9. AKTUELLE RECHNER-ARCHITEKTUREN

9.1 Mikrocontroller

Mikrocontroller sind ganze Mikrorechner auf einem Chip, d.h. sie haben neben einem CPU und (Festwert-)Speicher auch spezielle Peripherie-Schnittstellen für die Ein-/Ausgabe zu anwendungsspezifischer Elektronik wie Sensoren und Aktoren sowie oft auch weitere Hardwareeinheiten integriert.

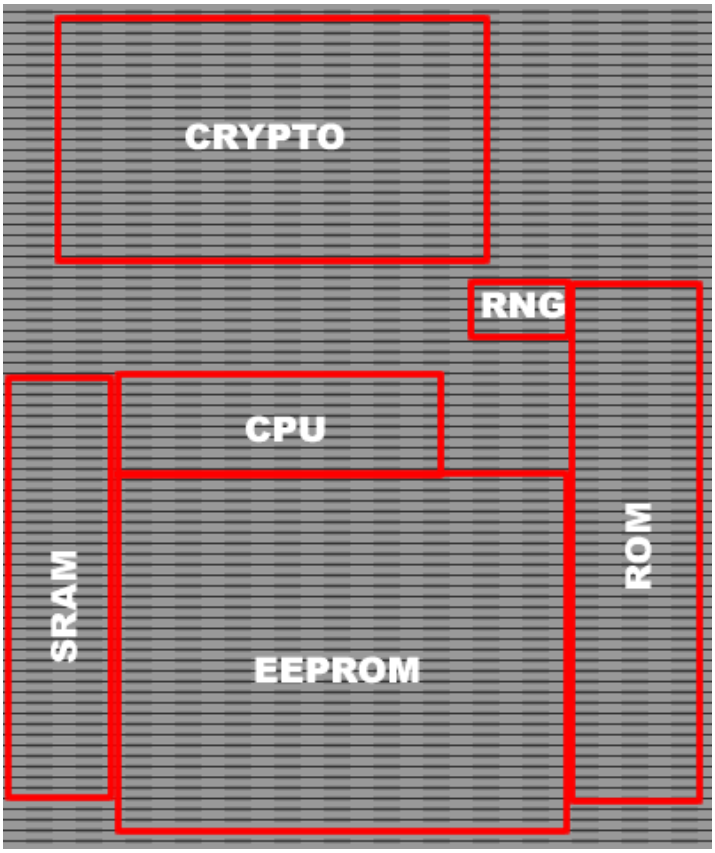
Das Ziel bei Mikrocontrollern ist, mit möglichst wenig externer Hardware und geringen Kosten (und Platz- und Energiebedarf) voll funktionsfähige **eingebettete Systeme** aufbauen zu können.

Meist sind die Schnittstellen zur Peripherie so **flexibel konfigurierba**r, dass der Mikrocontroller für eine (möglichst große) Vielzahl verschiedener Anwendungen mit unterschiedlichen Anforderungen eingesetzt werden kann.

Typische Anwendungsfelder von Mikrocontrollern sind eingebettete Systeme wie:

- Computerperipherie, Automobilelektronik
- mobile Geräte wie Handys, (Digital-)Kameras
- „intelligente“ Haushaltsgeräte (Waschmaschine, …)
- Unterhaltungselektronik (DVD-Player, …)
- Roboter
- Chipkarten
- …

Für Anwendungen mit sehr großen Stückzahlen werden auch anwendungsspezifische Mikrocontroller entwickelt.



Floorplan eines Mikrocontrollers einer Chipkarte
(mit Spezialeinheiten für Ver-/Entschlüsselung CRYPTO
und Zufallszahlenerzeugung RNG)

Die Entwicklung einer Mikrocontroller-Anwendung geschieht typischerweise in folgenden Schritten:

- Auswahl des Mikrocontrollers
- Entwurf und Aufbau der Schaltung, in die der Mikrocontroller eingebettet ist
- Entwurf der Software
- Test der Software mit einer Simulation (der Anwendung)
- Herunterladen des Programms in den (Flash-) Speicher des Mikrocontrollers
- Herunterladen der Parameter in den nichtflüchtigen EEPROM-Bereich
- Test der Software im Zielsystem

Typisches Datenblattübersicht für einen Mikrocontroller

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 16K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 1K Byte Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega16L
  - 4.5 - 5.5V for ATmega16
- Speed Grades
  - 0 - 8 MHz for ATmega16L
  - 0 - 16 MHz for ATmega16
- Power Consumption @ 1 MHz, 3V, and 25°C for ATmega16L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

