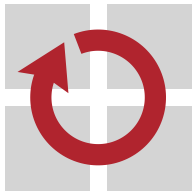
Übungen zu Grundlagen der systemnahen Programmierung in C (GSPiC) im Sommersemester 2018

2018-05-29

Bernhard Heinloth

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

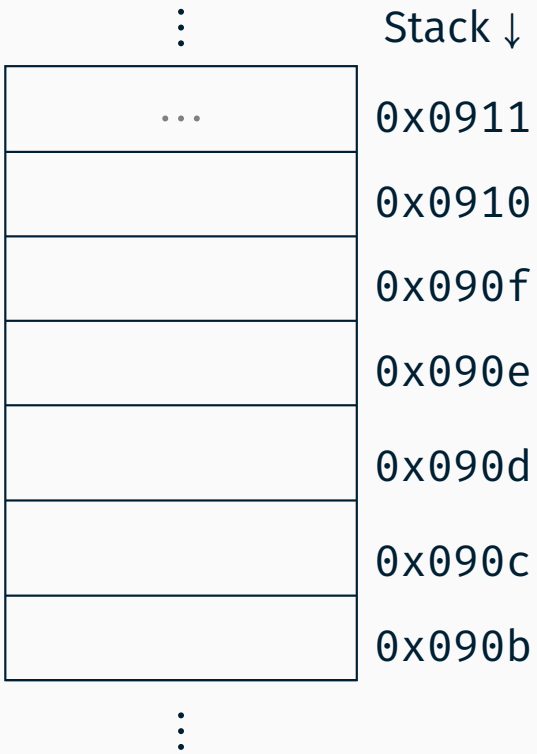
TECHNISCHE FAKULTÄT

Zeiger & Felder

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

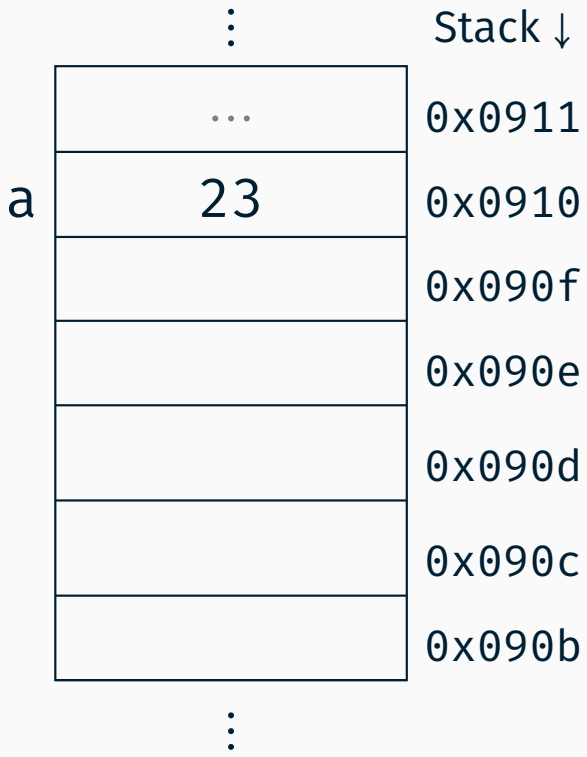
```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```



Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

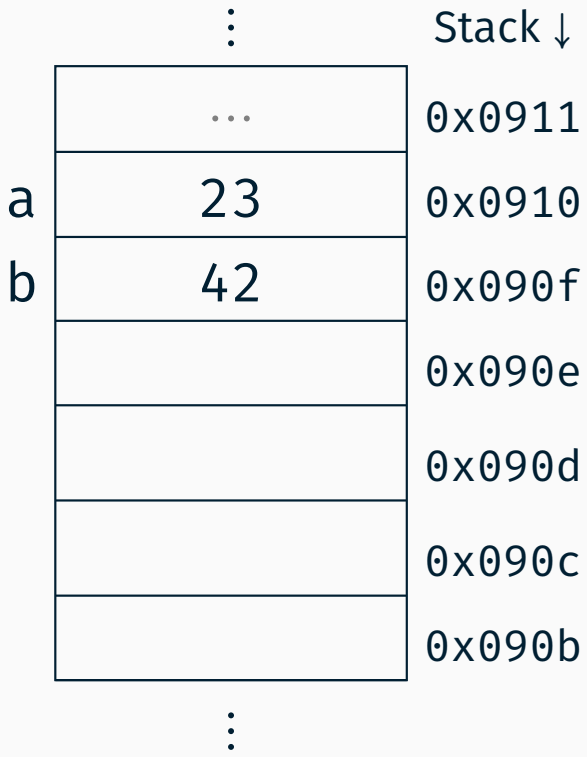


Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

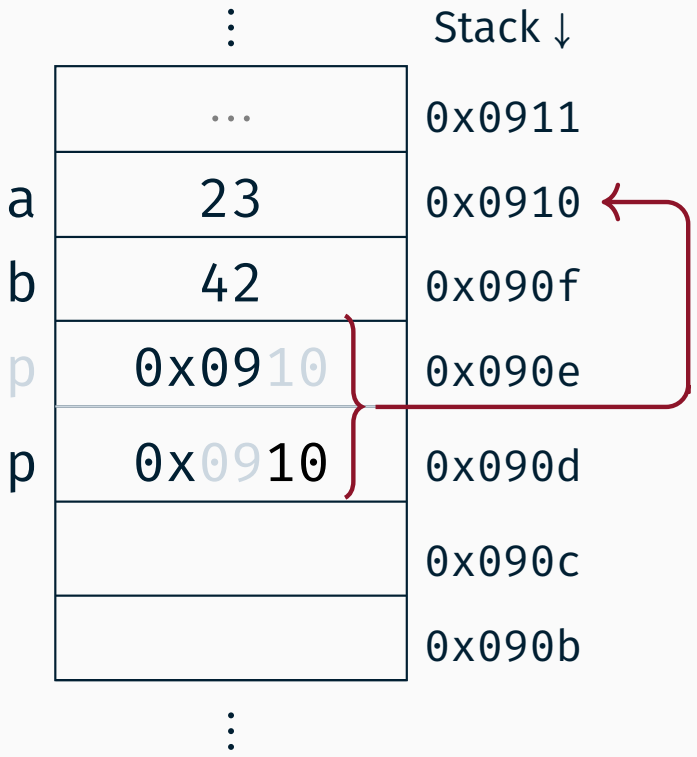


Achtung: Die genaue Anordnung der Variablen auf dem Stack ist abhängig vom Übersetzer und den gewählten Optimierungen!

