Abgeleitet: Vererbungsart Basis
```

Vererbung

Syntax:

```
class Abgeleitet: Vererbungsart Basis
```

Vererbungsart	Elemente aus **Basis**
`public`	`public` und `protected` bleiben
`protected`	`public` und `protected` werden zu `protected`
`private`	`public` und `protected` werden zu `private`

Vererbung

Syntax:

```
class Abgeleitet: Vererbungsart Basis
```

Vererbungsart	Elemente aus **Basis**
`public`	`public` und `protected` bleiben
`protected`	`public` und `protected` werden zu `protected`
`private`	`public` und `protected` werden zu `private`

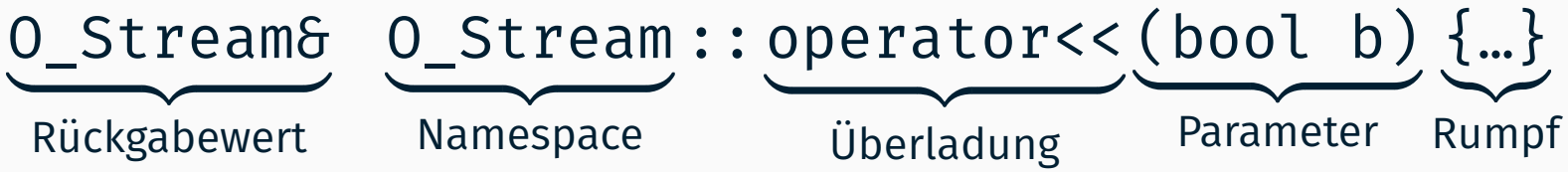
`private` Elemente können **nie** von außen erreicht werden!

Mehrfachvererbung

```
class FooBar: public Foo, protected Bar
{
    // ...
}
```

Operatorüberladung

Beispiel `O_Stream`:



Operatorüberladung

Beispiel `O_Stream`:

```
O_Stream&  O_Stream::operator<<(bool b){…}
```

Rückgabewert | Namespace | Überladung | Parameter | Rumpf

Manipulatoren:

```
O_Stream& f(O_Stream&){…}
```

Entwicklungsumgebung

Makefile Targets

make qemu QEMU **ohne** Hardware-Virtualisierung

make kvm QEMU **mit** Hardware-Virtualisierung

Makefile Targets

make qemu	QEMU **ohne** Hardware-Virtualisierung
make kvm	QEMU **mit** Hardware-Virtualisierung
make qemu-gdb	starte in QEMU und verbinde zu integrierten GDB-Stub $\rightarrow$ Fehlersuche mit gdb.

Makefile Targets

make qemu	QEMU **ohne** Hardware-Virtualisierung
make kvm	QEMU **mit** Hardware-Virtualisierung
make qemu-gdb	starte in QEMU und verbinde zu integrierten GDB-Stub $\rightarrow$ Fehlersuche mit gdb.
make netboot	für Boot am Test-Rechner ins NFS kopieren

Makefile Targets

make qemu	QEMU **ohne** Hardware-Virtualisierung
make kvm	QEMU **mit** Hardware-Virtualisierung
make qemu-gdb	starte in QEMU und verbinde zu integrierten GDB-Stub $\rightarrow$ Fehlersuche mit gdb.
make netboot	für Boot am Test-Rechner ins NFS kopieren

Suffix `-noopt` um Optimierungen auszuschalten (sonst `-O3`)

Fragen?