5

05. Assembler-Programmierung

Datenstrukturen des ATMega32

emg
DMM

5

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

5

Datenspeicher im ATMega32

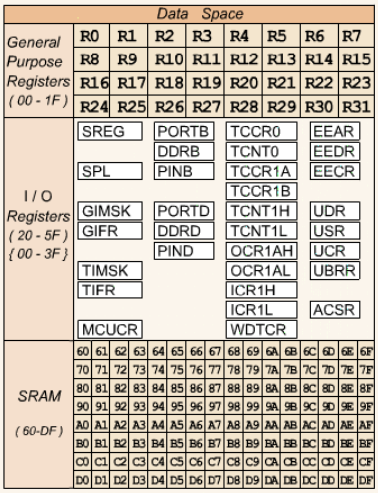
32 x 8 Bit

64 x 8 Bit

(LD Data , ST Data)
{ IN Port, OUT Port }

Accessing by different
Data Transfer Instructions

2048 Byte



Für den Datenspeicher gibt es einen zusammenhängenden Adressbereich mit Bereichen unterschiedlicher Bedeutung:

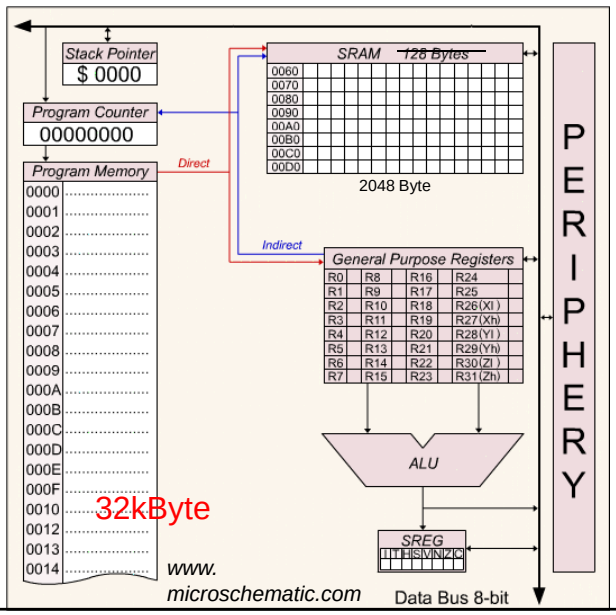
1. 32 allgemeine Register
2. 64 I/O Register
3. SRAM (2048 Byte)

www.
microschematic.com

emg
DMM

5

Adressierung im ATMega32



www.
microschematic.com

emg
DMM

5

32 Register im ATMega32 (8-Bit)

Datenregister zur freien Verfügung

R0 bis R15	Register zur freien Verfügung

Datenregister für spezielle Anwendungen

R16 - R23	frei definierbare 8-Bit Register zur freien Verfügung, wichtig für bestimmte Befehle
R24/R25	16-Bit Register zur freien Verfügung z.B. für 16-Bit-Zähler
R26/R27	X-Register (16-Bit)
R28/R29	Y-Register (16-Bit)
R30(ZL)/ R31(ZH)	Z-Register: 16Bit-Pointerregister für RAM-Speicher

emg
DMM

5

64 Ports im ATMega32

- Status-Register
- Stackpointer
- MCU-General Control Register
- Interrupt Mask Register, Flag Register
- Timer Timer Interrupt Register, Interrupt Flag Register
  Timer 0 (2 Register: Control-, Zählregister)
  Timer 1 (6 Register: Ctrl A/B, Zähl-,Vergleichsreg.)
  Watchdog Timer Control Register
- EEPROM Adress-, Data-, Control-Register
- Peripherie
  Serial Peripheral Control-, Status-, Data-Register
  UART Data-, Status-, Control-, Baudrate-Register
  Analog Comparator Control/Status-Register
- I/O Ports A-D: Data Direction, Status Register, Data

