

Systemnahe Programmierung in C

16 μ C-Systemarchitektur – Prozessor

J. Kleinöder, D. Lohmann, V. Sieh, P. Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

Sommersemester 2026

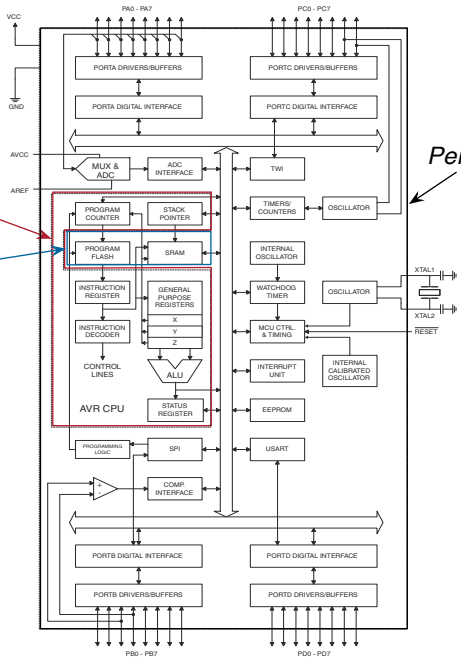
<http://sys.cs.fau.de/lehre/ss26>



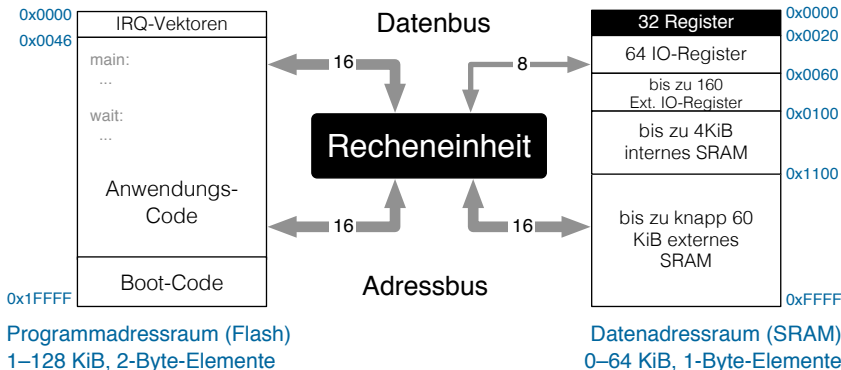
Beispiel ATmega32: Blockschaltbild

CPU-Kern
Speicher

Peripherie

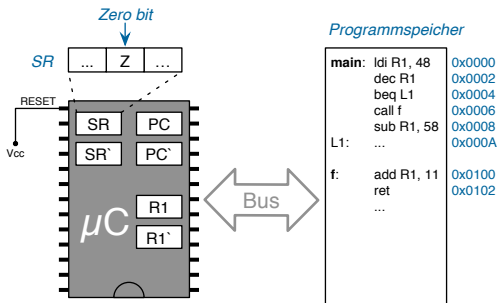


Beispiel ATmega-Familie: CPU-Architektur



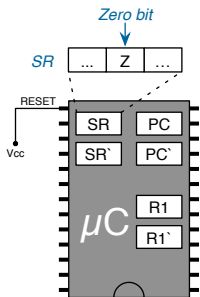
- Harvard-Architektur (getrennter Speicher für Code und Daten)
- Peripherie-Register sind in den Speicher eingebündelt
~> ansprechbar wie globale Variablen





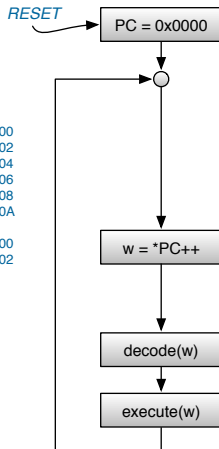
- Hier am Beispiel eines sehr einfachen Pseudoprocessors
 - Nur ein Vielzweckregister (R1)
 - Programmzähler (PC) und Statusregister (SR) (+ „Schattenkopien“)
 - Kein Datenspeicher, kein Stapel \rightsquigarrow Programm arbeitet nur auf Registern

Wie arbeitet ein Prozessor?

