6

06. Assembler-Programmierung

Programmstrukturen des ATMega32

emg
DMM

6

Literatur

www.mikrocontroller.net Alles über AVR

www.avr-asm-tutorial.net AVR-Assembler-Einführung

www.microschematic.com AVR-Aufbau, Register, Befehle
2008: www.ouravr.com/attachment/microschematic/INDEX.swf

Atmel Dokumentation

emg
DMM

6

Mögliche Interrupts

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Sprung zum Programm

Externe Interrupts

Timer/Counter

Schnittstellen

A/D-Umsetzung
EEPROM
Analog Komparator
Schnittstellen
Flash-Memory

emg
DMM

6

Externe Interrupts

werden kontrolliert vom GICR- und MCUCR-Register:

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	General Interrupt Control Register
Initial Value	0	0	0	0	0	0	0	0	

IVSEL: 0 = Interrupt-Routinen am Anfang Flash-Speicher
IVCE: Interrupt Vector Change Enable: Interrupts zulassen

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	MCU Control Register
Initial Value	0	0	0	0	0	0	0	0	

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

emg
DMM

Interrupt.asm

Verzweigungsbefehle

Status-Register Bits

C	Carry, gesetzt, wenn Byte bei Operation überläuft: 0xF0 -> 0x00
Z	Zero, gesetzt, wenn Null als Ergebnis auftritt
N	Negative, gesetzt, wenn höchstes Bit gesetzt wird, das auch als Vorzeichenbit interpretiert werden kann
V	Validity, gesetzt, wenn im Zweier-Complement ein Überlauf eintritt
S	Sign, gesetzt, wenn nur entweder N oder V gesetzt sind (Exklusiv-Oder) $S = N \oplus V$
H	HalfCarry, gesetzt, wenn Halbbyte bei Operation überläuft: 0x0F -> 0x10
T	Speicher für BST BitSTore und BLD BitLoaD-Befehle
I	Globales Interrupt Enable, 0 nach Interruptbearbeitung, von RETI wieder gesetzt

Status-Verzweigungsbefehle

Durch Abfrage des Statusregisters

I	T	H	S	V	N	Z	C

`BRBS/BRBC`	Branch, if Statusbit set / cleared
`BRCS/BRCC`	Branch, if Carrybit set / cleared
`BRSH/BRLO`	Branch, if Same_Higher / Lower (Z)
`BRMI/BRPL`	Branch, if negative (MInus) / positive (PLus)
`BRVS/BRVC`	Branch, if Zweierkomplementbit V set / cleared
`BRGE/BRLT`	Branch, if Greater Equal 0/ Less Than (Sign)
`BRHS/BRHC`	Branch, if HalfCarrybit set / cleared
`BRTS/BRTC`	Branch, if T-bit set / cleared
`BRIE/BRID`	Branch, if Interrupt Enabled / Disabled

Macros

Macro-Verwendung

Befehlsfolgen, die im Programm immer wieder verwendet werden sollen, können in einem Macro zusammengefasst werden. Macros sind reine Textbausteine im Programmtext, die beim Assemblerlauf an allen verwendeten Stellen wieder eingesetzt werden.

+ Macros machen den Programmtext übersichtlicher
+ Macros können bis zu 10 Parameter verwenden (@0...@9)
  (Übergabe mit Kommaliste nach Macroname)

-Macros sparen keinen Speicherplatz
-Macros können nur absolute Adressierung verwenden

Macro-Definition

Die Macros sind Assembler-Direktiven:

```
.MACRO Delay1      ; Verzögerung um 4 Takte
        NOP
        NOP
        NOP
        NOP
.ENDMACRO
```

Der Aufruf erfolgt im Programmtext mit:

```
...
Delay1
...
```


Unterprogramme

Unterprogrammaufruf

Der Aufruf eines Unterprogramms erfolgt durch spezielle Befehle, wie z.B.:

```
RCALL   NAME        ; Aufruf des Unterprogramms NAME
```

mit dem Namen des Unterprogramms als symbolische Adresse (LABEL). Diese Adresse steht als `Label` im Namensfeld des ersten auszuführenden Befehls des Unterprogramms:

```
NAME: LDI R16, 0xFF;
      ...
```


RCALL

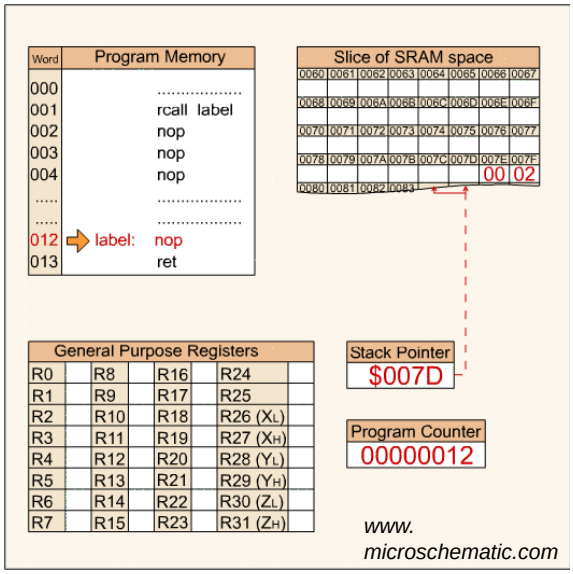
RCALL schreibt den aktuellen Programmzählerinhalt (PC) als Rücksprungadresse auf den aktuellen Stack und lädt den Befehlszähler mit der im Adressteil von RCALL stehenden Unterprogrammadresse.
Anschließend übernimmt das Unterprogramm die Programmsteuerung.