8-bit AVR® Microcontroller with 16K Bytes In-System Programmable Flash

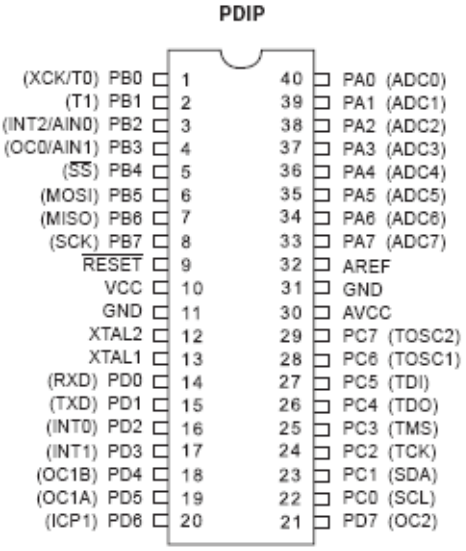
ATmega16
ATmega16L

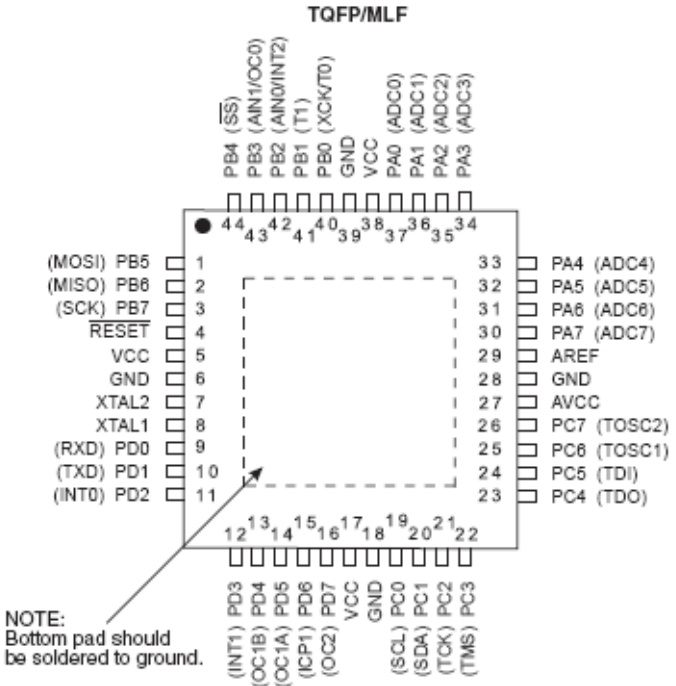
2466L–AVR–06/05

Typische Gehäuseformen und Anschlussbelegung

Pin Configurations

Figure 1. Pinout ATmega16





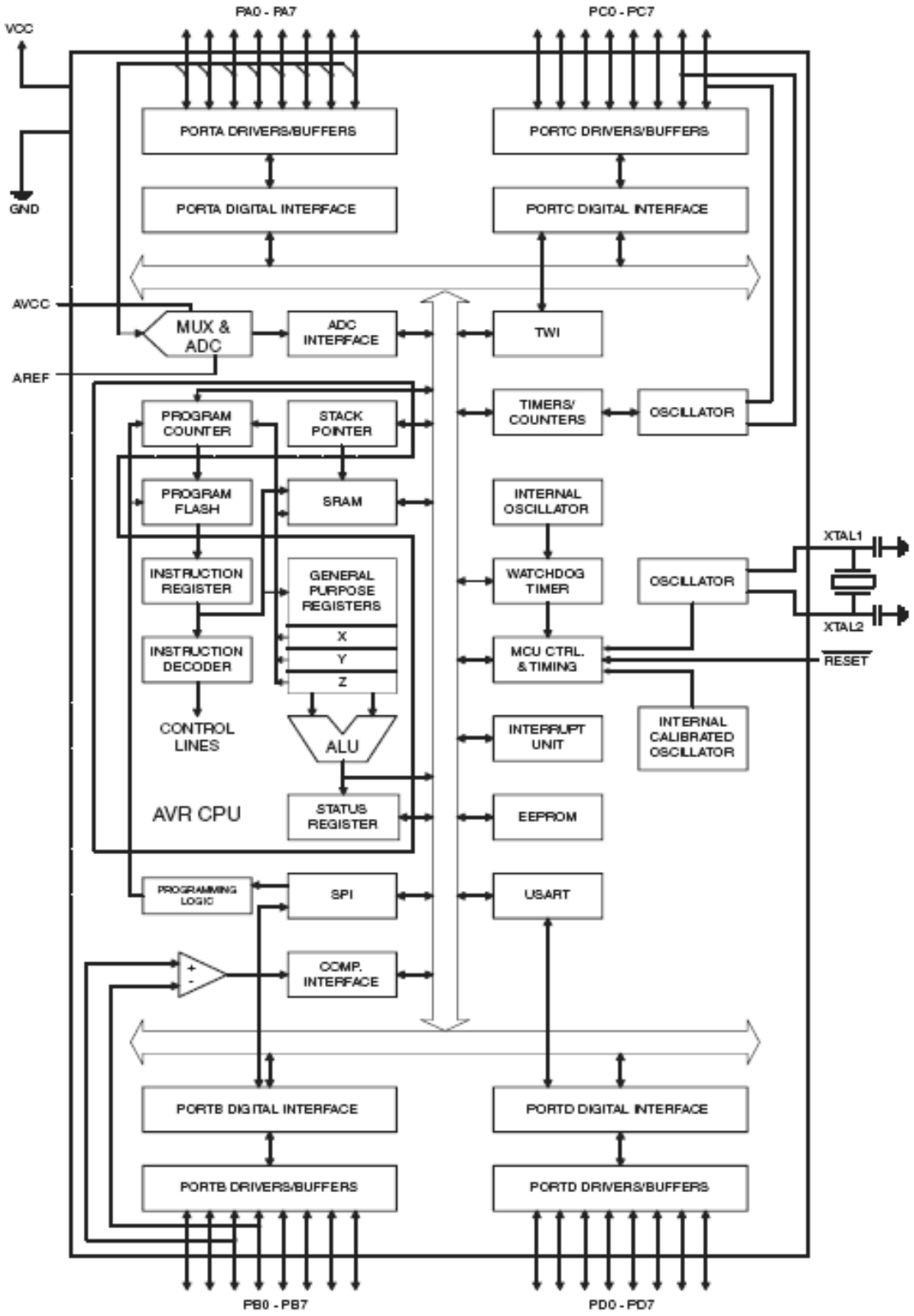
Disclaimer

Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

9.2 Mikrocontroller ATmega16

9.2.1 Übersicht

Der ATmega16 ist Mitglied der 8 Bit-AVR-Mikrocontroller-Familie der Firma *Atmel*.



Eigenschaften des ATmega16

Prozessorkern und Speicher

- 32 Universalregister mit Daten-Wortlänge **8 Bit**
- 131 Befehle mit 1 oder 2 **16 Bit**-Worten (RISC-orientierter Befehlssatz)
- bis zu 16 MHz Takt (16 MIPS Million Instructions per Second durch *Fetch-during-execute*-Befehlsausführung)
- Harvard-Architektur:
  16 KByte Flash ROM als Programmspeicher
  1 KByte SRAM als separater Datenspeicher
- 512 Byte EEPROM zusätzlich für Nur-Lese-Daten (für nicht-flüchtige Parameter; mit eigenem Adressbereich)
- JTAG-Interface für Programmladen, Test und Debugging
- Power-Save-Betriebsarten