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

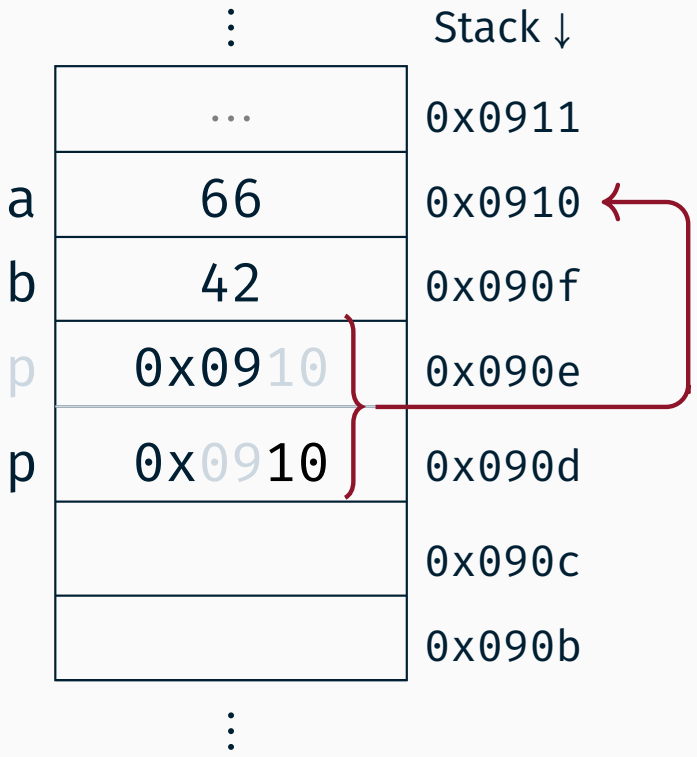


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

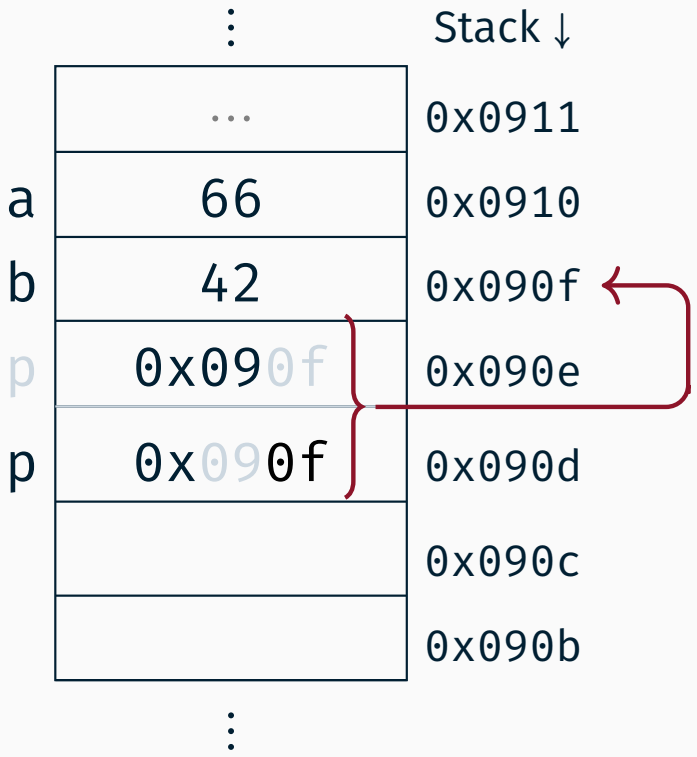


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

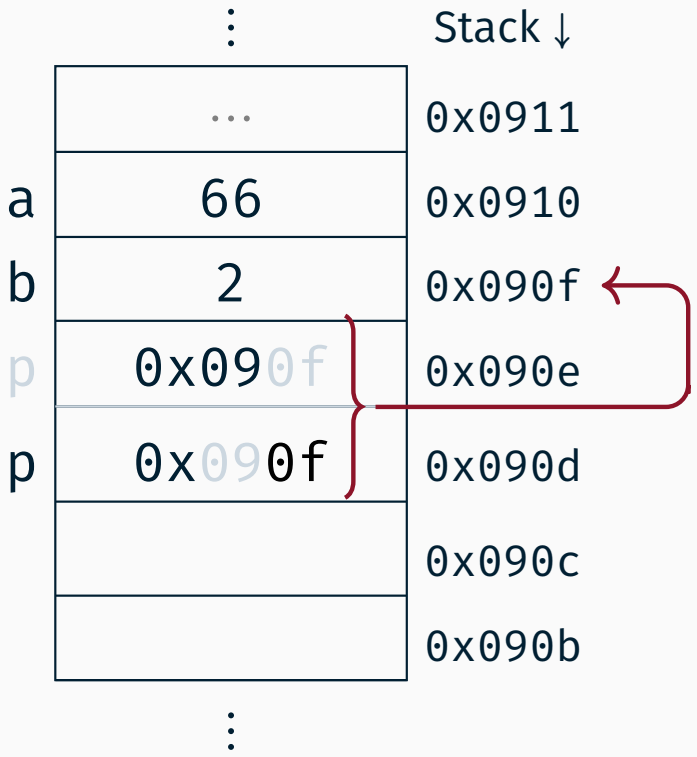


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

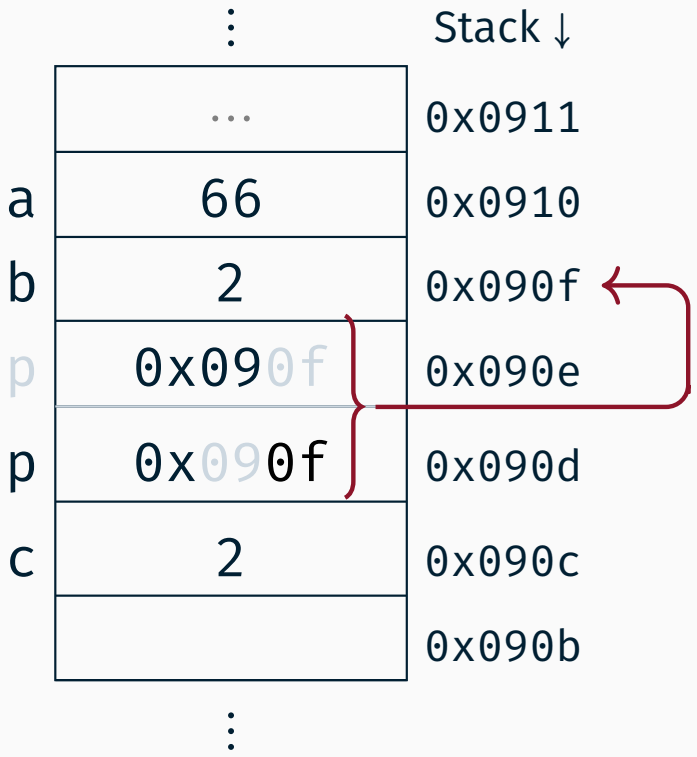


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Zeiger

- Variable: `uint8_t x`
- Zeiger: `uint8_t *y`
- Adressoperator: `&x`
- Verweisoperator: `*y`

```
