

Mikrocontroller

Teil 1:

Aufbau und Struktur Programmierung

Autor: Dipl.-Ing. Edgar Hoch

(bearb.: Dipl.-Ing. J. Wemheuer, Stand: 11.2017)

Literaturempfehlungen

- C.Kühnel, Programmieren der AVR RISC Mikrocontroller mit BASCOM-AVR, [ISBN 3898119378](#) (2000) [ISBN 3907857046](#) (2.Aufl.2004) [ISBN 978-3-907857-14-4](#) (3. überarbeitete und erweiterte Auflage 2010)
- R.Mittermayr, AVR-RISC: Embedded Software selbst entwickeln, Franzis 2008 [ISBN 3772341071](#)
- F.Schäffer, AVR: Hardware und C-Programmierung in der Praxis, Elektor 2008 [ISBN 3895762008](#)
- G.Schmitt, Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie, Oldenbourg 4.Aufl.2008 [ISBN 3486587900](#) [ISBN 3486580167](#) (2006) [ISBN 3486577174](#) (2005)
- M.Schwabl-Schmidt, Programmiertechniken für AVR-Mikrocontroller, Elektor 2008 [ISBN 3895761761](#)
- M.Schwabl-Schmidt, Systemprogrammierung für AVR-Mikrocontroller, Elektor 2009 [ISBN 3895762180](#)
- W.Trampert, Messen, Steuern und Regeln mit AVR Mikrocontrollern, Franzis 2004 [ISBN 3772342981](#)
- W.Trampert, AVR-RISC Mikrocontroller, Franzis [ISBN 3772354769](#) (2003) [ISBN 3772354742](#) (2002) [ISBN 3772354750](#) (2000)
- P.Urbaneck, Embedded Systems: Ein umfassendes Grundlagenwerk, ... (2007) [ISBN 3981123018](#)
- S./F.Volpe, AVR-Mikrocontroller-Praxis, Elektor 2001 [ISBN 3895760633](#)
- R.Walter, AVR-Mikrocontroller-Lehrbuch 3. Auflage, Denkholtz 2009 [ISBN 9783981189445](#)

Einleitung

Mit der Entwicklung der Digitaltechnik wurde sehr schnell erkannt, dass der Markt elektronische Systeme benötigt, deren Funktion nicht alleine durch die Hardware bestimmt wird. Man versuchte Systeme zu entwickeln, deren Funktion durch den Anwender bestimmt werden kann, ohne die eigentliche Schaltung umbauen zu müssen.

Die erste „programmierbare“ Schaltung wurde und wird ALU genannt. Hinter diesem Kürzel steht *Arithmetic Logic Unit*. Sozusagen das Kernstück aller Mikroprozessoren. Aus dieser Schaltgruppe entwickelten sich Mikrorechner in mehreren Sparten – je nach Anwendung. Wir werden in diesem Studienheft nicht alle Entwicklungen darstellen können und werden uns auf drei Entwicklungen fixieren:

Mikroprozessoren sind die zentrale Recheneinheit jedes Computers. Ihre Anwendung ist hauptsächlich die Rechnertechnik.

Mikrocontroller sind Mikroprozessoren, die neben den Komponenten der Mikroprozessoren noch weitere sonst periphere Baugruppen beinhalten. Mikrocontroller sind Einchipsysteme, auch SoC (System on a Chip) genannt: Alle Baugruppen befinden sich in einem Gehäuse. Anwendung finden Mikrocontroller zum Beispiel in der Steuerungstechnik.

RISC-Prozessoren sind Mikrocontroller mit einem reduzierten Befehlssatz. Sie sind im Aufbau nahezu identisch mit den Mikrocontrollern. Alle Baugruppen befinden sich in einem Gehäuse. Die Anwendung dieser Prozessoren findet in sehr vielen Gebrauchsgegenständen des täglichen Lebens statt. Teilsteuerungen in Waschmaschinen, Steuerungen im Automobil usw. statt. Diese Prozessoren werden Sie in den folgenden Kapiteln dieses Studienbriefes kennen lernen.

Folgende Studienziele werden mit diesem Studienbrief erreicht:

- Kenntnis der Grundstruktur einer ALU
- Kenntnis der Grundstruktur und Baugruppen eines Mikroprozessors
- Kenntnis der Grundstruktur und Baugruppen eines Mikrocontrollers
- Kennen lernen des Grundbefehlssatzes eines Mikrocontrollers
- Programmentwicklungen in Assembler

A green letter 'S' is centered within a light gray square box.