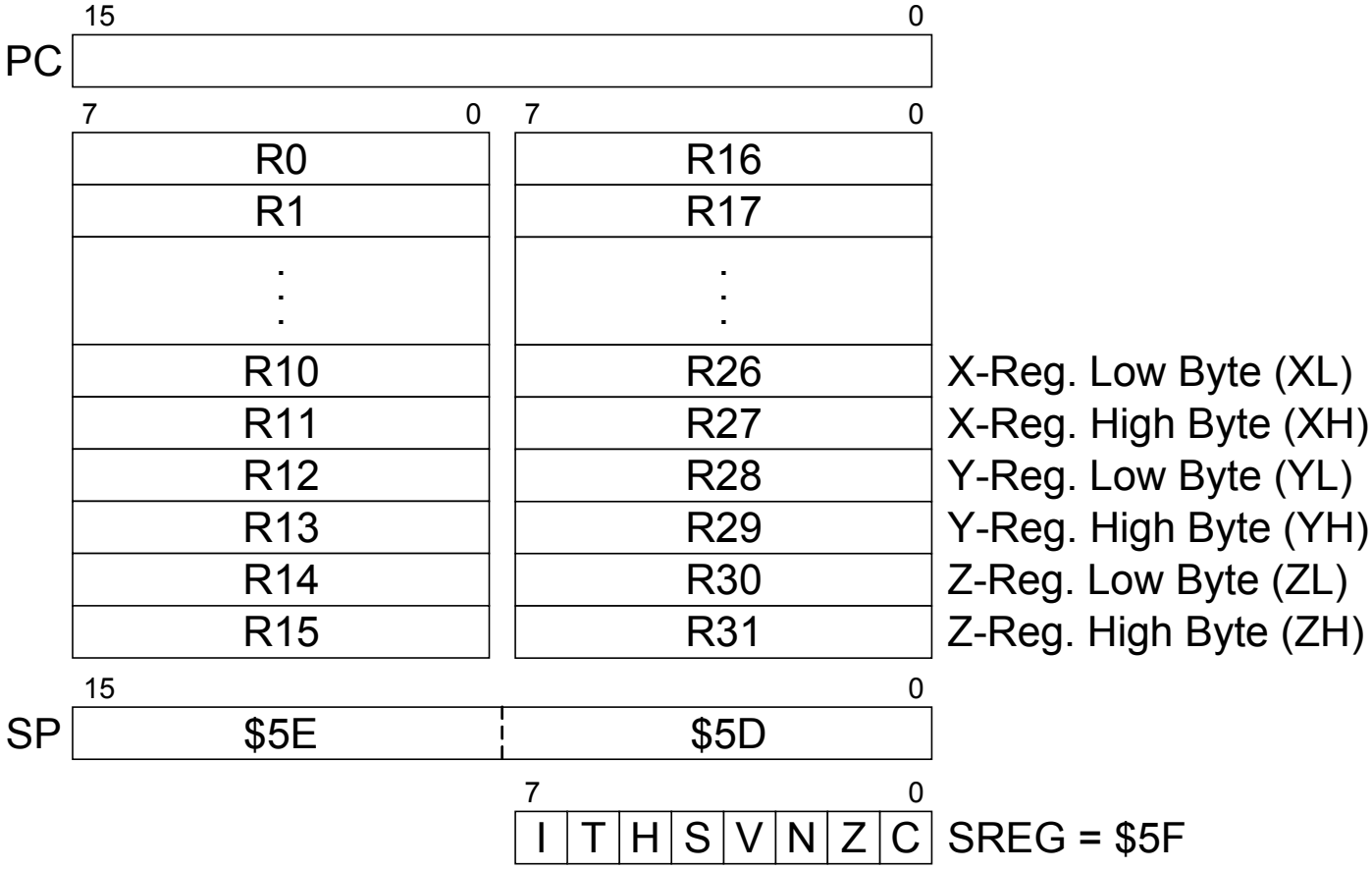
Peripherie (konfigurierbar)

- 4 · 8-Bit bidirektionale Parallel-Ports (A bis D)
- 2 · 8-Bit Timer/Counter
- 1 · 16-Bit Timer/Counter
- Echtzeit-Timer
- Watchdog-Timer
- 8 ADC-Kanäle (10 Bit Analog Digital Converter)
- Analog-Komparator
- 4 PWM (Pulse Width Modulation)-Ausgabekanäle
- byteorientierter 2-Draht Multi-Master-Bus (TWI)
- programmierbare asynchrone serielle Schnittstelle (USART)
- synchrone serielle Schnittstelle (SPI)
- externe und interne Interrupts, Reset

Links: www.atmel.com
www.avr-asm-tutorial.net

9.2.2 Prozessorkern (CPU-Core)

Programmiermodell



Programmiermodell wie HATmega16, aber mit 8-Bit-Statusregister SREG

Die Flags C, Z, N, V, S und H werden durch die Hardware je nach Befehl auch automatisch abhängig vom Ergebnis einer Datenmanipulationsoperation gesetzt, aber **nicht** bei einer Datentransferoperation.

Alle Statusbits können auch per Befehl gesetzt oder gelöscht werden.

Beachten: Der Stackpointer SP sowie das Statusregister liegen im Datenspeicher, wären also auch wie Daten zugreifbar.

Flags (C bis H werden automatisch per Hardware gesetzt)

C-Flag (**C**arry):
Übertrag bei Addition oder ‘Borgen‘ bei Subtraktion, Kopie von Bit 15 (MSB) bei Multiplikation.

Z-Flag (**Z**ero):
Wird gesetzt, wenn Ergebnis der letzten arithmetisch/logischen Operation Null ist, nicht aber bei Datentransferbefehlen etc.

N-Flag (**N**egative):
Kopie von Bit 7 des Ergebnisses der letzten arithmetisch/logischen Operation (Vorzeichenbit), nicht aber bei Datentransferbefehlen etc.

V-Flag (o**V**erflow):
Überlauf des gültigen Zahlenbereichs (Zweierkomplement).

S-Flag (**S**ign):
Exklusives ODER von Negativ- und Overflow-Flag

$$S = N \oplus V$$

H-Flag (**H**alf carry):
Übertrag von Bit 3 auf Bit 4 (wird für BCD-Arithmetik benötigt, da zwei „gepackte BCD-Zahl“ in einem Byte).

T-Flag (Bit Copy Storage):
Steht zur freien Verfügung, z.B. als Bit-Zwischenspeicher. Es wird von BLD- und BST-Befehlen als Quelle bzw. Ziel verwendet; z. B. um andere Status-Bits zwischenzuspeichern.

I-Flag (Global **I**nterrupt Enable):
Muss gesetzt sein, um Interrupts generell zu erlauben, bzw. sperrt Interrupts, wenn es gelöscht ist (siehe Interrupts).