01 uint8_t a = 23;
02 uint8_t b = 42;
03 uint8_t * p = &a;
04 *p = 66;
05 p = &b;
06 *p -= 40;
07 uint8_t c = *p;
```

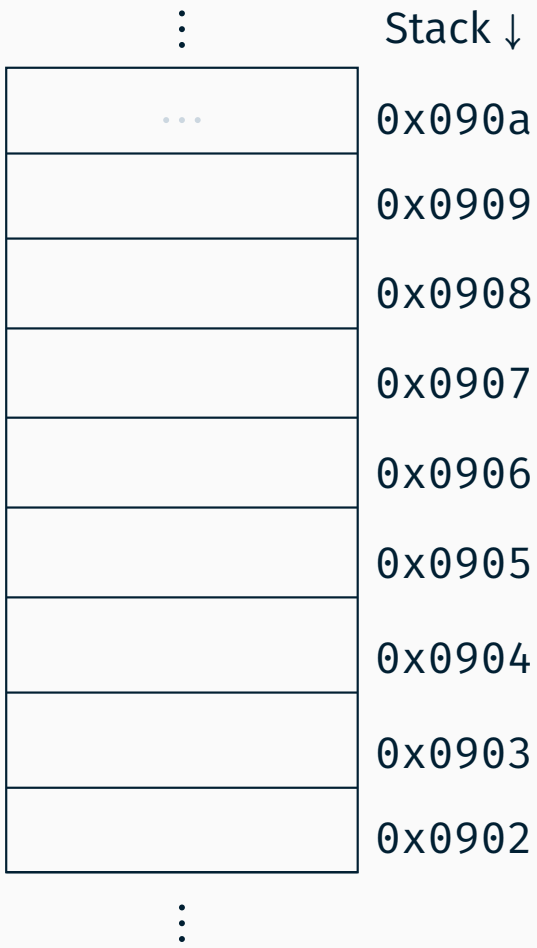


Achtung: ATmega328PB hat 8-bit Register und 16-bit Adressen

Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

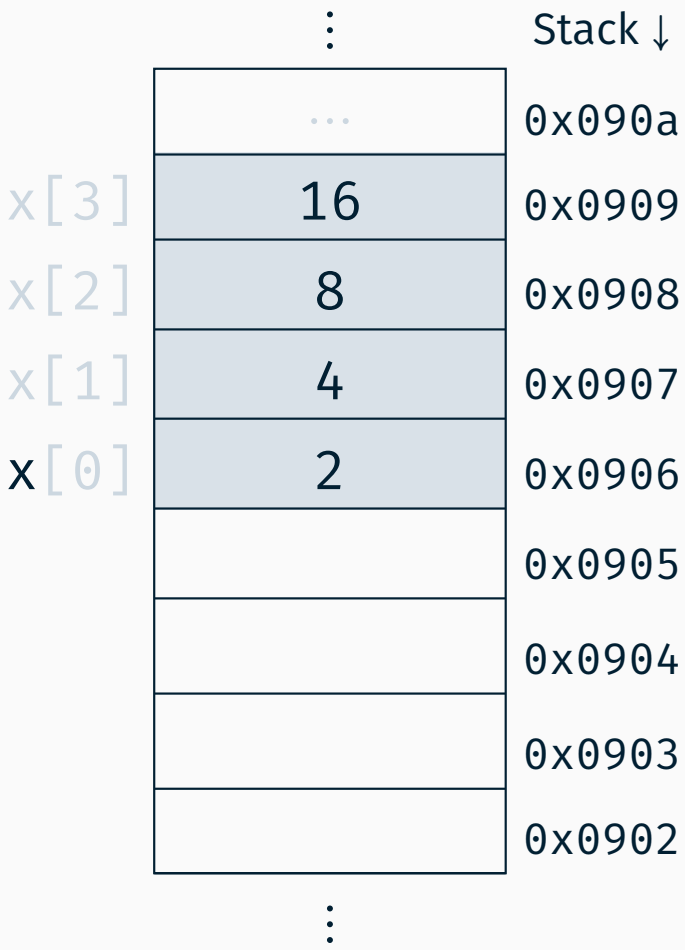
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

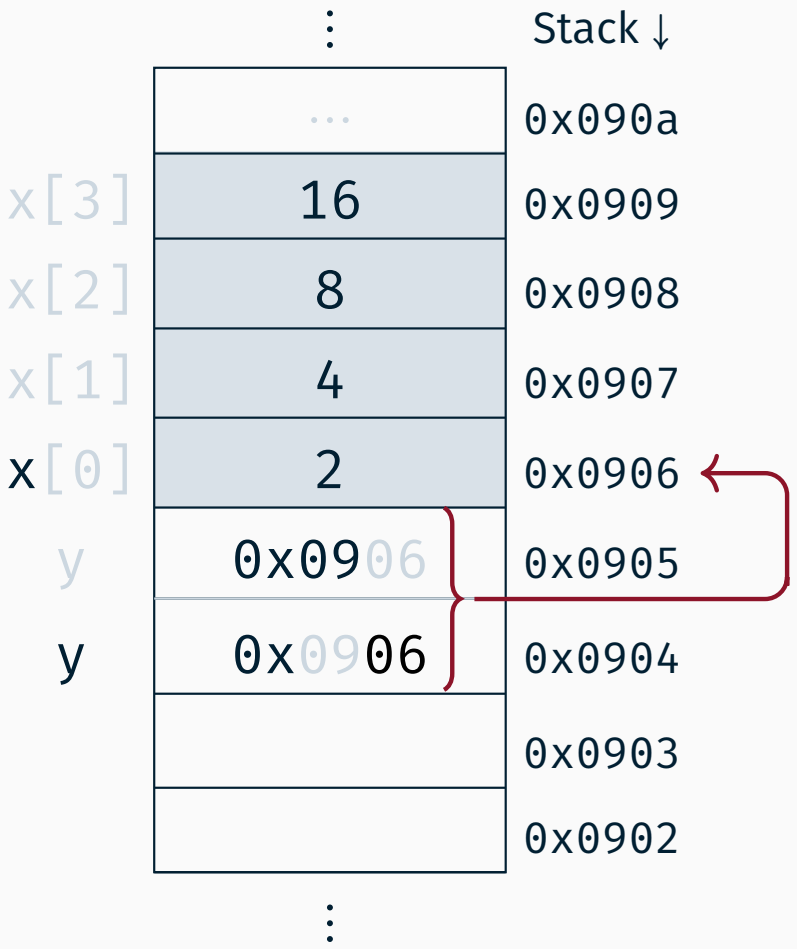
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

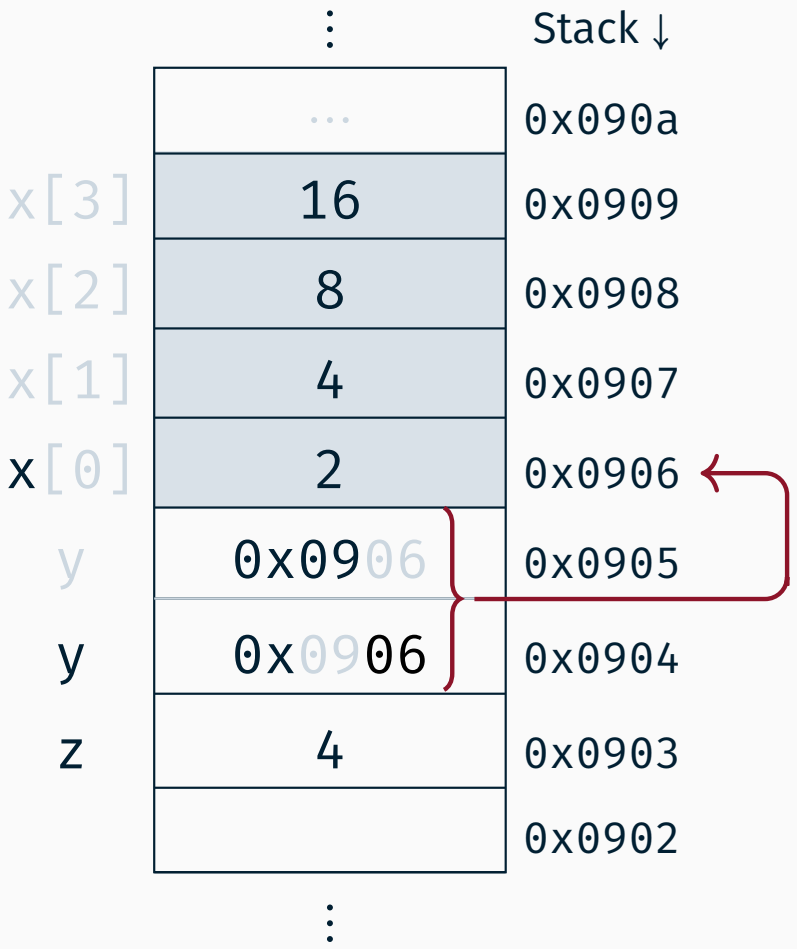
```
08  uint8_t x[] = {2,4,8,16};
09  uint8_t *y = x;