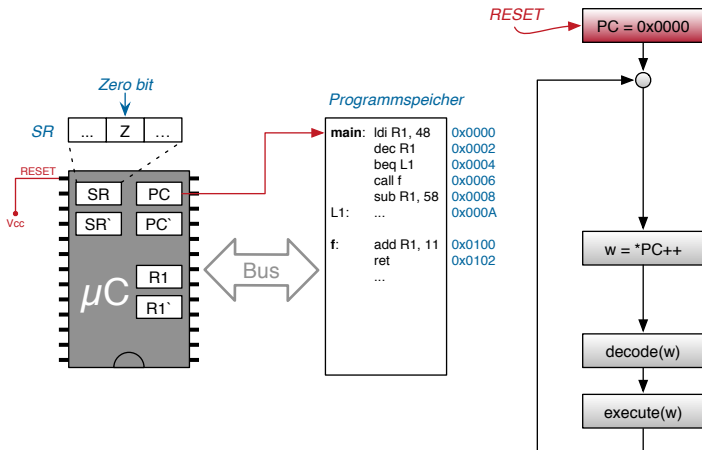


Programmspeicher

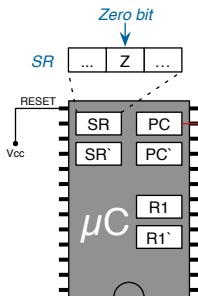
```
main: ldi R1, 48 0x0000
      dec R1    0x0002
      beq L1   0x0004
      call f   0x0006
      sub R1, 58 0x0008
      ...     0x000A
L1:   ...
f:   add R1, 11 0x0100
      ret      0x0102
      ...
```



Wie arbeitet ein Prozessor?

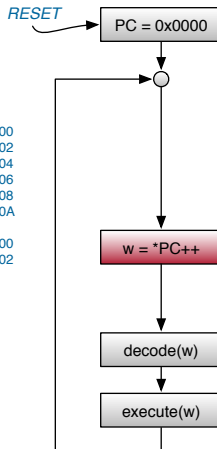


Wie arbeitet ein Prozessor?

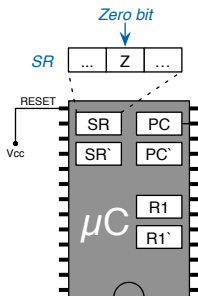


Programmspeicher

```
main: ldi R1, 48 0x0000
      dec R1    0x0002
      beq L1   0x0004
      call f   0x0006
      sub R1, 58 0x0008
      ...     0x000A
L1:   ...
f:   add R1, 11 0x0100
      ret      0x0102
      ...
```

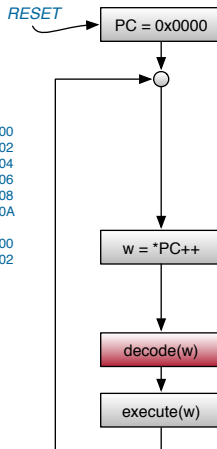


Wie arbeitet ein Prozessor?

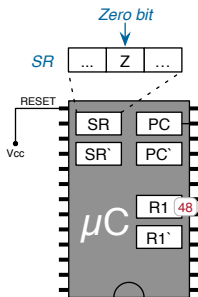


Programmspeicher

main:	<code>ldi R1, 48</code>	0x0000
	<code>dec R1</code>	0x0002
	<code>beq L1</code>	0x0004
	<code>call f</code>	0x0006
	<code>sub R1, 58</code>	0x0008
	<code>...</code>	0x000A
L1:	<code>...</code>	
f:	<code>add R1, 11</code>	0x0100
	<code>ret</code>	0x0102
	<code>...</code>	

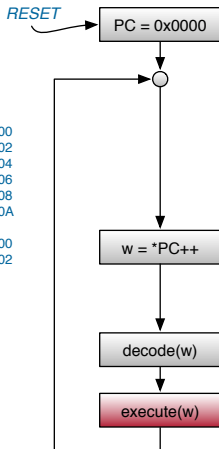


Wie arbeitet ein Prozessor?

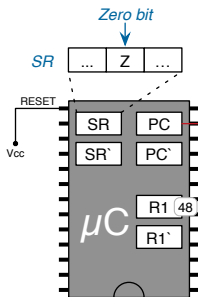


Programmspeicher

main:	<code>ldi R1, 48</code>	0x0000
	<code>dec R1</code>	0x0002
	<code>beq L1</code>	0x0004
	<code>call f</code>	0x0006
	<code>sub R1, 58</code>	0x0008
	<code>...</code>	0x000A
L1:	<code>...</code>	
f:	<code>add R1, 11</code>	0x0100
	<code>ret</code>	0x0102
	<code>...</code>	

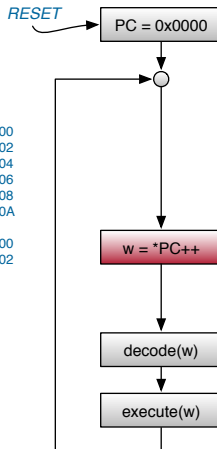


Wie arbeitet ein Prozessor?