9.2.3 Befehlssatz

Vollständige Befehlsliste des ATmega16

Mnemo.	Oper.	Description	Operation	Flags	Clock
Arithmetic and Logic Instructions					
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND	Rd ← Rd ∧ Rr	Z,N,V,S	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd ∧ K	Z,N,V,S	1
OR	Rd, Rr	Logical OR	Rd ← Rd ∨ Rr	Z,N,V,S	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd ∨ K	Z,N,V,S	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,S,H	1
SBR	Rd, K	Set Bit(s) in Register	Rd ← Rd ∨ K	Z,N,V,S	1
CBR	Rd, K	Clear Bit(s) in Register	Rd ← Rd ∧ ($FFh - K)	Z,N,V,S	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1
TST	Rd	Test for Zero or Minus	Rd ← Rd ∧ Rd	Z,N,V,S	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd × Rr (UU)	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd × Rr (SS)	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd × Rr (SU)	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd × Rr) << 1 (UU)	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd × Rr) << 1 (SS)	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd × Rr) << 1 (SU)	Z,C	2
Branch Instructions					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2
JMP	k	Jump	PC ← k	None	3
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3/4
ICALL		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	3/4
CALL	k	Call Subroutine	PC ← k	None	4/5
RET		Subroutine Return	PC ← STACK	None	4/5
RETI		Interrupt Return	PC ← STACK	I	4/5
CPSE	Rd, Rr	Compare, Skip if Equal if (Rd = Rr)	If (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd, Rr	Compare	Rd - Rr	Z,C,N,V,S,H	1
CPC	Rd, Rr	Compare with Carry	Rd - Rr -C	Z,C,N,V,S,H	1
CPI	Rd, K	Compare with Immediate	Rd - K	Z,C,N,V,S,H	1
SBRC	Rr, b	Skip next inst. if Bit in Register Cleared	if (Rr(b) = 0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip next inst. if Bit in Register Set	if (Rr(b) = 1) PC ← PC + 2 or 3	None	1/2/3
SBIC	A, b	Skip n. inst. if Bit in I/O Register Cleared	if (I/O(A,b) = 0) PC ← PC + 2 or 3	None	1/2/3
SBIS	A, b	Skip next inst. if Bit in I/O Register Set	if (I/O(A,b) = 1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC ← PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC ← PC+k + 1	None	1/2

BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V = 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than, Signed	if (N ⊕ V = 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
Data Transfer Instructions					
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LDS	Rd, k	Load Direct from data space	Rd ← (k)	None	2
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X), X ← X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
STS	k, Rr	Store Direct to data space	(k) ← Rd	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Increment	Rd ← (Z), Z ← Z + 1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1
OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2

Bit and Bit-test Instructions					
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n),Rd(0) ← 0,C ← Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1),Rd(7) ← 0,C ← Rd(0)	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) ← C,Rd(n+1) ← Rd(n),C ← Rd(7)	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	Rd(7) ← C,Rd(n) ← Rd(n+1),C ← Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n = 0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ↔ Rd(7..4)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	2
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	2
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Two's Complement Overflow	V ← 1	V	1
CLV		Clear Two's Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU Control Instructions					
BREAK		Break	Stop mode	None	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

Rd: Destination (and Source) Register in the Register File
Rr: Source Register in the Register File
R: Result after instruction is executed
K: Constant data
k: Constant address
b: Bit in the Register File or I/O Register (3-bit)
s: Bit in the Status Register (3-bit)
X,Y,Z: Indirect Address Register
(X=R27:R26, Y=R29:R28 and Z=R31:R30)
A: I/O location address
q: Displacement for direct addressing (6-bit)

Typische Beschreibung eines Befehls

CP – Compare

Description:

This instruction performs a compare between two registers Rd and Rr. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr

	Syntax:	Operands:	Program Counter:
(i)	CP Rd,Rr	$0 \le d \le 31, 0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0001	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
          cp    r4,r19    ; Compare r4 with r19
          brne  noteq     ; Branch if r4 <> r19
          ...
   noteq: nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

9.2.4 Speichermodell (Data Memory Map)

Beim ATmega16 liegen Daten- (SRAM, 1 KByte) und Programmspeicher (Flash-ROM, 16 KByte) sowie ein nichtflüchtiger EEPROM-Speicher (512 Byte) als drei separate Einheiten on-chip vor. Ein externer Speicher ist nicht vorgesehen.

Der Zugriff auf die drei Speichertypen geschieht für Instruktionen (Programmspeicher) implizit und für Daten explizit über unterschiedliche Befehle (SRAM, Flash) bzw. spezielle Register (EEPROM):

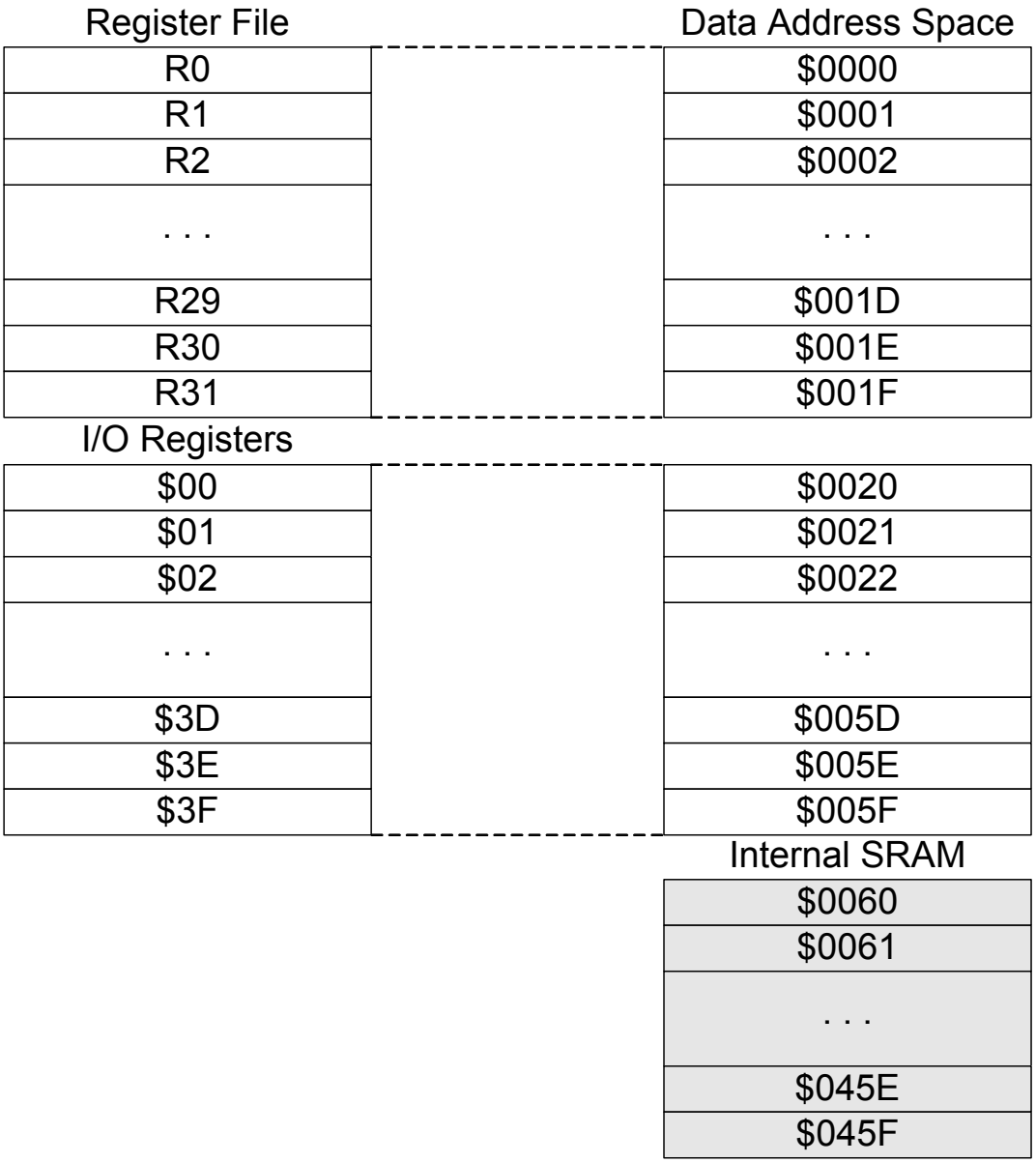
```
SRAM:    LDS   R1, $0060
         LD    R1, X+

Flash:   LPM
```