10  uint8_t z = x[1];
11  z = *y;
12  y = y+2;
13  z = *y;
14  z = x[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

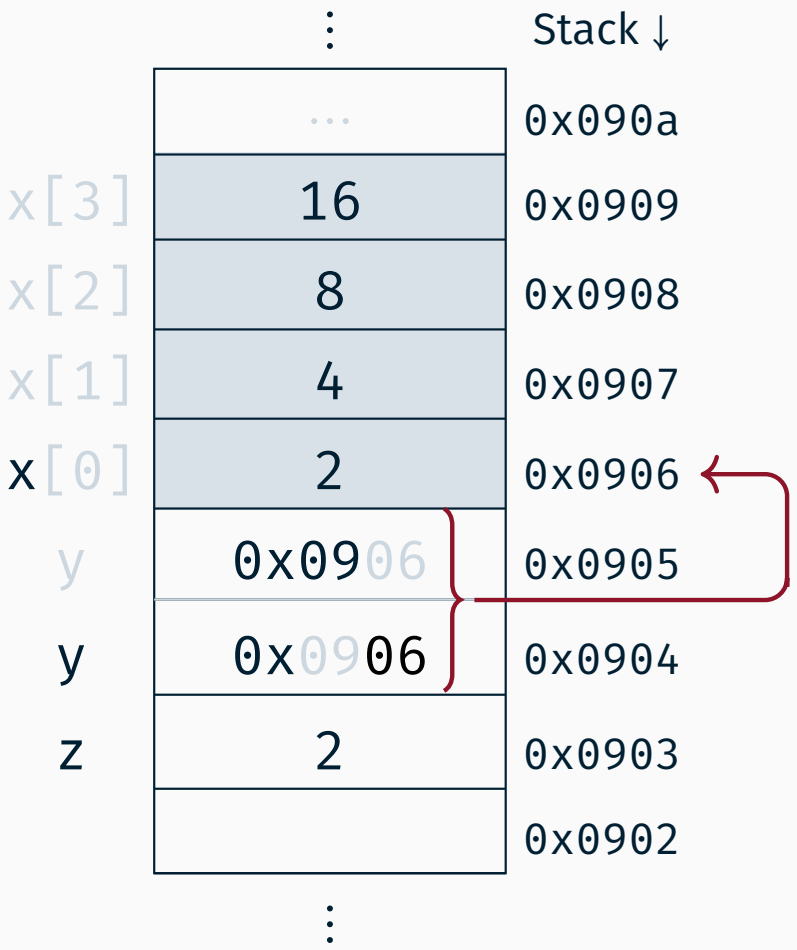
```
08  uint8_t x[] = {2,4,8,16};
09  uint8_t *y = x;
10  uint8_t z = x[1];
11  z = *y;
12  y = y+2;
13  z = *y;
14  z = x[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

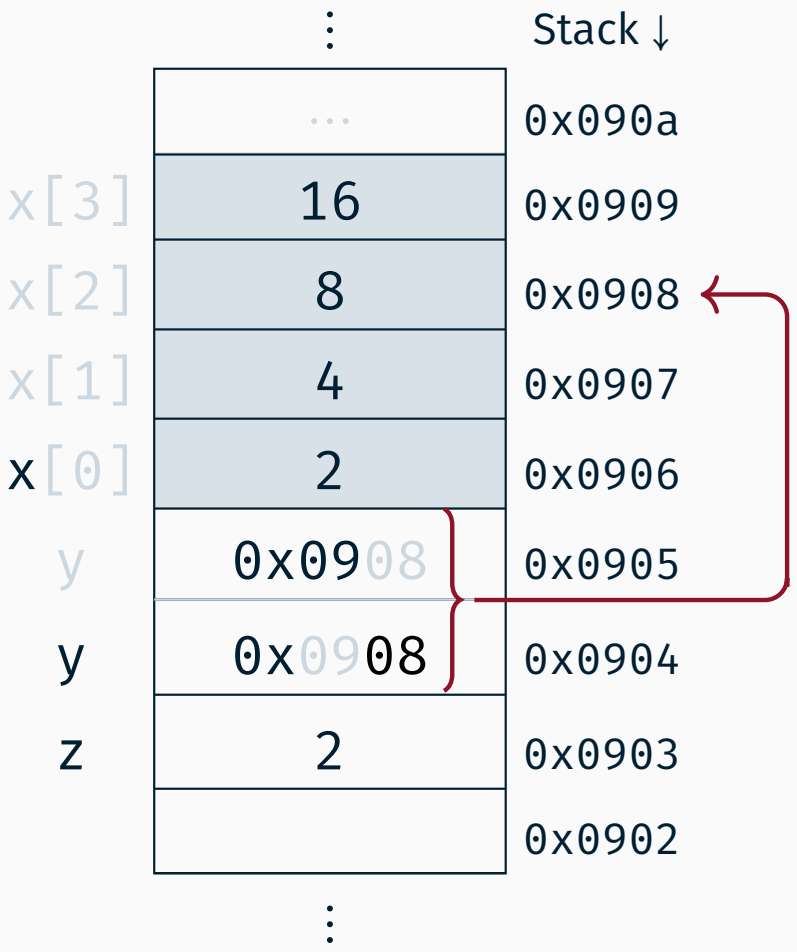
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

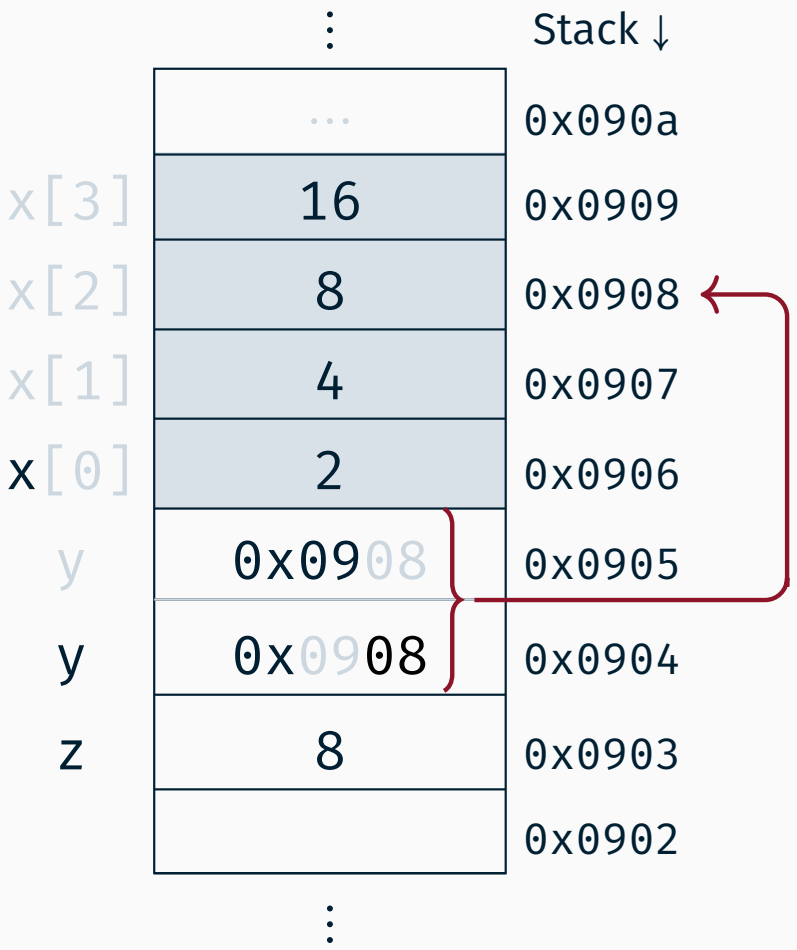
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

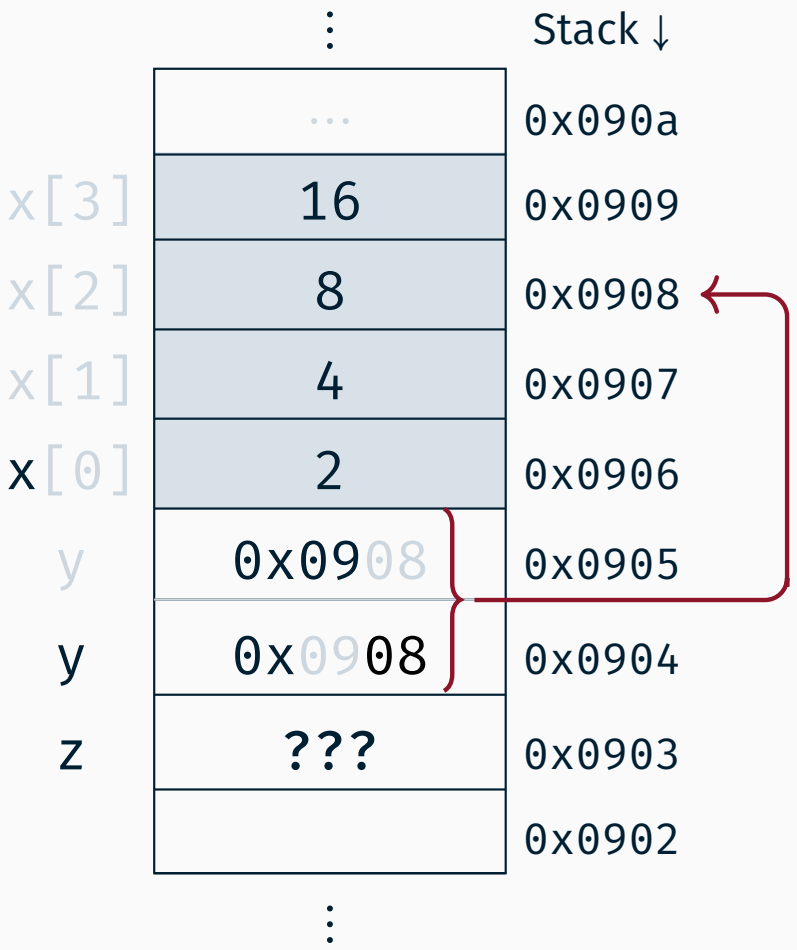