emg
DMM

Das Statusregister

I	T	H	S	V	N	Z	C

Befehle: High-Setzen: SEC Low-Setzen: CLC

		Gesetzt, wenn
Bit0	C Carry:	im Ergebnis Übertrag entsteht
Bit1	Z Zero:	als Ergebnis Null entsteht
Bit2	N Negative:	Ergebnis negativ/kleiner
Bit3	V Zweiter Komplement-Überlauf:	Überlauf
Bit4	S Sign:	Vorzeichen negativ, S=N XOR V
Bit5	H Halbübertrag:	Halbübertrag
Bit6	T Bitspeicher:	gespeichertes Bit = 1
Bit7	I Interrupt:	Interrupts erlaubt

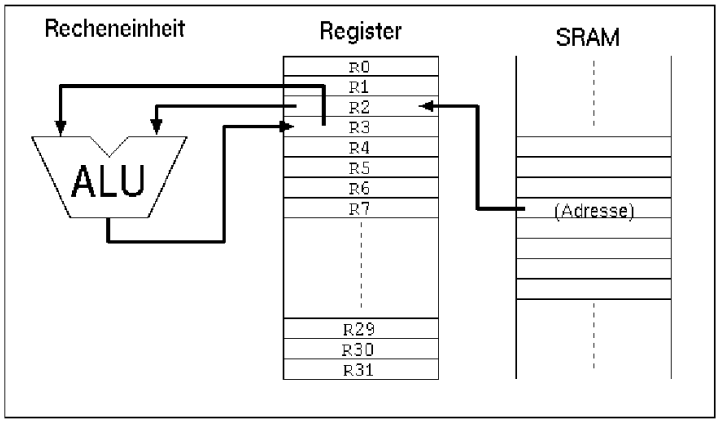
Die Bits im Statusregister

Bit	*Rechnen*	*Logik*	*Ver-gleich*	*Bits*	*Schieben*	*Sonst*
Z	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR Z, BSET Z, CLZ, SEZ, TST	ASR, LSL, LSR, ROL, ROR	CLR
C	ADD, ADC, ADIW, SUB, SUBI, SBC, SBCI, SBIW	COM, NEG	CP, CPC, CPI	BCLR C, BSET C, CLC, SEC	ASR, LSL, LSR, ROL, ROR	-
N	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR N, BSET N, CLN, SEN, TST	ASR, LSL, LSR, ROL, ROR	CLR
V	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR V, BSET V, CLV, SEV, TST	ASR, LSL, LSR, ROL, ROR	CLR
S	SBIW	-	-	BCLR S, BSET S, CLS, SES	-	-
H	ADD, ADC, SUB, SUBI, SBC, SBCI	NEG	CP, CPC, CPI	BCLR H, BSET H, CLH, SEH	-	-
T	-	-	-	BCLR T, BSET T, BST, CLT, SET	-	-
I	-	-	-	BCLR I, BSET I, CLI, SEI	-	RETI

Tabelle gibt an, welcher Befehl welches Bit ändert

5

Zugriff auf SRAM

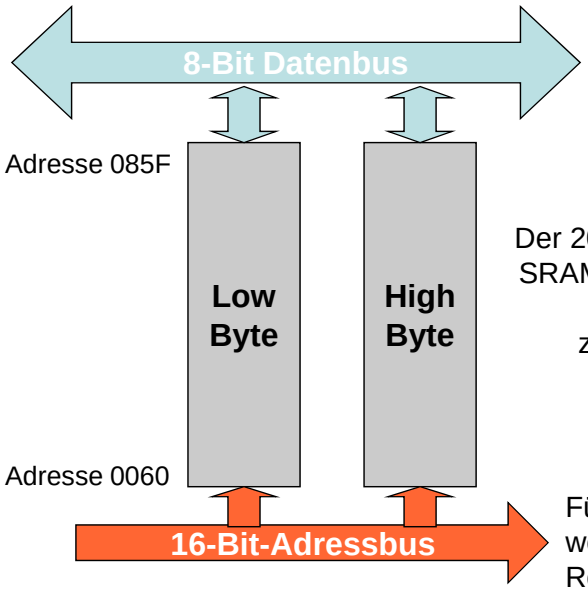


Werte aus dem SRAM müssen zur Verarbeitung in Register geholt und von dort zurück in SRAM geschrieben werden.