Der Rücksprung in das aufrufende Programm erfolgt durch spezielle Rücksprungbefehle, wie z.B.

```
RET          ; (RETurn)
```

6

RCALL

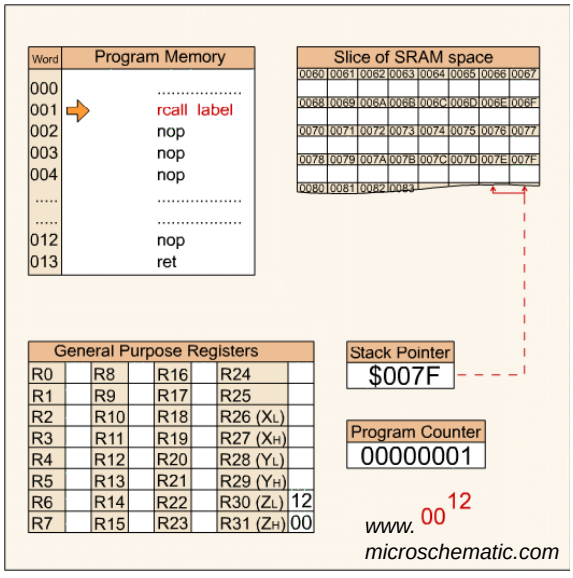


Direkter Aufruf eines Unterprogramm über ein Label

emg
DMM

6

ICALL



Indirekter Aufruf eines Unterprogramm über den Inhalt des Z-Registers:

Unterprogramm kann zur Laufzeit des Programms gewählt werden

emg
DMM

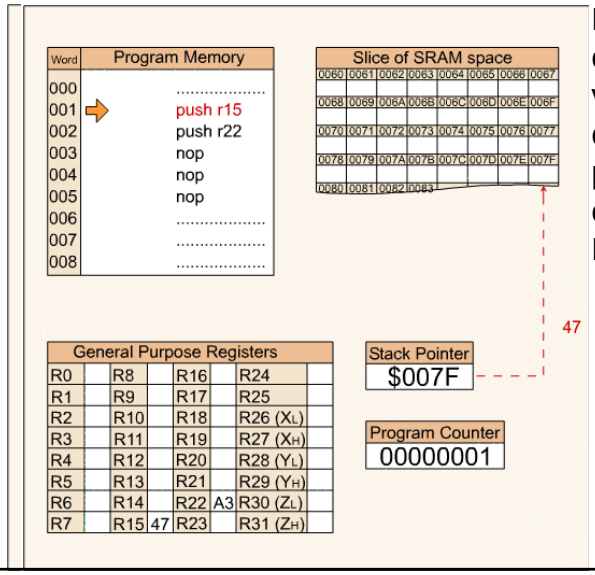
PUSH und POP

Für das Retten und Laden der allgemeinen Register stehen folgende **`Befehle`** zur Verfügung:

```
PUSH mem/reg ; Retten des Status- und der Arbeitsregister
               ; auf  STACK
POP  mem/reg ; Laden der Arbeitsregister aus  STACK
               ; auf  STACK
```

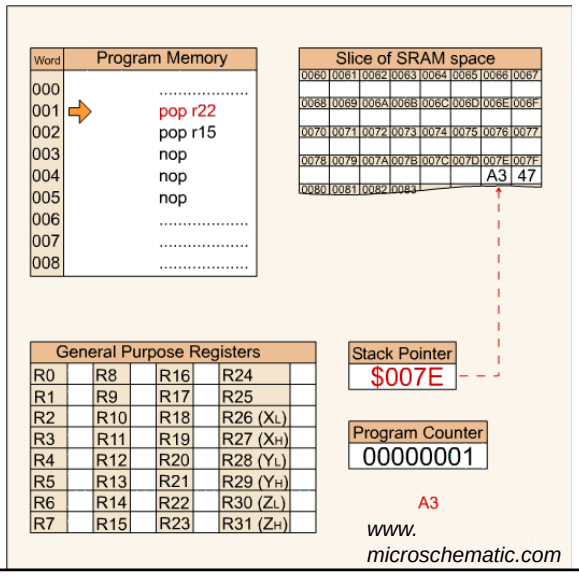
Das Retten und das Laden des Prozessorstatus kann sowohl im aufrufenden Oberprogramm als auch im Unterprogramm durchgeführt werden.

PUSH



Retten des Inhalts eines Registers vor dem Beginn eines Unter-programms oder einer Interrupt-Routine

POP



Zurückholen des Inhalts eines Registers nach dem Ende eines Unter-programms oder einer Interrupt-Routine

6

Parameter-Übergabe

emg
DMM