```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7];
```



Wiederholung: Felder

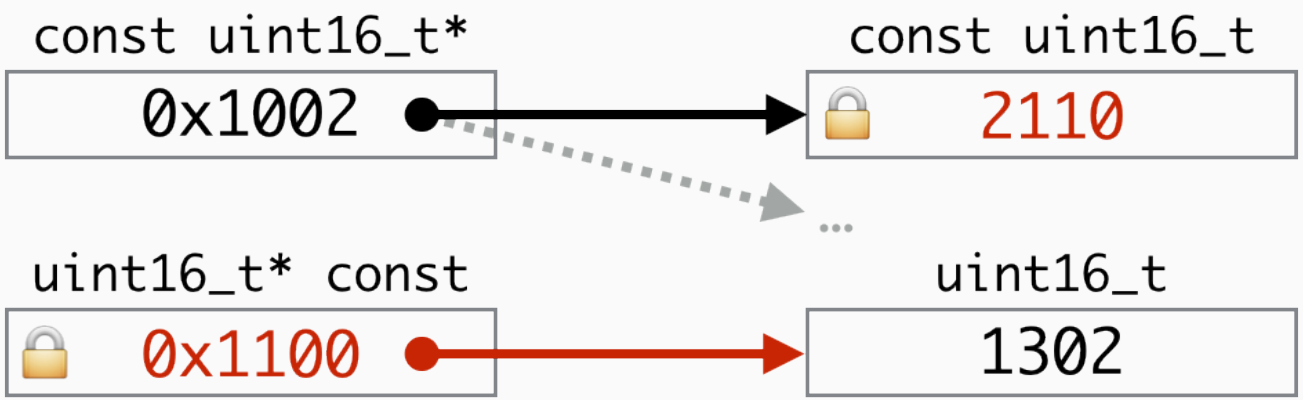
- Konstanter Zeiger: `uint8_t a[]`
- Variabler Zeiger: `uint8_t *b`
- Aktuelles Element: `*b`
- x-te Element: `b[x]`
- x-te Element: `*(b+x)`

```
08 uint8_t x[] = {2,4,8,16};
09 uint8_t *y = x;
10 uint8_t z = x[1];
11 z = *y;
12 y = y+2;
13 z = *y;
14 z = x[7]; // ϟϟϟ
```



const uint8_t* vs. uint8_t* const

- `const uint8_t*`
  - ein Pointer auf einen `uint8_t`-**Wert**, der konstant ist
  - Wert nicht über den Pointer veränderbar