emg
DMM

5

Speicherorganisation des Mega32



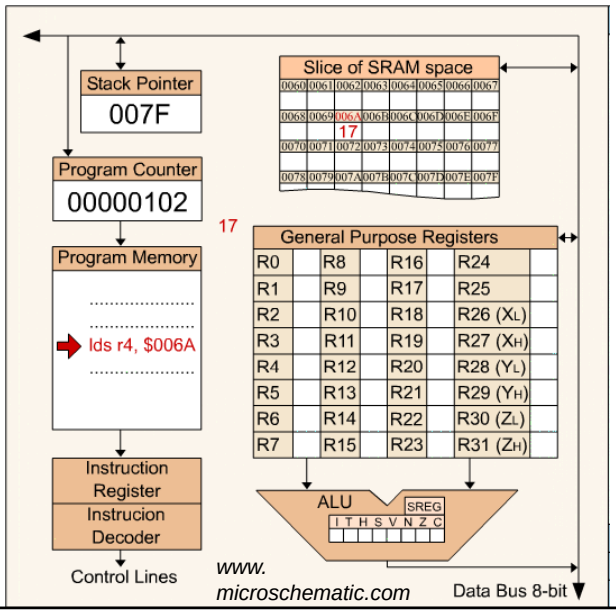
Der 2048 Byte große SRAM-Speicher wird Byteweise zugegriffen.

Für 16-Bit Adressen werden die X,Y,Z Register verwendet

emg
DMM

5

LDS /STS Direktzugriff auf SRAM

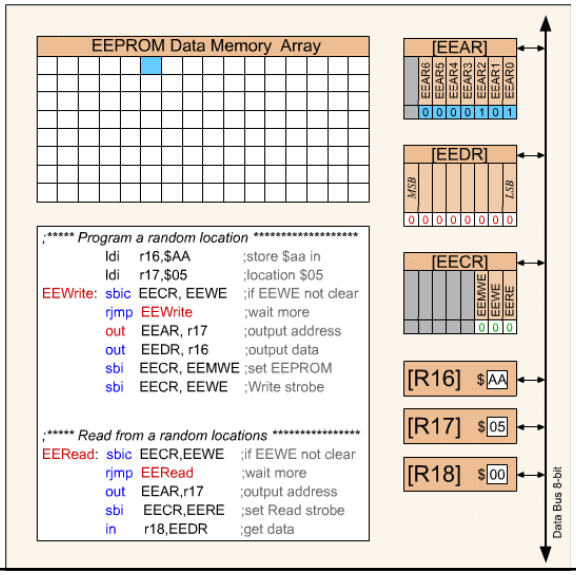


lds r4, $006A
Lade SRAM-Inhalt (8-Bit) an Adresse $006A in das Register R4

sts $006A, r4
Speichere R4 Register-Inhalt (8 Bit) im SRAM an Adresse $006A

5

EEPROM-Zugriff



Für den EEPROM (1 kByte) gibt es die drei Register

EEAR Adresse
EEDR Direction
EECR Control

auf die wie auf ein Peripheriegerät zugegriffen wird.