Der EEPROM-Speicher ist wie der Datenspeicher byteorganisiert, nimmt aber eine Sonderrolle ein, weil er nur ca. 100.000 mal beschreibbar ist und das Schreiben wesentlich länger dauert als das Lesen. Deshalb ist er nicht direkt per Programm schreibbar, sondern nur über spezielle Register (EEAR, EECR, EEDR).

Der EEPROM-Speicher ist aber bei der Programmierung des Mikrocontrollers wie der Flash-Speicher direkt vom Programmiersystem aus ladbar.

```
Deklaration mit:   .ESEG    …
                   .DB, .DW …
                   .BYTE    …
```

Arbeitsspeicher



Die 32 Prozessor-Register R0 bis R31 sind auch unter den Speicheradressen $0000 bis $001F zugreifbar.

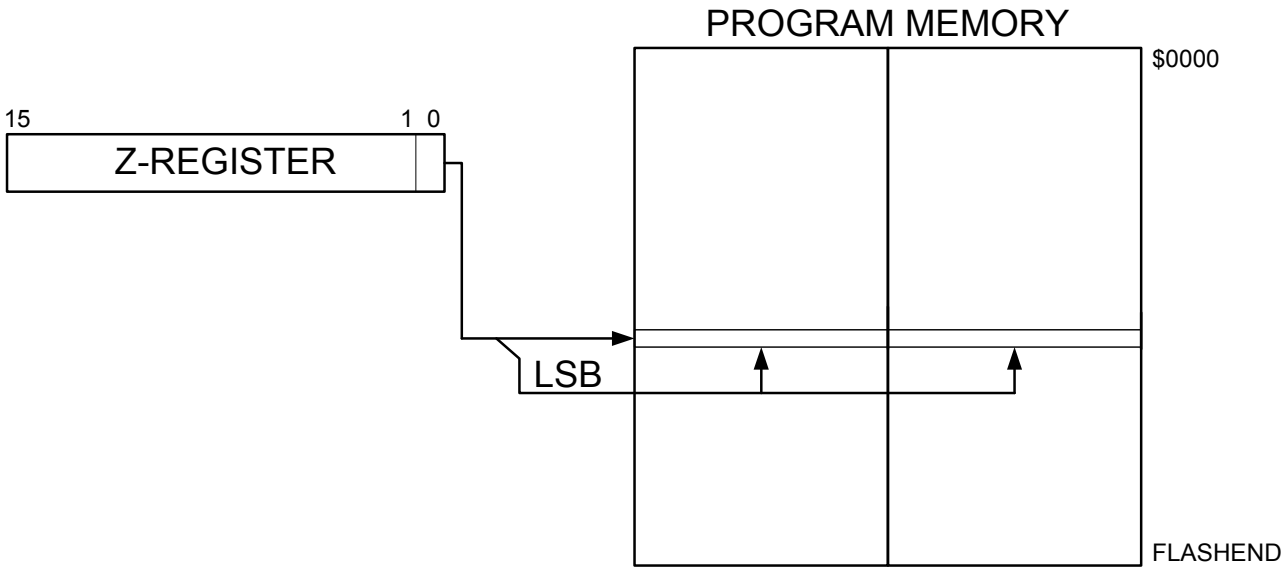
Die 64 E/A-Register sind durch IN- bzw- OUT-Befehle über die Ein-/Ausgabeadressen $00 bis $3F zugreifbar (vgl. „*separated I/O*“) oder **alternativ** auch durch normale Speichertransferoperationen unter den Speicheradressen $0020 bis $005F (vgl. „*memory mapped I/O*“, Achtung: Offset von $20).

Das interne SRAM (1 KByte) ist der Arbeitsspeicher und belegt die Adressen $0060 bis $045F.

9.2.5 Adressierungsarten

Zusätzlich zum HATmega16 erlaubt der reale ATmega16 noch die Adressierung des Programmspeichers und E/A-Adressierung.