Inhaltsverzeichnis

Einleitung	3
1 Mikroprozessoren Hardware	7
1.1 Die Arithmetisch Logische Einheit ALU	7
1.2 Von Neumann Computermodell	8
1.3 Die Komponenten des von Neumann Modells	8
1.4 Mikrocontroller	13
1.5 Die Anforderungen an Mikrocontrollersystem	14
1.6 Mikrocontrollerfamilien und Typen	14
1.7 Die Harvard-Architektur	15
1.8 Zusammenfassung	16
1.9 Lernerfolgskontrolle	16
2 Entwicklungssysteme	17
2.1 Der Aufbau des Entwicklungssystems	17
2.2 Der Aufbau des Mikrocontrollers AtmelMega8	18
2.3 Allgemeine Betriebsdaten	18
2.3.1 Die Betriebsspannung	19
2.3.2 Der Systemtakt	19
2.4 Speicherorganisation und Größe der Speicher	19
2.5 Register	20
2.5.1 Die Besonderheit von Registern:	20
2.5.2 Besondere Register	21
2.6 Die PORTs	22
2.7 Datendirektionsregister	23
2.8 Zusammenfassung	26
2.9 Lernerfolgskontrolle	26
3 Mikrocontroller Praxis 1	27
3.1 Das Workpad Entwicklungssystem in Betrieb nehmen	27
4 Mikrocontroller Programmierung in Assembler	30
4.1 Der Assembler	30
4.1.1 Textlayout des Quellcodes	30
4.2 Der Editor des myAVR Workpads	31
4.3 Programmübertragung auf das Entwicklungsboard	32
4.4 Klassifizierung der Assembler-Befehle	33
4.5 Einen konstanten Wert in ein Register laden (Transferbefehl)	33
4.6 Datenausgabe an PORT oder Datendirektionsregister DDR	34
4.7 Der unbedingte relative Sprung	34
4.8 Erstes Programm in Assembler „LED steuern“	35
4.9 Übung 1	37
4.10 Darstellung eines Programms in einem Programmablaufplan PAP	37
4.11 Zweites Programm in Assembler „Taste steuert LED“	39
4.11.1 Anschluss der Hardware	39
4.11.2 Pullup-Widerstand	39
4.11.3 Der Programmablaufplan	40
4.11.4 Der IN-Befehl	40
4.11.5 Internen Pullup-Widerstand setzen	41
4.12 Zusammenfassung	43
4.13 Übung 2	43
4.14 Lernerfolgskontrolle	44
5 Mikrocontroller Praxis 2	45

5.1	Bitmanipulationsbefehle.....	45
5.1.1	Setzen/Rücksetzen eines Bits in einem Register oder PORT	45
5.1.2	Programmentwicklung 1.....	46
5.1.3	Bedingte Sprungbefehle mit Skip.....	47
5.1.4	Programmoptimierung	50
5.2	Übung 3.....	52
5.3	Weitere Skip-Befehle	52
5.3.1	SBRC – Skip if Bit in Register is Cleared.....	52
5.3.2	SBRS – Skip if Bit in Register is Set	52
5.4	Zusammenfassung	53
5.5	Lernerfolgskontrolle.....	53
6	Mikrocontroller Praxis 3	54
6.1	Unterprogramme	54
6.1.1	Stackpointer	55
6.1.2	Der Stack	55
6.2	Der Unterprogrammaufruf	55
6.2.1	Aufruf mit rcall	55
6.2.2	Zurück mit ret.....	56
6.2.3	Aufruf mit icall.....	56
6.2.4	Praktische Umsetzung.....	57
6.3	Zusammenfassung	58
6.4	Lernerfolgskontrolle.....	58
7	Mikrocontroller Praxis 4	59
7.1	Zählbefehle mit Registern	59
7.1.1	Inc reg.....	59
7.1.2	Dec reg	59
7.2	Das Statusregister SREG.....	59
7.3	Bedingte Sprungbefehle in Abhängigkeit des Ergebnisses einer arithmetischen oder logischen Operation.....	61
7.3.1	BRNE Sprung bei not equal	61
7.3.2	Zeitbetrachtungen	62
7.3.3	Verschachtelung mehrerer Programmschleifen.....	63
7.4	Übung 4.....	64
7.5	Weitere bedingte Sprungbefehle.....	64
7.5.1	BREQ Sprung if equal	65
7.5.2	BRCS Sprung if Carry Set	65
7.5.3	BRCC Sprung if Carry Cleared.....	65
7.5.4	BRLO Sprung if lower	66
7.5.5	BRGE Sprung if Greater or Equal.....	66
7.6	Zusammenfassung	67
7.7	Lernerfolgskontrolle.....	67
8	Mikrocontrollerpraxis 5	68
8.1	Arithmetische Befehle	68
8.1.1	Addition mit ADD	68
8.1.2	Addition mit ADI.....	68
8.1.3	Subtraktion mit SUB.....	69
8.1.4	Subtraktion mit SUI	69
8.1.5	Besondere Subtraktion mit CPI	69
8.2	Programmentwicklung Blinklicht	70
8.3	Zusammenfassung	72
8.4	Lernerfolgskontrolle.....	72

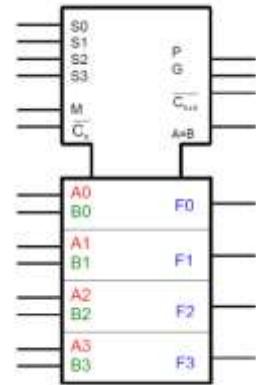
9	Lösungen zu Lernerfolgskontrollen und Übungen.....	74
	Lernerfolgskontrolle 1.8	74
	Lernerfolgskontrolle 2.9	74
	Übung 1 (Kap. 4.9)	75
	Übung 2 (Kap.4.12)	76
	Lernerfolgskontrolle 4.13	77
	Übung 3 (Kap 5.2)	78
	Lernerfolgskontrolle 5.5	80
	Lernerfolgskontrolle 6.4	80
	Übung 4 (Kap 7.4)	80
	Lernerfolgskontrolle 7.6	81
	Lernerfolgskontrolle 8.4	82
10	Anhänge.....	83
10.1	Registerübersicht des AVR atmega8.....	83
10.2	Befehlsübersicht des AVR atmega8.....	84