www.
microschematic.com

5

Adressierung

emg
DMM

5

Adressierungsarten im ATMega32

Datenspeicher-Zugriff
•Direkte Adressierung,

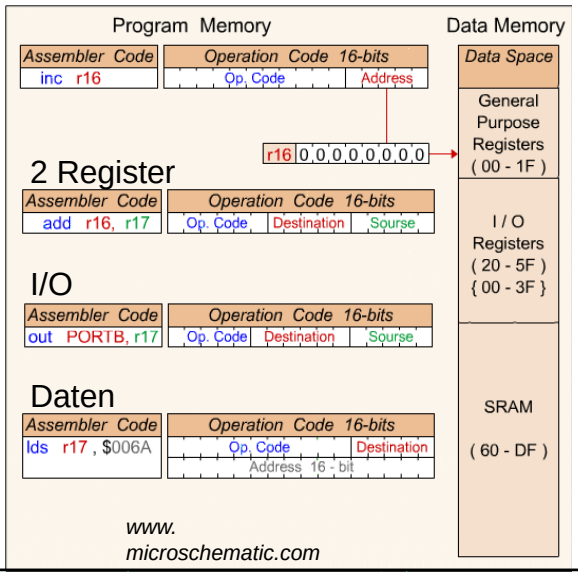
•Indirekte Adressierung,
•Indirekte Adressierung mit Displacement,

•Indirekte Adressierung mit Pre-decrement,
•Indirekte Adressierung mit Post-increment.

Programmspeicher-Zugriff
• Konstant, Relativ und Indirekt

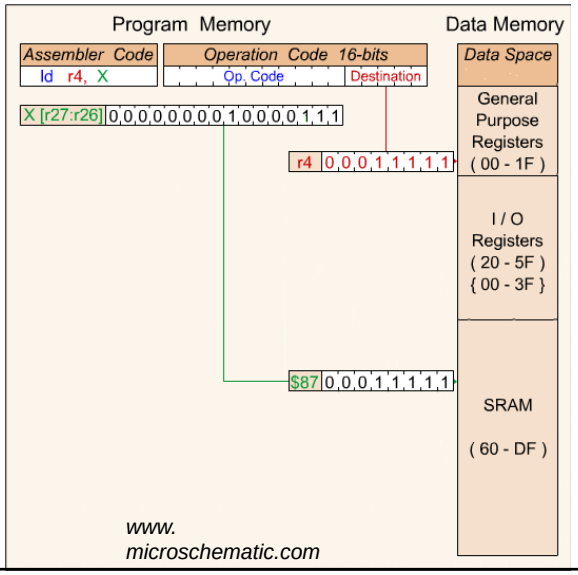
emg
DMM

Direkte Adressierung



Die direkte Adressierung verändert den Inhalt eines der Register im unteren Bereich des Speichers bis 0x5F, kann aber den gesamten Datenbereich adressieren.

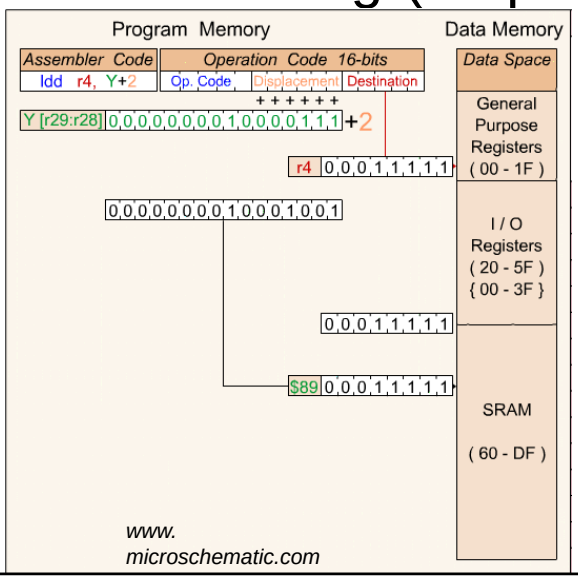
Indirekte Adressierung



Die indirekte Adressierung lädt den Inhalt einer SRAM-Zelle (16-Bit Adresse!) in eines der Register im unteren Bereich des Speichers bis 0x1F.