Programmspeicher-Adressierung



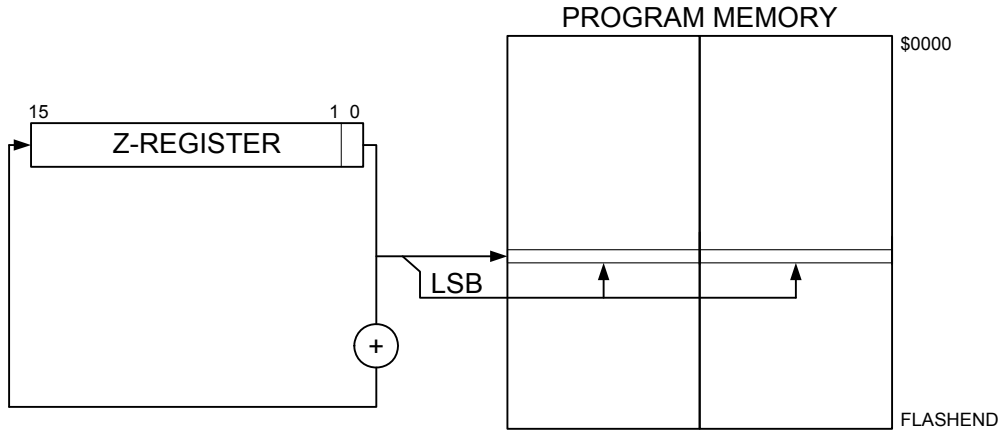
Achtung: Der indirekte Zugriff auf Daten im Programmspeicher erfolgt byteweise und ist nur mit dem Z-Register möglich.

Wortadresse in Bits 1 bis 15,
LSB = 0 wählt Low Byte aus,
LSB = 1 High Byte

LPM R15, Z ; R15 ← PROGMEM (Z)

LPM ; R0 ← PROGMEM (Z)
(0-Komponentenvariante mit R0, Z-Register implizit als Zeiger)

Programmspeicher-Adressierung mit Post-Inkrement

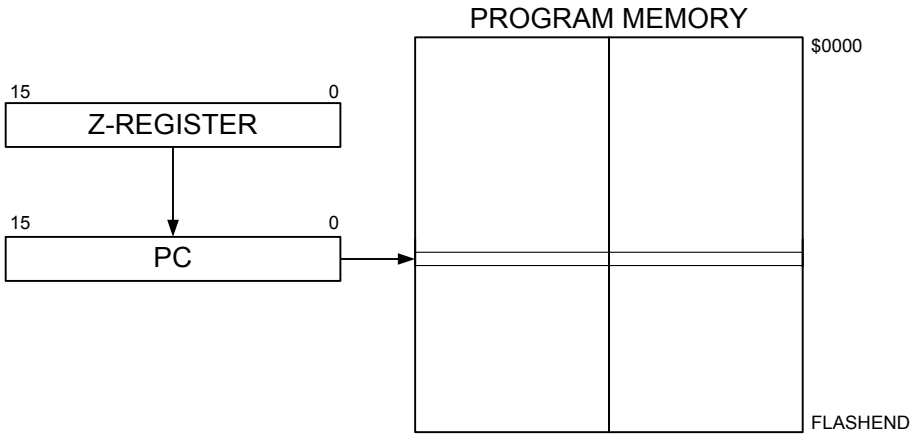


LPM R15, Z+ **; R15 ← PROGMEM (Z)**
; Z ← Z+1

Nützlich für Adressierung von Tabellen im Programmspeicher.

Auch zusammen mit Store-Befehl SPM für Programmspeicher anwendbar (Achtung: Zum Schreiben ist vorheriges Löschen einer Page in Flash-ROM erforderlich).

Indirekte Programm-Adressierung (Nur über Z-Register möglich!)

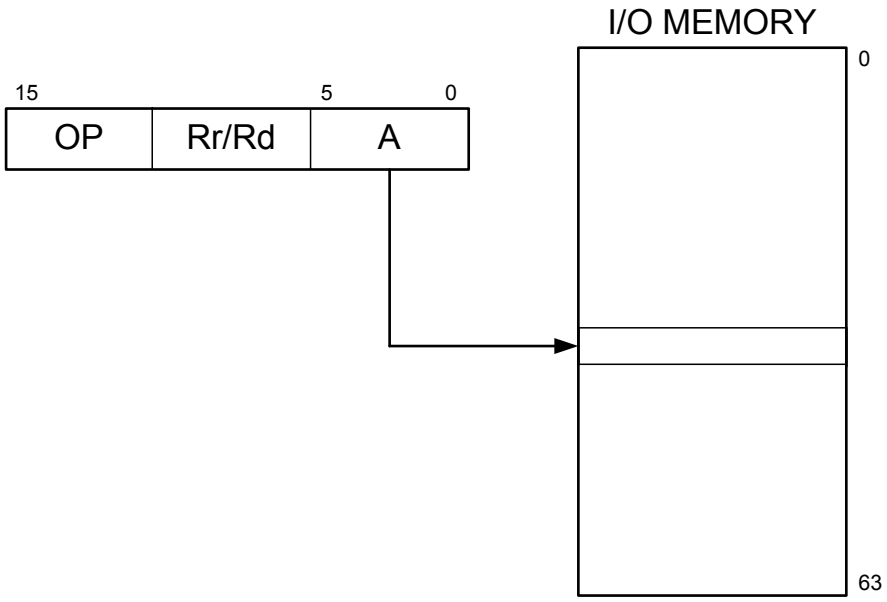


IJMP **; PC ← Z**

ICALL **; STACK ← PC+1**
; PC ← Z
; SP ← SP-2

Sprungziel bzw. Unterprogrammaufruf können also noch zur Laufzeit modifiziert werden (z. B. für Sprungverteiler).

E/A-Adressierung (I/O direct adressing)



Der ATmega16 hat einen eigenen Adressraum von 64 Byte für E/A-Register (6-Bit-Adressfeld A in IN/OUT-Befehlen). Dieser ist alternativ (aber langsamer) auch über Speicheradressen im Datenbereich ansprechbar.

IN R6, $1B	**R6 ← IOREG ($1B)**
OUT $1B, R7	**IOREG ($1B) ← R7**

oder alternativ:

```
LDS R6, $3B
STS $3B, R7
```

Achtung: Speicheradresse = I/O-Adresse + $20

LDS, STS benötigen 1 Takt mehr als IN, OUT

9.2.6 Assembler-Programmierung

Entwicklungsumgebung AVR Studio 4

Assembler-Direktiven wie beim HATmega16

Befehlszeile

Marke	Befehl	Operand(en)	Kommentar
ANF:	LDI	R16, $12	; Lade Register R16 mit der Zahl $12

Marke: Symbolische Bezeichnung (Label) für die Speicheradresse, an der sich der Befehl befindet (Groß-/Kleinschreibung ist irrelevant, ´:´ wichtig).