- `uint8_t* const`
  - ein **konstanter Pointer** auf einen (beliebigen) `uint8_t`-Wert
  - Pointer darf nicht mehr auf eine andere Speicheradresse zeigen



Hands-on

Hands-on: Strukturen, Felder & Zeiger

- `struct` für GPS-Koordinaten
- Feld von GPS-Koordinaten
- Call-by-Value vs. Call-by-Reference
- Zeigerarithmetik
- Funktionszeiger

Ausgabe über die serielle Konsole (nach Tastendruck auf `BUTTON0`)

Exkurs: Serielle Konsole

Die **serielle Konsole**

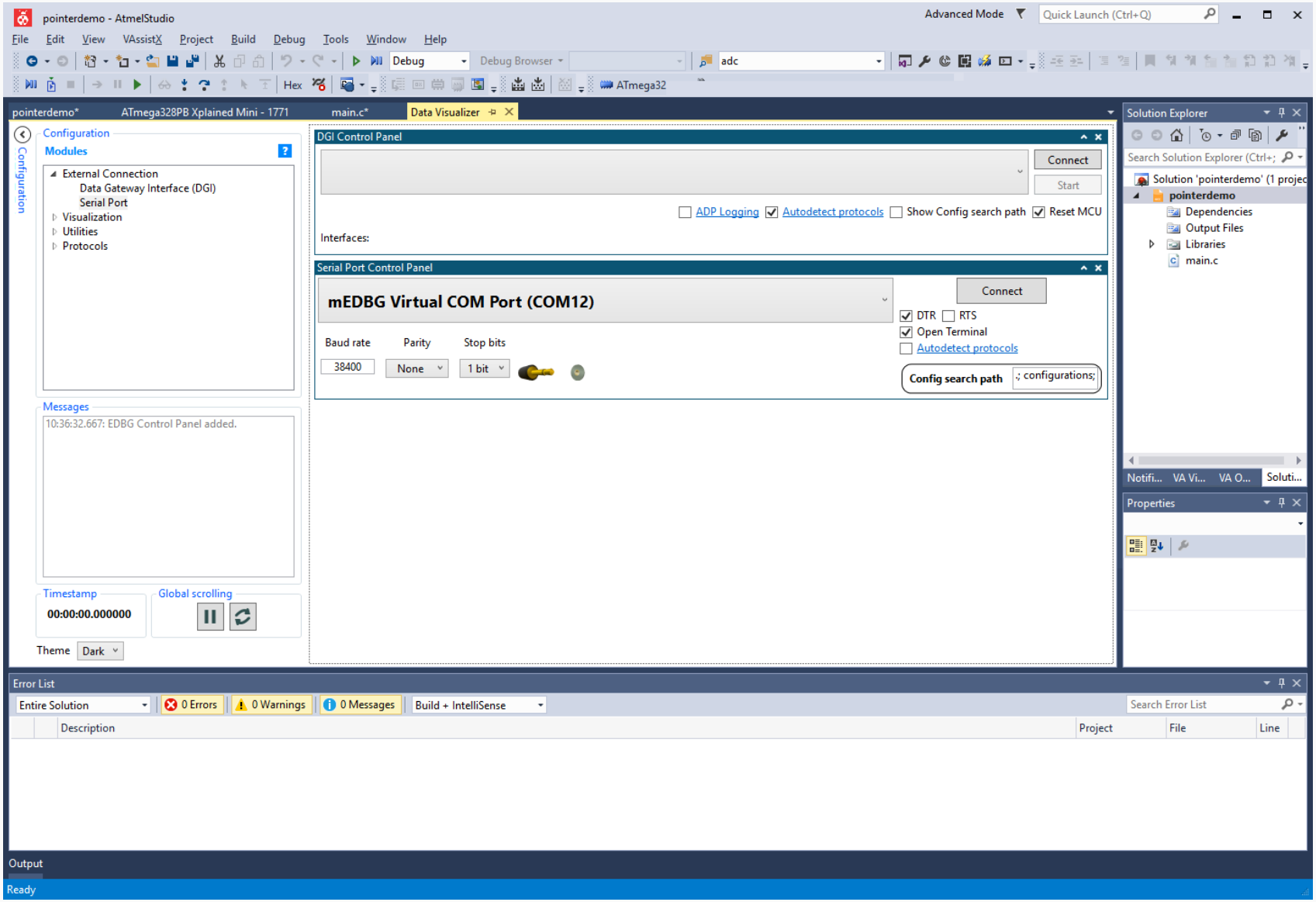