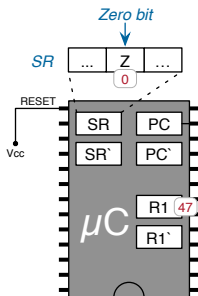


Programmspeicher

```
main: ldi R1, 48 0x0000
      dec R1    0x0002
      beq L1   0x0004
      call f   0x0006
      sub R1, 58 0x0008
      ...     0x000A
L1:   ...
f:   add R1, 11 0x0100
      ret      0x0102
      ...
```

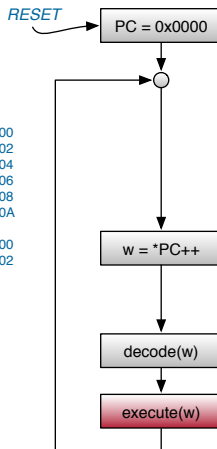


Wie arbeitet ein Prozessor?



Programmspeicher

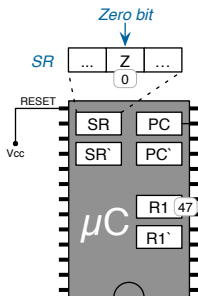
```
main: ldi R1, 48 0x0000
      dec R1    0x0002
      beq L1   0x0004
      call f   0x0006
      sub R1, 58 0x0008
      ...     0x000A
L1:   ...
f:   add R1, 11 0x0100
      ret      0x0102
      ...
```



```
w: dec <R>
R -= 1
if(R == 0) Z = 1
else Z = 0
```

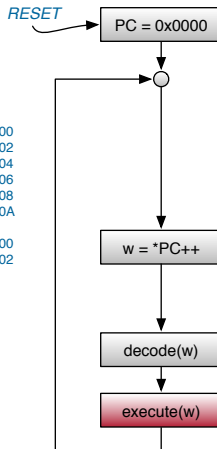


Wie arbeitet ein Prozessor?



Programmspeicher

```
main: ldi R1, 48 0x0000
      dec R1    0x0002
      beq L1   0x0004
      call f   0x0006
      sub R1, 58 0x0008
      ...     0x000A
L1:   ...
f:   add R1, 11 0x0100
      ret      0x0102
      ...
```

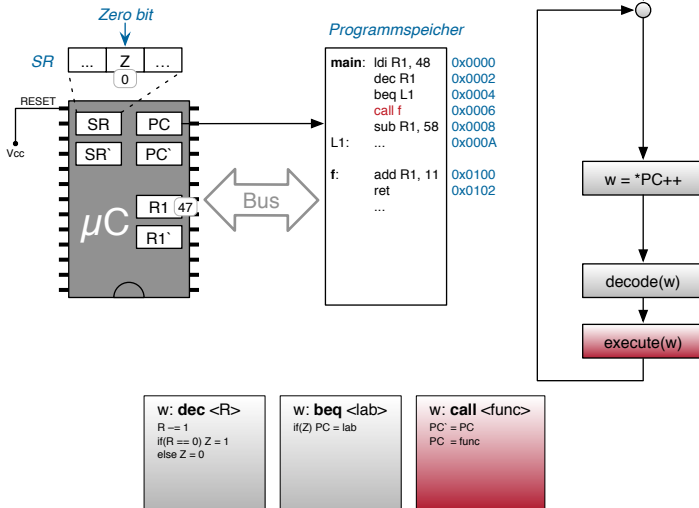


w: dec <R>
R -= 1
if(R == 0) Z = 1
else Z = 0

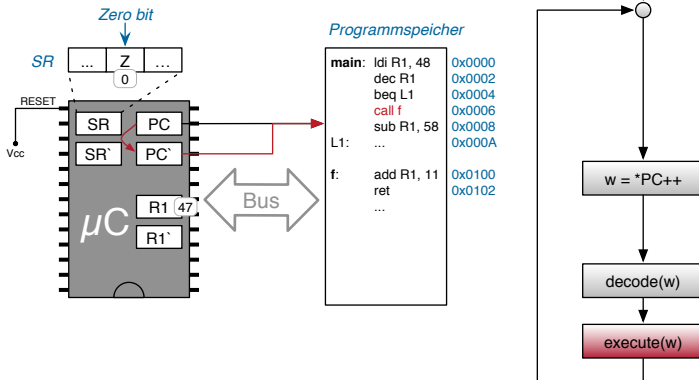
w: beq <lab>
if(Z) PC = lab



Wie arbeitet ein Prozessor?



Wie arbeitet ein Prozessor?