Befehl: Assembler-Befehl (Op-Code) im Mnemo-Code aus ATmega16-Befehlsliste oder Assembler-Direktive (Groß-/Kleinschreibung irrelevant).

Operand(en): Bis zu 2 Operanden (je nach Befehl), ggf. durch ”,” getrennt. Marken oder Ausdrücke mit arithmetischen/logischen Operationen (z. B. +, -, *, /, !, ~, !=, &, |, <=, <) sind möglich und werden vom Assembler zur Assemblierzeit ausgewertet, dsgl. Shiftoperationen (>>, <<).

Darstellung von Literalen:

dezimal	10
hexadezimal	$F8 oder 0xF8
binär	0b10110101
Zeichen	’X’
PC	aktueller Wert des *Program Location Counter* des Assemblers

Kommentar: Durch „;“ vom Befehl getrennt. Eigene Kommentarzeilen mit „;“ am Anfang.

Jeder Befehl sollte einen Kommentar tragen!

Makros

Makroanweisungen („*Makros*“) sind Befehlsfolgen, die nur einmal definiert werden und vom Assembler bei jedem Aufruf eingefügt werden (= Makroexpansion beim Einsetzen des Bezeichners).

Bei der Definition eines Makros wird also kein Code erzeugt, aber an jeder Stelle des Aufrufs.

Anmerkung: Die Definition muss vor dem ersten Aufruf stehen.

Makros dienen ähnlich wie Unterprogramme dazu, umfangreiche Programmieraufgaben zu strukturieren. (mögliche *Seiteneffekte* beachten!)
Beim Aufruf eines Makros werden die formalen Parameter (@0, …, @9) durch die aktuellen Parameter ersetzt.

Beispiel:

Nachbilden des fehlenden Befehls für die unmittelbare Addition einer Konstanten zu einem Register durch Subtraktion des Komplements

```
.MACRO  ADDI        ; Beginn der Makrodefinition
  SUBI  @0, -@1     ; Substrahiere vom Register @0
                    ; die Konstante -@1
.ENDM               ; Ende der Makrodefinition
```

Aufruf mit: ADDI R1, $20

Der formale Parameter @0 wird beim Aufruf durch R1 ersetzt, und @1 durch $20.

D. h., ADDI R1, $20 wird ersetzt durch SUBI R1, -$20.

Tipps zur Assemblerprogrammierung des ATmega16

* Allen Registern sollten mit einer .DEF-Direktive ein aussagekräftiger symbolischer Name (Label) zugewiesen werden. Das gilt auch für die E/A-Register, ggf. sogar für einzelne Bits, um Programme lesbar und wartbar zu machen.

* Deklarationen von Bezeichnern, Makros etc. können in einer Datei abgelegt und mittels .INCLUDE “Dateiname.INC“ einem Programm zugeordnet werden.

* Als Pointer auf den Datenspeicher (RAM) können die X-, Y- und Z-Register verwendet werden.

* Für das Lesen aus dem Programmspeicher (Flash) kann nur das Z-Register verwendet werden. Register R0 wird beim LPM-Befehl besonders behandelt. Daher sollten ggf. beide Register für das Lesen von nicht flüchtigen Paramertern reserviert werden.

* Werden häufig konstante Werte oder Zugriffe auf einzelne Bits von Registern verwendet, sollten die Register R16 bis R23 dafür verwendet werden.

* Für alle anderen Anwendungsfelder dienen dann vorzugsweise die Register R1 bis R15.

* Der Zugriff auf die E/A-Register geschieht am besten mit den IN- und OUT-Befehlen.

* Initialisierung des Stackpointers (auf die höchste RAM-adresse mit dem vordefinierten Label RAMEND) nicht vergessen.

* Unterprogramme sollten Register, die nur intern verwendet werden, als erstes retten und am Ende wieder restaurieren.

* Label in Unterprogrammen und Makros sollten unterprogramm- bzw. makro-spezifische Namenszusätze haben, um Namenskonflikte mit anderen Programmteilen zu vermeiden.

9.2.7 Peripherie-Einheiten

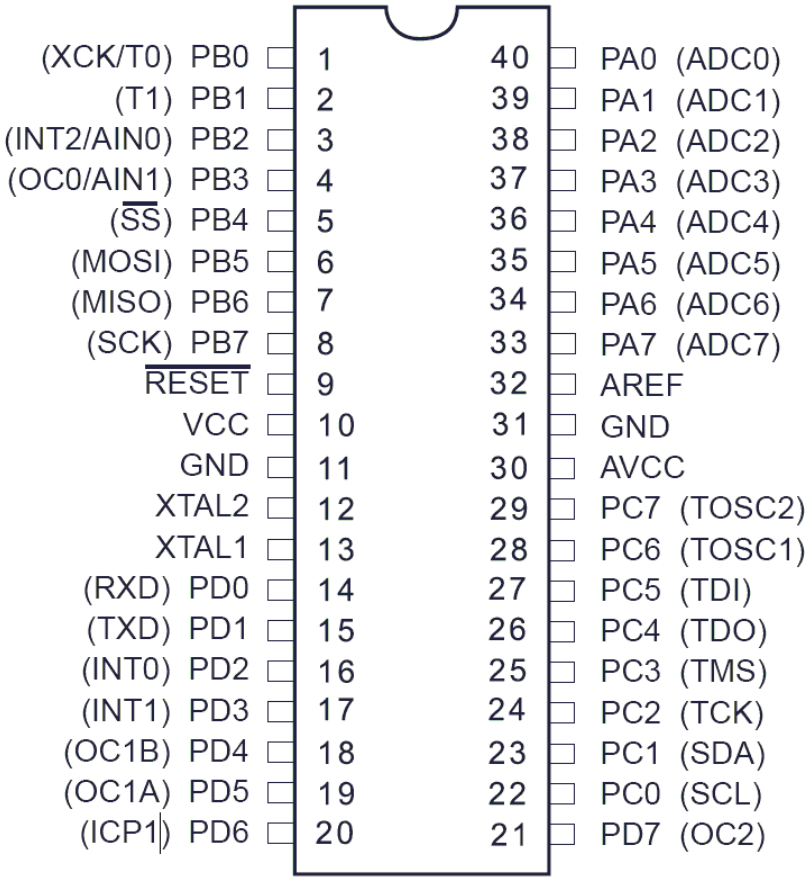
Der ATmega16 hat vielfältige Ein-/Ausgabe-Möglichkeiten, die über Kontrollregister sehr flexibel konfigurierbar sind. Die Portanschlüsse des ATmega können dazu eine von mehreren verschiedenen Funktionen übernehmen.

