

Angewandte Informatik

Prof. Dr. rer. nat. Peter Gerwinski

3. Dezember 2015

Angewandte Informatik

1 Einführung

2 Einführung in C

3 Bibliotheken

4 Algorithmen

4.1 Differentialgleichungen

4.2 Rekursion

4.3 Stack und FIFO

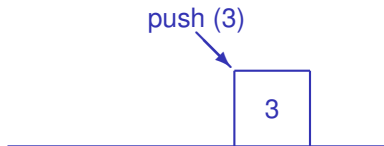
4.4 Aufwandsabschätzungen

5 Hardwarenahe Programmierung

...

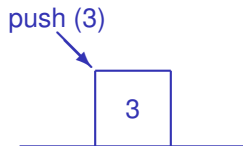
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

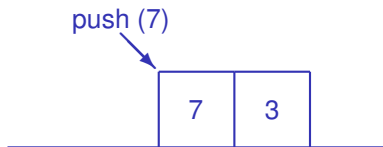
„Last In – First Out“



LIFO = Stack = Stapel

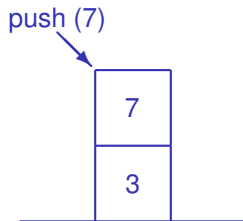
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

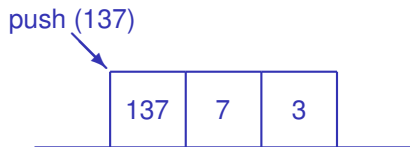
„Last In – First Out“



LIFO = Stack = Stapel

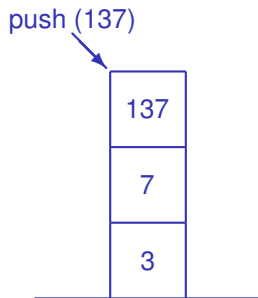
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

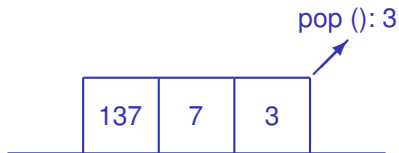
„Last In – First Out“



LIFO = Stack = Stapel

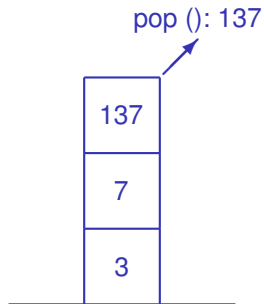
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

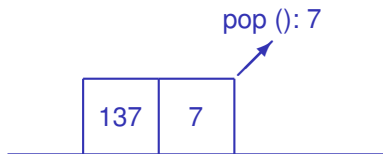
„Last In – First Out“



LIFO = Stack = Stapel

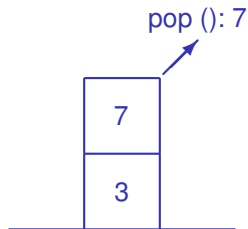
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

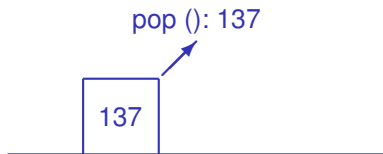
„Last In – First Out“



LIFO = Stack = Stapel

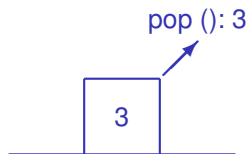
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“



LIFO = Stack = Stapel

Angewandte Informatik

1 Einführung

2 Einführung in C

3 Bibliotheken

4 Algorithmen

4.1 Differentialgleichungen

4.2 Rekursion

4.3 Stack und FIFO

4.4 Aufwandsabschätzungen

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

5.2 I/O-Ports

5.3 Interrupts

5.4 volatile-Variable

5.5 Software-Interrupts

...

5 Hardwarenahe Programmierung

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

5.1.1 Zahlensysteme

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

5.1.1 Zahlensysteme

Basis		Beispiel
2	Binärsystem	1 0000 0011
8	Oktalsystem	0403
10	Dezimalsystem	259
16	Hexadezimalsystem	0x103
256	IP-Adressen (IPv4)	0.0.1.3

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

5.1.1 Zahlensysteme

Oktal- und Hexadezimal-Zahlen lassen sich ziffernweise in Binär-Zahlen umrechnen:

000	0	0000	0	1000	8
001	1	0001	1	1001	9
010	2	0010	2	1010	A
011	3	0011	3	1011	B
100	4	0100	4	1100	C
101	5	0101	5	1101	D
110	6	0110	6	1110	E
111	7	0111	7	1111	F

5.1.2 Bit-Operationen in C

C-Operator	Verknüpfung	Anwendung
&	Und	Bits gezielt löschen
	Oder	Bits gezielt setzen
^	Exklusiv-Oder	Bits gezielt invertieren
~	Nicht	Alle Bits invertieren
<<	Verschiebung nach links	Maske generieren
>>	Verschiebung nach rechts	Bits isolieren

Aufgabe: Schreiben Sie C-Funktionen, die ein „Array von Bits“ realisieren, z. B.

void set_bit (int i);	Bei Index <i>i</i> auf 1 setzen
void clear_bit (int i);	Bei Index <i>i</i> auf 0 setzen
int get_bit (int i);	Bei Index <i>i</i> lesen