1 Mikroprozessoren Hardware

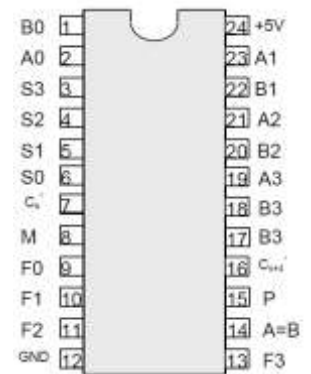
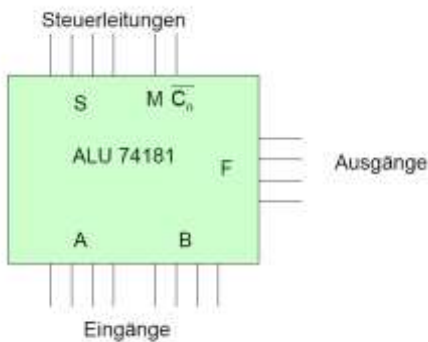
1.1 Die Arithmetisch Logische Einheit ALU

Wie schon in der Einleitung angesprochen, werden in der ALU sowohl arithmetische als auch logische Verknüpfungen durch Steuersignale ausgeführt. Im Studienheft Modul 13, Digitaltechnik haben Sie die Grundverknüpfungen schon kennen gelernt. Diese Grundverknüpfungen und arithmetischen Funktionen wie Addition oder Subtraktion können durch die ALU realisiert werden.

Den folgenden Betrachtungen wird die ALU 74181 zugrunde gelegt. Rechts seitlich sehen Sie das digitale Symbol der ALU, darunter die Anschlussbezeichnungen im 24-poligen DIL Gehäuse.



Die Ein- und Ausgänge der ALU haben folgende Bedeutung:



Die an den Eingängen A und B anstehende Bitstruktur wird in der ALU verknüpft bzw. verrechnet. Wie die beiden Bitstrukturen miteinander verknüpft bzw. verrechnet werden sollen, bestimmt die Bitstruktur an den Steuerleitungen. Diese Bitstrukturen werden auch als Befehl oder Operationscode bezeichnet. Das Ergebnis eines Befehls erscheint sofort an den Ausgängen mit der Bezeichnung F.

Steuerleitungen S0 S1 S2 S3	M=1 logische Operationen	M=0 arithmetische Operationen	
		C _n =0	C _n =1
0 0 0 0	$F = \bar{A}$	$F = A + 1$	$F = A$
0 0 0 1	$F = A \vee \bar{B}$	$F = (A \vee B) + 1$	$F = (A \vee B)$
0 0 1 0	$F = \bar{A} \wedge B$	$F = (A \vee \bar{B}) + 1$	$F = (A \vee \bar{B})$
0 0 1 1	$F = 0$	$F = 0$	$F = -1$
0 1 0 0	$F = A \wedge \bar{B}$	$F = A + (A \wedge \bar{B}) + 1$	$F = A + (A \wedge \bar{B})$
0 1 0 1	$F = \bar{B}$	$F = (A \vee B) + (A \wedge \bar{B}) + 1$	$F = (A \vee B) + (A \wedge \bar{B})$
0 1 1 0	$F = A \vee B$	$F = A - B$	$F = A - B - 1$
0 1 1 1	$F = A \wedge \bar{B}$	$F = A \wedge \bar{B}$	$F = (A \wedge \bar{B}) - 1$
1 0 0 0	$F = \bar{A} \vee B$	$F = A + (A \wedge B) + 1$	$F = A + (A \wedge B)$
1 0 0 1	$F = \bar{A} \vee \bar{B}$	$F = A + B + 1$	$F = A + B$
1 0 1 0	$F = B$	$F = (A \vee \bar{B}) + (A \wedge B) + 1$	$F = (A \vee \bar{B}) + (A \wedge B)$
1 0 1 1	$F = A \wedge B$	$F = A \wedge B$	$F = A \wedge B - 1$
1 1 0 0	$F = 1$	$F = A + A + 1$	$F = A + A$
1 1 0 1	$F = A \vee \bar{B}$	$F = (A \vee B) + A + 1$	$F = (A \vee B) + A$
1 1 1 0	$F = A \vee B$	$F = (A \vee \bar{B}) + A + 1$	$F = (A \vee \bar{B}) + A$
1 1 1 1	$F = A$	$F = A$	$F = A - 1$

Das Ergebnis eines Befehls erscheint sofort an den Ausgängen mit der Bezeichnung F.

Die möglichen Operationen einer ALU sehen Sie in der Tabelle.