Der ATmega16 unterstützt das Abfragen, ob neue Eingaben da sind, sowohl durch ständiges Überprüfen (Polling) als auch asynchron über Unterbrechungen (Interrupts → s. u.).

Interne Zählerbausteine (Timer) erlauben verschiedene Zeitgeber- und Zählerfunktionen.

Die bis zu 64 **Kontroll**register liegen im **Daten**-Speicher-Adressraum $0020 bis $005F bzw. **E/A**-Adressraum $00 bis $3F (64 Byte I/O-Register).

Hinweis: Nachfolgend nur Überblick über die wichtigsten E/A-Funktionen mit typischen Beispielen.

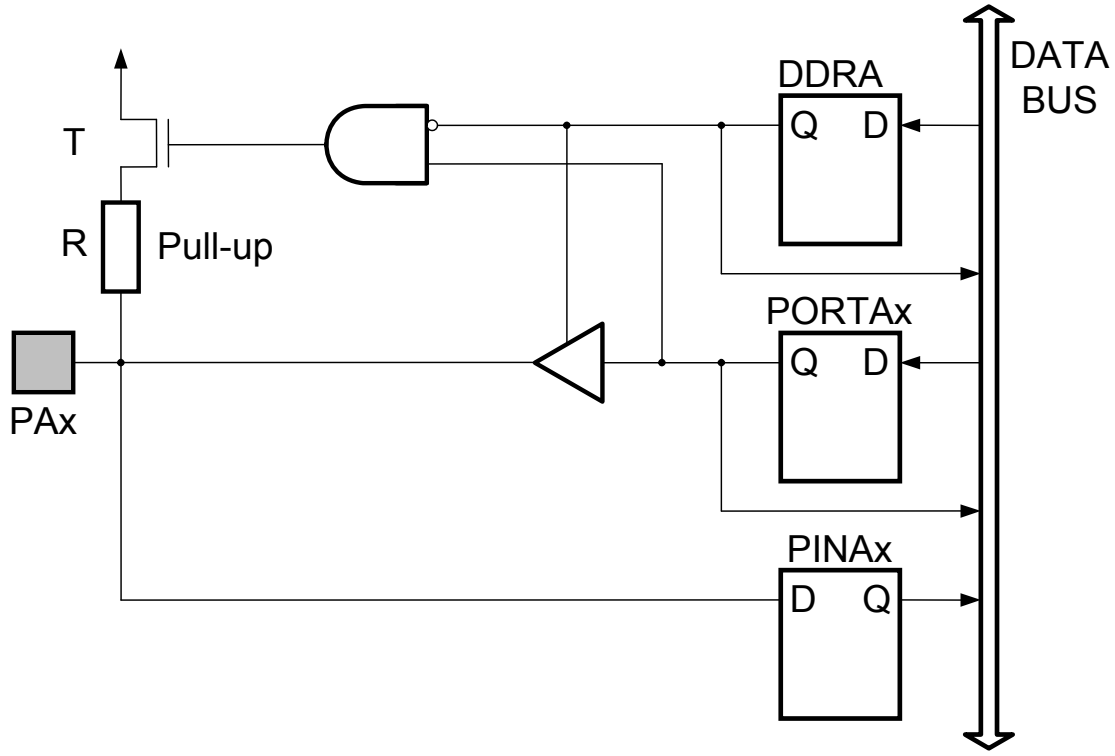


Übersicht E/A-Register ATmega16 (mit den Standard-Labeln)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	SPH	–	–	–	–	–	SP10	SP9	SP8
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register							
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
$35 ($55)	MCUCR	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)							
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register							
	OCDR	On-Chip Debug Register							
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte							
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte							
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte							
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte							
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte							
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)							
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register							
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]			
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
$1F ($3F)	EEARH	–	–	–	–	–	–	–	EEAR8
$1E ($3E)	EEARL	EEPROM Address Register Low Byte							
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$0F ($2F)	SPDR	SPI Data Register							
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
$0C ($2C)	UDR	USART I/O Data Register							
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte							
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
$05 ($25)	ADCH	ADC Data Register High Byte							
$04 ($24)	ADCL	ADC Data Register Low Byte							
$03 ($23)	TWDR	Two-wire Serial Interface Data Register							
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register							

Parallelports

Schaltung E/A-Pin Exemplarisch für Port A (vereinfacht)



$19 ($39)	PINA	8-Bit Eingabepin-Register (read-only)
$1A ($3A)	DDRA	8-Bit Data Direction Register (r/w)
$1B ($3B)	PORTA	8-Bit (Ausgabe-)Port-Register (r/w)

Jeder Pin kann mittels DDR-Register einzeln für Eingabe (hochohmig oder mit Pull-up-Widerstand) oder Ausgabe konfiguriert werden.

DDRAx	**PORTAx**	**I/O**
0	0	Input Tri-State (hochohmig, Hi-Z)
0	1	Input Pull-up (intern)
1	0	Output Low (“0”)
1	1	Output High (“1”)

Beispiel-Konfiguration für Port A

Pin		
0	Ausgabe “1”	DDRA : 0000 1111
1	Ausgabe “1”	$ 0 F
2	Ausgabe “0”	(1→ output, 0 → input)
3	Ausgabe “0”	
4	Eingabe Tri-State	PORTA : 1100 0011
5	Eingabe Tri-State	$ C 3
6	Eingabe Pull-up	
7	Eingabe Pull-up	

```
                        ; Konfiguration PORTA
LDI R16, $0F            ; Setze DDRA-Direction
OUT $1A, R16            ;
LDI R16, $C3            ; Setze PORTA-Out-Data
OUT $1B, R16            ;
...

IN R18, $19             ; Lies Pin-Werte von
                        ; PORTA in R18
```

Mit dem IN-Befehl werden alle aktuellen Werte der Pins von Port A (inkl. Rücklesen der Ausgabepins) eingelesen.

Port B, Port C, Port D

können ebenfalls wie Port A als Parallelport genutzt werden. Der Zugriff erfolgt analog über die entsprechenden Register/Adressen (siehe Tabelle).

Diese Ports bieten andere alternative Schnittstellenfunktionen als Port A.