5

Indirekte Adressierung mit Verschiebung (Displacement)

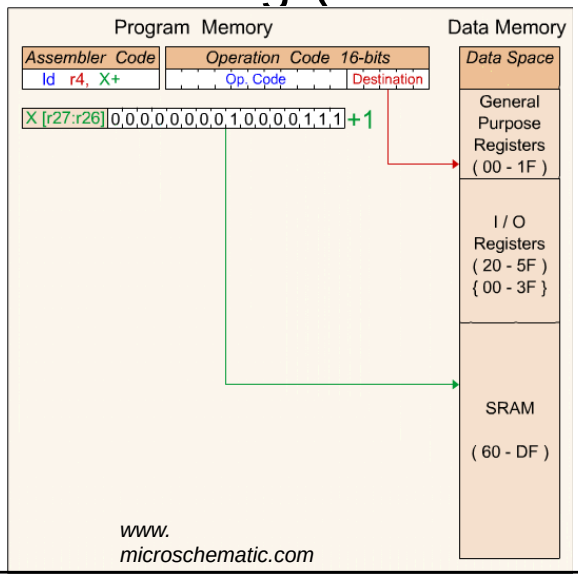


Die indirekte Adressierung mit Verschiebung erhöht die Adresse und lädt dann den Inhalt einer SRAM-Zelle (16-Bit Adresse!) in eines der Register im unteren Bereich des Speichers bis 0x1F.

emg
DMM

5

Indirekte Adressierung mit Erhöhung (Post-increment)

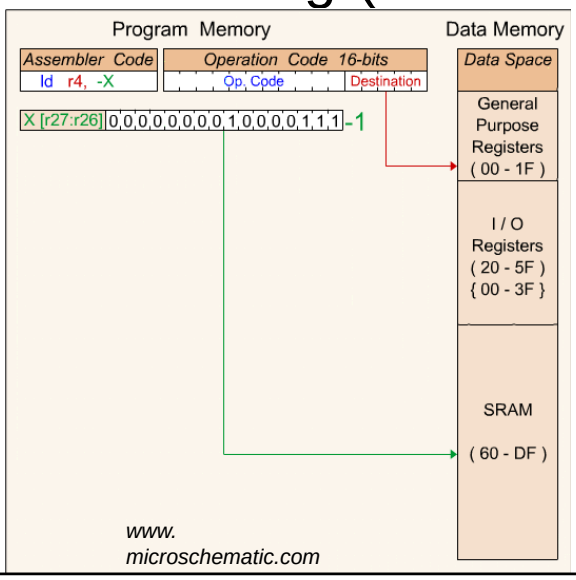


Die indirekte Adressierung mit Verschiebung lädt den Inhalt einer SRAM-Zelle (16-Bit Adresse!) in eines der Register im unteren Bereich des Speichers bis 0x1F und erhöht dann die Adresse für den nächsten Zugriff.

emg
DMM

5

Indirekte Adressierung mit Reduzierung (Pre-decrement)



Die indirekte Adressierung mit Reduzierung senkt die Adresse und lädt dann den Inhalt einer SRAM-Zelle (16-Bit Adresse!) in eines der Register im unteren Bereich des Speichers bis 0x1F.

emg
DMM

5

Programmspeicher Adressierung

Constant memory addressing

lpm	load program memory	lädt je nach LSB des Z-Registers (0/1) Low-Byte oder High-Byte des Programmspeichers in Register R0

Relative memory adressing

rjmp *label*	relative jump	lädt den Programmzähler mit neuer Programm-Adresse (= label) und springt dorthin (wie goto)

Indirect memory adressing

ijmp	indirect jump	lädt den Programmzähler mit neuer Programm-Adresse aus dem Z-Register und springt dorthin

emg
DMM

Assembler

Elemente einer Assemblersprache

Befehle sind mnemonische Kürzel für Operationen.

Operanden sind Registernamen, Variablen, Konstanten oder Zahlen, mit denen Operationen ausgeführt werden.

Label sind symbolische Adressmarken. z.B. main:

Deklarationen werden zur Vereinbarung von Variablen und Konstanten verwendet.

5

Elemente einer Assemblersprache

Operatoren ermöglichen u.a. Rechnungen mit symbolischen Adressen: `+`, `-`, `AND`, `NOT`, `OFFSET`. Diese Rechnungen werden vom Assembler bei der Übersetzung vom Quellcode in den ausführbaren Objektcode vorgenommen.