Die ALU ist bei allen Mikrorechnern das Kernstück. Sozusagen das Rechenwerk des Systems. Dennoch, allein mit einer ALU ist noch kein Mikrorechner realisiert. Um ein lauffähiges System aufzubauen, werden noch ein paar Bausteine benötigt. Einen groben Überblick gibt das von Neumann Modell.

1.2 Von Neumann Computermodell

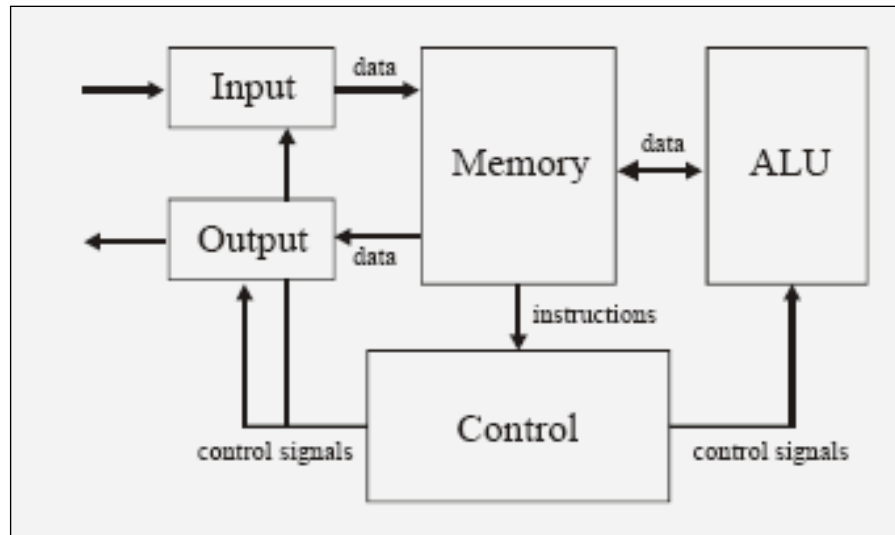


Abbildung 1:
Das von-Neumann-
Computermodell mit den
Grundkomponenten

Die von Neumann Architektur ist ein Schaltungskonzept zur Realisierung universeller Computer. Die **von-Neumann-Architektur** bildet die Grundlage für die Arbeitsweise der meisten Mikroprozessoren. Sie ist benannt nach dem österreichisch-ungarischen, später in den USA tätigen Mathematiker John von Neumann, dessen wesentliche Arbeit zum Thema 1945 veröffentlicht wurde. Sie wird manchmal auch nach der gleichnamigen US-Universität **Princeton-Architektur** genannt.

1.3 Die Komponenten des von Neumann Modells

Nach dem v. Neumann Modell sind für ein lauffähiges Computersystem neben der ALU noch folgende Baugruppen notwendig:

Input: für das Einlesen von externen Daten, die z.B. über eine Tastatur oder über einen Sensor generiert werden

Output: für das Auslesen von Ergebnisdaten an z.B. Anzeigeeinheiten oder an Aktoren.

Memory: das sind Speichersysteme, die Zwischenergebnisse, Ergebnisse und Programmdaten speichern

Control: Mit dieser Einheit ist das Steuer- oder Leitwerk gemeint, das die Anweisungen eines Programms interpretiert und die Befehlsfolge steuert.

Damit die einzelnen Komponenten miteinander arbeiten können, bedarf es eines Systems, das die Baugruppen miteinander verbindet: das Bussystem.

Das Bussystem ist Teil der Hardware, welche die einzelnen Komponenten eines Computersystems miteinander verbindet. Dabei unterscheiden wir die Bussysteme in:

Adressbus: Dieses System überträgt Daten nur in eine Richtung, vom Prozessor an die weiteren Komponenten wie Speicher oder Ein-Ausgabegeräte.

Datenbus: Auf diesem Bussystem werden Daten und Befehle zwischen den einzelnen Komponenten ausgetauscht.

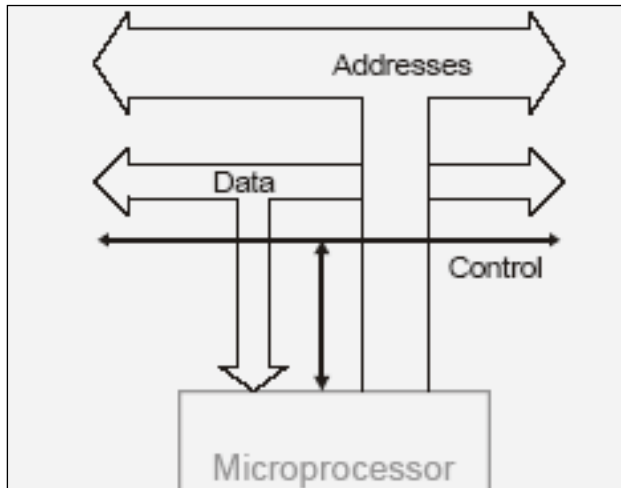


Abbildung 2:
Das Bussystem eines Prozessorsystems

Steuerbus: Dieses Bussystem wird auch Controlbus genannt. Durch den Steuerbus werden vornehmlich die Datenflussrichtung des Datenbusses und die Zusammenarbeit der einzelnen Baugruppen des Systems gesteuert und kontrolliert.