- erlaubt dem Programm auf dem SPiCboard einfache Kommunikation (Aus- und Eingaben) mit dem PC
- nutzt die serielle Schnittstelle (UART)
- benötigt `#include <console.h>`
- wird durch die Initialisierungsfunktion `sb_console_connect_default();` auf 38400 bit/s (ohne Paritätsbit und mit einem Stoppbit) eingestellt
- ermöglicht Ausgabe über die libspicboard-Funktion `sb_console_putString("Hallo Welt!n");`
- oder über mächtige Standardbibliotheksfunktionen wie `printf` und `scanf` (aber nicht ganz trivial).

Exkurs: Serielle Konsole

Verwendung in Atmel Studio 7

- im Menü *Tools* den *Data Visualizer* starten
- auf der Seite *Configuration* auswählen
- in der Gruppe *Modules* den Punkt *External Connection* erweitern und *Serial Port* auswählen.
- das neu angezeigte *Serial Port Control Panel* anpassen:
  - Verbindung wählen, etwa *mEDBG Virtual COM Port (COM4)*
  - die Übertragungsrate (*Baud rate*) auf *38400* setzen
  - Parität (*Parity*) mit *None* deaktiveren
  - einfaches Stoppbit (*Stop bits* auf *1*)
  - Haken bei *DTR* (und *RTS* auf langsamen Systemen)
  - *Connect* zum verbinden

Exkurs: Serielle Konsole



Exkurs: Serielle Konsole