Direktiven sind Kommandos, die der Programmierer dem Assembler zur Organisation des Speichers gibt.

Diese werden nicht in den Objektcode für den Mikroprozessor übersetzt, sondern beim Übersetzungsvorgang berücksichtigt.

emg
DMM

5

Befehlsarten beim ATMega32

28 Arithmetik- und Logik-Befehle

`ADD,SUB,MUL,FMUL,AND,OR,CLR..`

38 Verzweigungsbefehle

`JMP,CALL,RET,CP,SBRC,BREQ,..`

39 Datentransfer-Befehle

`MOV,LDI,ST,LPM,IN,OUT,PUSH..`

22 Bit- und Bit-test-Befehle

`LSL,ROL,BSET,SBI,BST,SEC,..`

4 MCU Control-Befehle

`BREAK, NOP, SLEEP, WDR`

emg
DMM

Befehle in AVR-Controllern

www.microschematic.com

ADC	BRMI	CLZ	LD Z+	ROL	ST X	Instruction set
ADD	BRNE	COM	LDD Y	ROR	ST -X	ADC
ADIW	BRPL	CP	LDD Z	SBC	ST X+	Add with Carry two Registers
AND	BRSH	CPC	LDI	SBCI	ST Y	
ANDI	BRTC	CPI	LDS	SBI	ST -Y	Operation: Rd←Rd+Rr+C
ASR	BRTS	CPSE	LPM	SBIC	ST Y+	Syntax: adc Rd,Rr
BCLR	BRVC	DEC	LSL	SBIS	ST Z	
BLD	BRVS	EOR	LSR	SBIW	ST -Z	Operands: 0 ≤ d ≤ 31
BRBC	BSET	ICALL	MOV	SBR	ST Z+	0 ≤ r ≤ 31
BRBS	BST	IJMP	NEG	SBRC	STD Y	Flags: H,S,V,N,Z,C
BRCC	CBI	IN	NOP	SBRS	STD Z	
BRCS	CBR	INC	OR	SEC	STS	
BREQ	CLC	LD X	ORI	SEH	SUB	
BRGE	CLH	LD -X	OUT	SEI	SUBI	(Overlap onto boxes)
BRHC	CLI	LD X+	POP	SEN	SWAP	Arithmetic and Logic
BRHS	CLN	LD Y	PUSH	SER	TST	Bit and Test
BRID	CLR	LD -Y	RCALL	SES	SLEEP	Branch
BRIE	CLS	LD Y+	RET	SET		Data Transfer
BRLO	CLT	LD Z	RETI	SEV	WDR	Architecture / Index
BRLT	CLV	LD -Z	RJMP	SEZ		Expressions / Directives

Schreibweise von Befehlen

```
Label: OPCode ZielOP, QuellOP; Kommentar
```

Beispiel für Assemblerprogramm *test1.asm*:

```
.INCLUDE „m32def.inc“;  Assembler Directive lädt Konstanten
.DEF mp = R16           ;Assembler Directive: mp ist Register16
RJMP MAIN               ;relative jmp zu Label main
MAIN: LDI mp,0b11111111; Load immediate 255 nach mp
      OUT DDRB,mp       ; Macht Port B zu 8-Bit Ausgang
LOOP:
      LDI mp,0x00       ;Load immediate 0 nach mp
      OUT PORTB,mp      ; Setzt Port B auf Wert in mp
      LDI mp,0xFF       ;Load immediate 255 nach mp
      OUT PORTB,mp      ; Setzt Port B auf Wert in mp
      RJMP LOOP         ; Rücksprung zu LOOP - endlos
```

Assemblertest.asm
Assemblertest1.asm

Einbindung von Assembler in C

Über einen integrierten ***In-Line-Assembler*** kann mit den meisten C-Compilern bei der Übersetzung auch reiner Assembler-Maschinensprachecode eingebunden werden. Die der Unterroutine übergebenen Variablen können dabei mit ihren Bezeichnungen verwendet werden.