Allgemein gilt, dass das Bussystem eine vorgegebene Anzahl von Informationen parallel verarbeiten kann. Gängige Größen sind:

- 8 Bit parallel
- 16 Bit parallel
- 32 Bit parallel
- 64 Bit parallel

Prozessoren werden nach der Verarbeitungsbreite der Daten durch die interne Architektur eingeteilt. Diese „Verarbeitungsbreite“ wird als Breite der Bitstruktur interpretiert, die ein Prozessor mit einem Befehl gleichzeitig verarbeiten kann.

Wird mit einem Prozessor gearbeitet, der z.B. eine 8-bit-Struktur aufweist, so ist die maximale Dezimalzahl die 255. Nach dieser Struktur müssen auch die Komponenten Eingabegeräte, Ausgabegeräte und die Arbeitsspeicher ausgewählt werden.

Werden größere Bitstrukturen benötigt, werden Rechenoperationen in mehrere Teile aufgeteilt und nacheinander abgearbeitet.

Die Von-Neumann-Architektur weist noch weitere Aspekte auf:

Die intern verarbeitbare Signalmenge ist binär codiert (vergleiche Steuerleitungen der ALU).

Der Mikroprozessor verarbeitet Daten mit fester Datenlänge (Verarbeitungsbreite, Bussystem, Register und Speicher).

Die Programmbefehle werden sequenziell - also nacheinander - in der Reihenfolge ihrer Speicherung abgearbeitet.

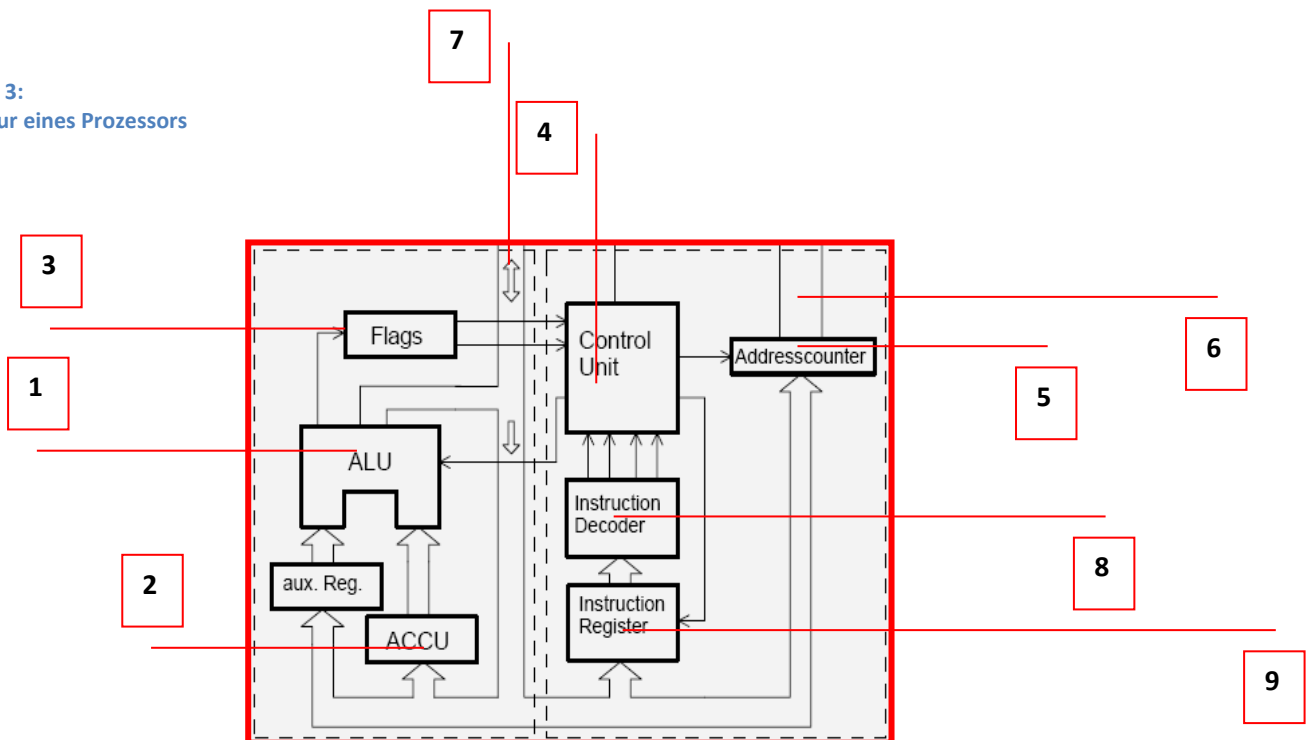
Ein Befehlszyklus hat immer dieselbe Struktur. Beispiel:

Lade den Befehlszeiger mit fester Startadresse (Festlegung der Anfangsadresse des Programms)

- Lies den Befehl aus der Speicheradresse, auf die der Befehlszeiger zeigt.
- Decodiere diesen Befehl im Steuerwerk.
- Führe den Befehl im Rechenwerk aus.
- ← Erhöhe den Befehlszeiger.