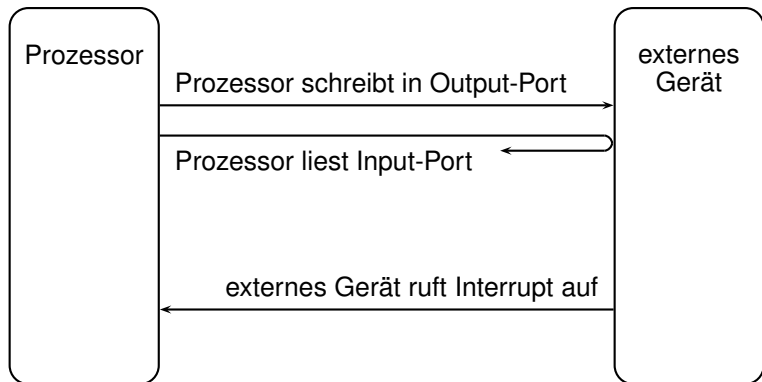
Hinweise:

- Die Größe des Bit-„Arrays“ (z. B. 1000) dürfen Sie als *vorher bekannt* voraussetzen.
- Sie benötigen ein Array, z. B. von **char**- oder **int**-Variablen.
- Sie benötigen eine Division (/) sowie den Divisionsrest (Modulo: %).

5.2 I/O-Ports

5.3 Interrupts

Kommunikation mit externen Geräten



5.2 I/O-Ports

In Output-Port schreiben = Leitungen ansteuern

Datei: `RP6Base/RP6Base_Examples/RP6Examples_20080915/
RP6Lib/RP6base/RP6RobotBaseLib.c`

Suchbegriff: `setMotorDir`

```
void setMotorDir(uint8_t left_dir, uint8_t right_dir)
```

```
{
```

```
    /* ... */
```

```
    if(left_dir)
```

```
        PORTC |= DIR_L;
```

```
    else
```

```
        PORTC &= ~DIR_L;
```

```
    if(right_dir)
```

```
        PORTC |= DIR_R;
```

```
    else
```

```
        PORTC &= ~DIR_R;
```

```
}
```

Manipulation einzelner Bits

→ Steuerung der Motordrehrichtung

Output-Port

5.3 Interrupts

Externes Gerät ruft (per Stromsignal) Unterprogramm auf
Zeiger hinterlegen: „Interrupt-Vektor“

Datei: [RP6Base/RP6Base_Examples/RP6Examples_20080915/
RP6Lib/RP6base/RP6RobotBaseLib.c](#)
Suchbegriff: ISR

„Dies ist ein Interrupt-Handler.“

Interrupt-Vektor 0 darauf zeigen lassen

Schreibweise herstellerspezifisch!

```
ISR (INT0_vect)
{
    mleft_dist++;
    mleft_counter++;
    /* ... */
}
```

Aufruf durch Sensor an Encoder-Scheibe
→ Entfernungsmessung

5.4 volatile-Variable

```
volatile uint16_t mleft_counter;
```

```
/* ... */
```

„Immer lesen und schreiben. Nicht wegoptimieren.“

```
volatile uint16_t mleft_dist;
```

```
/* ... */
```

```
ISR (INT0_vect)
```

```
{  
    mleft_dist++;  
    mleft_counter++;  
    /* ... */  
}
```

```
int main (void)
```

```
{  
    int prev_mleft_dist = mleft_dist;  
    while (mleft_dist == prev_mleft_dist)  
        /* just wait */;  
}
```

5.5 Software-Interrupts

```
mov ax, 0012
```

```
int 10
```

5.5 Software-Interrupts

`mov ax, 0012` ← Parameter in Prozessorregister

`int 10` ← Funktionsaufruf über Interrupt-Vektor

5.5 Software-Interrupts

`mov ax, 0012` ← Parameter in Prozessorregister
`int 10` ← Funktionsaufruf über Interrupt-Vektor

Beispiel: VGA-Grafikkarte

- Modus setzen: `mov ah, 00`
- Grafikmodus: `mov al, 12`
- Textmodus: `mov al, 03`

5.5 Software-Interrupts

`mov ax, 0012` ← Parameter in Prozessorregister
`int 10` ← Funktionsaufruf über Interrupt-Vektor

Beispiel: VGA-Grafikkarte

- Modus setzen: `mov ah, 00`
- Grafikmodus: `mov al, 12`
- Textmodus: `mov al, 03`

Verschiedene Farben: Output-Ports

- *Graphics Register*: Index `03CE`, Daten `03CF`
- Index 0: *Set/Reset Register*
- Index 1: *Enable Set/Reset Register*
- Index 8: *Bit Mask Register*
- Jedes Bit steht für Schreibzugriff auf eine Speicherbank.
- 4 Speicherbänke → 16 Farben

Angewandte Informatik

1 Einführung

2 Einführung in C

3 Bibliotheken

4 Algorithmen

4.1 Differentialgleichungen

4.2 Rekursion

4.3 Stack und FIFO

4.4 Aufwandsabschätzungen

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

5.2 I/O-Ports

5.3 Interrupts

5.4 volatile-Variable

5.5 Software-Interrupts

...