Es ist allerdings bei Registeroperationen darauf zu achten, dass die Register des Prozessors beim Einsprung in die Unterroutine Werte beinhalten können, die nicht denen der vorhergehenden Unterroutinendurchläufe entsprechen.

5

In-line Assembler für ATMega32

```
int ivar1, ovar1;          /* Variablen Deklaration */
.
asm volatile (
„Assembler-Befehl1"     „\n\t" /* Template-Liste */
„Assembler-Befehl2 %0" „\n\t"
.                          „\n\t"
.                          „\n\t"
„Assembler-Befehl99"    „\n\t"
: „=a" (ovar1),...         /* Output (=)Operanden */
: „d" (ivar1),...          /* Input Operanden Liste */
: „r2", „r3"               /* Clobber Liste: Welche */
constraints                /* Register ändern sich?*/
)
```

emg
DMM

5

Contraints für Inline-Assembler

Tabelle: Constraints und ihre Bedeutung

Constraint	Register	Wertebereich
a	einfache obere Register	r16...r23
b	Pointer-Register	y, z
d	obere Register	r16...r31
e	Pointer-Register	x, y, z
l	untere Register	r0...r15
q	Stack-Pointer	SPH:SPL
r	ein Register	r0...r31
t	Scratch-Register	r0
w	obere Register-Paare	r24, r26, r28, r30
x	Pointer-Register X	x (r27:r26)
y	Pointer-Register Y	y (r29:r28)
z	Pointer-Register Z	z (r31:r30)
0...9	Identisch mit dem angegebenen Operanden Wird verwendet, wenn ein Operand sowohl als Input als auch als Output dient, um sich auf diesen Operanden zu beziehen	

Constraint	Konstante	Wertebereich
G	Floatingpoint-Konstante	0.0
i	Konstante	
I	positive 6-Bit-Konstante	0...63
J	negative 6-Bit Konstante	-63...0
M	8-Bit Konstante	0...255

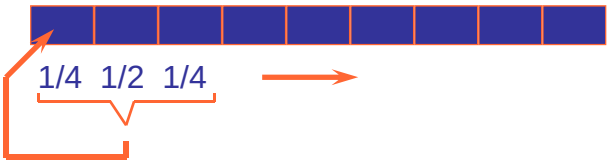
Constraint	Memory	Wertebereich
m	Memory	

emg
DMM

5

Beispielprogramm

Glättungsroutine in C und Assembler



Feld mit Mess-daten

```
/* Glättungsroutine in C geschrieben: */

byte glaettenc (float E1Wert,E2Wert,E3Wert)
{
  return (byte)(0.25*E1Wert + 0.5*E2Wert + 0.25*E3Wert);
}
```

Diese Funktion muß für jede Verschiebung aufgerufen werden.

```
mw = glaettenc (2, 4, 6);
```

emg
DMM

5

Glättungsroutine in Assembler

```
uint8_t glaettenasm(uint8_t E1Wert,uint8_t E2Wert,uint8_t E3Wert)
{ uint8_t ergebnis;
asm volatile (              /* Berechnung: (E2Wert*2+E1Wert+E3Wert)/4 */
"mov R16,%1;" "\n\t"        /* E1Wert nach R16   */
"add R16,%2;" "\n\t"        /* + E2Wert          */
"add R16,%2;" "\n\t"        /* + E2Wert          */
"add R16,%3;" "\n\t"        /* + E3Wert          */
"lsr R16"     "\n\t"        /* R16 / 2           */
"lsr R16"     "\n\t"        /* R16 / 2           */
"mov %0,R16"  "\n\t"        /* R16 nach Ergebnis */
:"=d" (ergebnis)            /* 1. Output-Operand(=) ist ergebnis in R16-R31 */
:"d" (E1Wert),"d" (E2Wert),"d" (E3Wert) /* 3 Input-Operanden in R16-R31*/
:"r16"                      /* Einziges verändertes Register ist R16*/
);
return ergebnis;
};
```

emg
DMM