Der Mikroprozessor arbeitet taktgesteuert und bearbeitet die Befehle sequenziell, also nacheinander. Betrachtet man die Struktur eines Mikroprozessors etwas genauer, ergibt sich folgendes Bild:

Abbildung 3:
Die Struktur eines Prozessors



Erklärung der Komponenten:

1 Die Arithmetisch Logische Einheit: hier finden die Berechnungen und logischen Verknüpfungen statt, die von der Control-Einheit vorgegeben werden und die als Bitkombinationen im Speicher des Systems stehen.

2 Der Akkumulator: Hierin werden Ergebnisse und Zwischenergebnisse geschrieben, die in der ALU erzeugt wurden.

3 Die Flags: In der Regel ein 8-Bit-Speicher (Register). In dieses Register wird die Beschreibung eines Ergebnisses einer arithmetischen oder logischen Operation geschrieben. Jedes Bit dieses Registers hat eine eigene Bedeutung. Beispielsweise, wenn das Ergebnis einer Addition einen Übertrag liefert, dann wird das durch das Setzen eines Bits (in der Regel Carry-Bit genannt) signalisiert.

4 Control Unit: Hier erfolgt die Aktivierung des Steuerbusses. Beispielsweise, ob Daten geschrieben oder gelesen werden sollen. Die Weitergabe erfolgt über den Steuerbus, an den alle Systemkomponenten angeschlossen sind.

5 Adresscounter: Auch Befehlszähler genannt. Dieser zeigt auf die aktuelle Adresse des Speichers aus dem Daten oder Programminstruktionen gelesen oder geschrieben werden.

6 Adressbus: Dieser liefert die aktuelle Adresse für Speicher oder Ein-Ausgabeeinheiten, die extern angeschlossen sind.

7 Datenbus: Hierüber erfolgt die Übertragung von Daten und Befehlen.

8 Decoder: Hier wird die Bitstruktur eines Befehls in Aktivitäten der einzelnen Komponenten umgesetzt.

9 Instruction Register (Befehlsregister): Das sind Speicherplätze im System, die für die schnelle Übertragung und Verarbeitung von Daten innerhalb des Systems verantwortlich sind. Sie halten den gerade auszuführenden Befehl bereit, der zur Ausführung selbst eventuell mehrere Taktzyklen benötigt. In manchen Systemen werden hier „auf Vorrat“ bereits die nächsten auszuführenden Befehle zwischengespeichert, um eine schnellere Befehlsausführung zu ermöglichen („Pipelining“).

Eine neben der Parallelstruktur sehr wichtige Größe bei Mikroprozessoren oder Mikrocontrollern ist die Taktfrequenz. Der Takt bestimmt die Verarbeitungsgeschwindigkeit der Daten im System.

Der Takt bestimmt, wann ein Befehl im System abgearbeitet wird und ist die zentrale Synchronisationsstelle aller Baugruppen im System. Typische Taktfrequenzen beginnen bei 1 MHz bis hin zu über 3 GHz.

Ein Mikroprozessor ist als Einzelbauelement allerdings nicht lauffähig. Damit ein System lauffähig wird, werden weitere Baugruppen wie **Speicher für Daten und Programme** und Ein-/Ausgabesysteme für die **Dateneingabe** von beispielsweise Sensoren und die **Ausgabe** von Rechenergebnissen an beispielsweise Aktoren benötigt. Diese Baugruppen ergänzen den Mikroprozessor zum Mikrorechner. Hier sehen Sie ein Beispiel eines Mikroprozessorsystems auf der Basis des Mikroprozessors 8085 der Firma Intel.

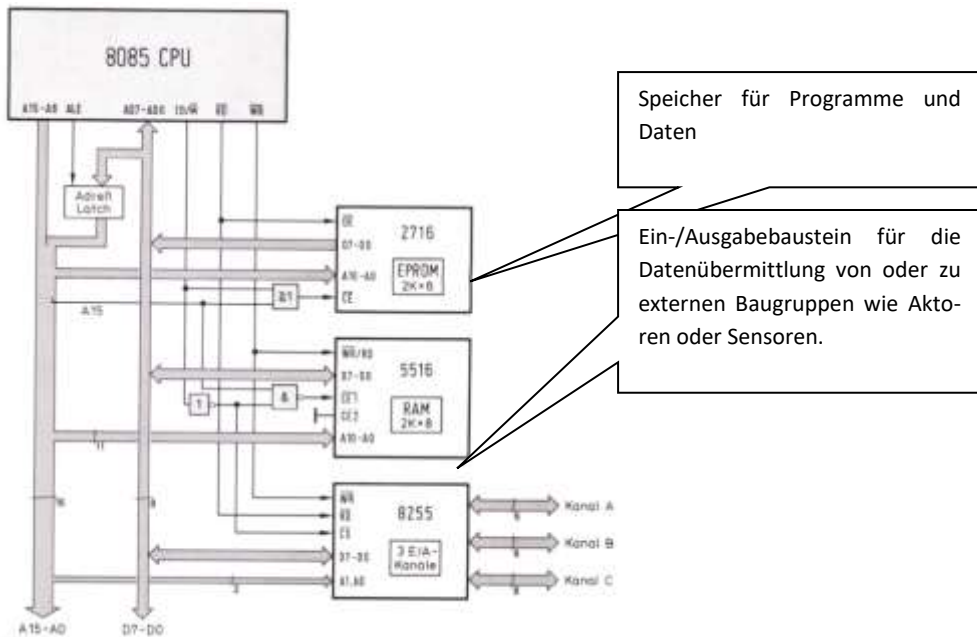


Abbildung 4: Mikroprozessorsystem auf der Basis 8080/8085

Typische Vertreter der geschichtlichen Entwicklung von Mikroprozessoren sind im folgenden Bild dargestellt.