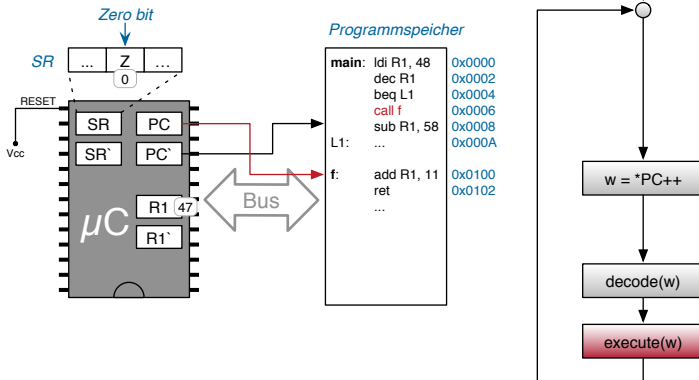
w: dec <R>
R -= 1
if(R == 0) Z = 1
else Z = 0

w: beq <lab>
if(Z) PC = lab

w: call <func>
PC' = PC
PC = func



Wie arbeitet ein Prozessor?



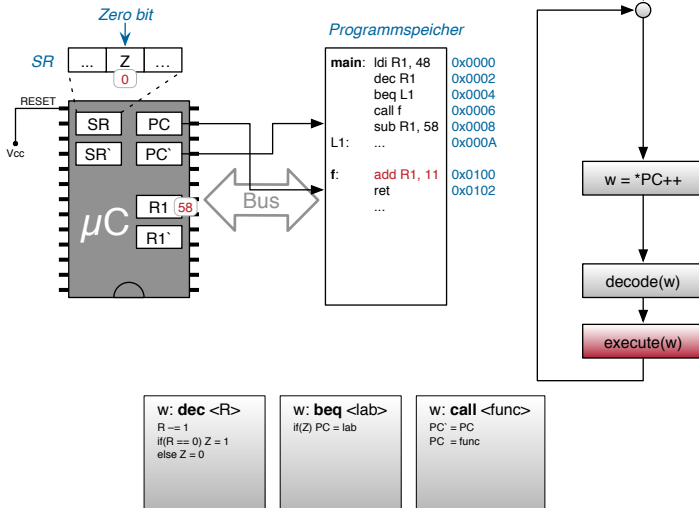
w: dec <R>
R -= 1
if(R == 0) Z = 1
else Z = 0

w: beq <lab>
if(Z) PC = lab

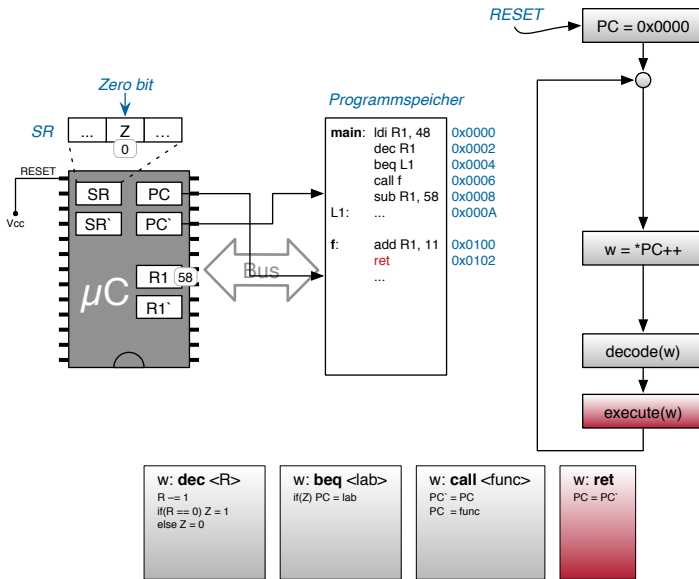
w: call <func>
PC' = PC
PC = func



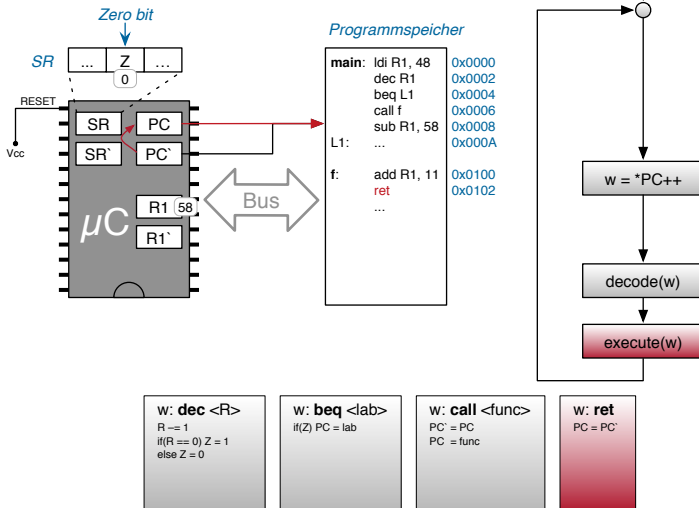
Wie arbeitet ein Prozessor?



Wie arbeitet ein Prozessor?



Wie arbeitet ein Prozessor?



Wie arbeitet ein Prozessor?