	Intel 4004	2300	1971	Intel
	Intel 8008	3500	1972	Intel
	Intel 8080	4500	1974	Intel
	6502	5000	1975	MOS Technology
	Intel 8088	29.000	1979	Intel
	Motorola 68000	68.000	1979	Motorola
	Intel 80286	134.000	1982	Intel
	Intel 80386	275.000	1985	Intel
	Intel 80486	1.200.000	1989	Intel
	Pentium	3.100.000	1993	Intel

Abbildung 5: Geschichtliche Entwicklung von Mikroprozessoren

Diese Mikroprozessoren wurden hauptsächlich für Rechenaufgaben entwickelt und waren das Kernstück der ersten PCs. Sie mussten große Datenmengen verarbeiten und waren dementsprechend für diese Aufgaben auch ausgelegt.

Für den Einsatz der Mikroprozessoren in der Steuerungstechnik waren diese Baugruppen allerdings nicht besonders geeignet. Diese Baugruppen werden in diesem Studienbrief nicht weiter untersucht. In der Steuerungstechnik werden keine großen Datenmengen verarbeitet – hier sind andere Rechnermerkmale wichtig. Diese sollen im Folgenden genauer betrachtet werden.

1.4 Mikrocontroller

Natürlich können alle Probleme der Steuerungstechnik auch mit den verschiedenen Baugruppen der Digitaltechnik und Analogtechnik gelöst werden. Durch das Ersetzen von „Hardware“ – also einzelner Baugruppen – durch „Softwarelösungen“ können Problemlösungen aber

- kostengünstiger produziert werden und
- weitaus flexibler gestaltet werden.

Denkt man beispielsweise daran, dass ein Antriebssystem – zum Beispiel ein Verbrennungsmotor – noch vor Jahren hauptsächlich durch Schraubenschlüssel und mechanische Veränderungen erfolgte, so ist es heute üblich, dass diese Optimierung hauptsächlich durch Veränderungen in der Software erfolgt, die den oder die Mikrocontroller steuern (Chiptuning).

Die vormals diskret mit Bauelementen realisierten Funktionen werden in von Mikrocontrollern gesteuerten Systemen in die Software verlagert. Die Komplexität und Größe der mechanischen und elektronischen Systeme kann durch den Einsatz von Mikrocontrollern wesentlich reduziert und flexibler gestaltet werden. Damit senken sich die Herstellungskosten. Je höher die Stückzahlen eines Produkts sind, umso bedeutender werden die Herstellungskosten.

Mikrocontroller findet man deswegen heute auch schon in recht einfachen Geräten wie MP3-Player, DVD-Player, Geschirrspüler, Kaffeemaschinen, Fernsehgeräten, Wecker und Uhren. Eine weitere große Bedeutung haben Mikrocontrollersysteme in der Steuerungstechnik bei der Messwerterfassung, intelligenten Sensoren und Aktoren in Automobilen, Handys und der Medizintechnik.

1.5 Die Anforderungen an Mikrocontrollersystem

Mikrocontrollersysteme arbeiten meist in Systemen, in denen es auch auf den Energieverbrauch ankommt. Die Controller sollen also möglichst wenig Energie benötigen und möglichst schnell Daten verarbeiten können, die von Sensoren geliefert werden und an Aktoren abgegeben werden. Weitere Anforderungen sind:

- programmierbar (Update, Optimierung, Wartung)
- flexible Schnittstellen zu anderen Komponenten
- Selbstdiagnose und Fehlerkorrekturmöglichkeiten
- echtzeitfähig (schnelle Reaktionszeiten)
- kostengünstig

1.6 Mikrocontrollerfamilien und Typen

Mikrocontroller werden von vielen verschiedenen Herstellern angeboten. In der Grobstruktur sind diese auf den ersten Blick einander sehr ähnlich. Bekannte Hersteller sind

- Intel – mit der Baureihe 8051 und deren Derivate
- Microsystems – mit PIC- Controller
- Atmel – mit der AVR-Reihe
- Infineon XC 167
- Daneben gibt es sehr viele weitere Hersteller von Mikrocontrollern.

Die Controller der Serie AVR des Herstellers Atmel sind recht fortschrittlich am Markt und haben enorme Zuwachsraten. Der Kern der Controller wurde an der Universität Trondheim in Norwegen entwickelt.

Diese Controller sollen für die folgenden Betrachtungen in diesem Studienbrief beispielhaft betrachtet werden.

Mikrorechnersysteme für Steuerungsaufgaben sind von der Struktur ähnlich aufgebaut wie die Mikroprozessoren. Sie müssen jedoch spezielle Aufgaben erfüllen und sind dafür entsprechend konzipiert. Wichtige Funktionen dieser Systeme sind:

- Ein-/Ausgabebaugruppen – Ports
- Verschiedene Speicherarten – EEPROM – FLASH – SRAM
- Analog- Digitalwandler
- Zeitgeberbaugruppen – Timer
- Zählerbaugruppen – Counter
- Integrierter Systemtakt

Alle Baugruppen sind in einem Chip untergebracht und machen das System zu einem vollwertigen Steuerungssystem – also einem kleinen Rechner.

Damit Sie im Studium kleine Steuerungen selbst entwickeln und die im Studienbrief dargestellten Programme testen können, empfehlen wir das Entwicklungssystem myAVR. Dieses Entwicklungssystem werden Sie auch in den Präsenzphasen des Fernstudiums benutzen.

AtmelMega8

Das Mikrocontrollersystem AtmelMega8 ist nach der Harvard-Architektur aufgebaut. Deshalb hier noch einmal die Charakteristiken dieser Mikrocontrollerarchitektur.

1.7 Die Harvard-Architektur

Die Harvard-Architektur ist im Kern älter als die von-Neumann-Architektur, denn die ersten Computer (etwa die Zuse 3) hatten eigentlich nur einen Datenspeicher. Der "Programmspeicher" wurde als Lochstreifen oder Lochkarten realisiert. Erst später wurde das Programm auf Speichermedien, z. B. einem Trommelspeicher abgelegt. Typisch war also die Trennung von Daten- und Programmspeicher.

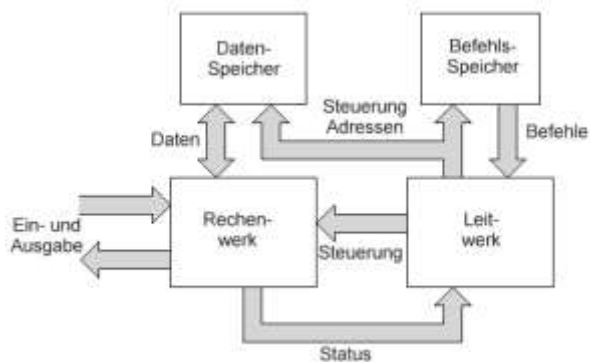


Abbildung 6:
Die Harvard-Architektur

Die Harvard-Architektur bezeichnet heute ein Schaltungskonzept zur Realisierung besonders schneller Mikrocontroller. Befehlsspeicher und Datenspeicher sind voneinander getrennt und werden über getrennte Busse angesteuert. Daher können Befehle und Daten gleichzeitig geladen, bzw. geschrieben werden. Bei einer klassischen von-Neumann-Architektur sind hierzu mindestens zwei aufeinander folgende Buszyklen notwendig.

Zudem sorgt die Trennung von Daten und Programm dafür, dass bei Softwarefehlern kein Programmcode überschrieben werden kann. Nachteilig ist allerdings, dass nicht benötigter Datenspeicher nicht als Programmspeicher genutzt werden kann.

Soviel zur Struktur des Mikrocontrollersystems. Wenden wir uns nun dem Aufbau des Entwicklungssystems zu, nachdem Sie die Lernerfolgskontrolle bearbeitet haben.

Z

1.8 Zusammenfassung

- Die wichtigste Komponente der Mikroprozessoren ist die arithmetisch-logische Einheit, genannt ALU.
- Die Architektur der verschiedenen Prozessoren wird in die beiden Grundstrukturen „von Neumann“ und „Harvard“ unterschieden.
- Die Verbindungen der einzelnen Komponenten innerhalb eines Prozessors werden Bus genannt. Im Besonderen der Adressbus, der Datenbus und der Steuerbus (Controlbus).
- In dem Von-Neumann-Konzept werden alle Daten über diese Bussysteme von einer Baugruppe zur anderen transportiert. Im Gegensatz dazu, werden Daten im Harvard-Konzept so mit den Baugruppen verbunden, dass die schnellste Laufzeit der Daten erreicht wird.
- Bei Mikroprozessoren werden nach Einsatzgebiet unterschiedliche Architekturen und Bezeichnungen verwendet. Mikroprozessoren stehen für Computeranwendungen, Mikrocontroller sind in der Steuerungstechnik zuhause. Ein optimierter Mikrocontroller ist beispielsweise ein RISC-Prozessor.

L

1.9 Lernerfolgskontrolle

Erarbeiten Sie die Unterschiede „Von-Neumann-Architektur“ und „Harvard-Architektur“.

2 Entwicklungssysteme

Systementwickler für Informationssystem auf Mikroprozessorbasis oder Mikrocontrollerbasis benötigen für Ihre Arbeit ein Hardwarekonzept und eine Programmiersprache, mit der eine Steuerungsaufgabe oder eine Problemlösung erarbeitet werden soll.

2.1 Der Aufbau des Entwicklungssystems

Über eine USB Schnittstelle kann das Board mit Hilfe eines PC programmiert werden. Als Programmiersprachen können Assembler, C-C++, Pascal oder BASIC, ein BASIC-Derivat, verwendet werden. Wir werden uns in diesem Studienbrief zunächst mit dem Assembler begnügen.

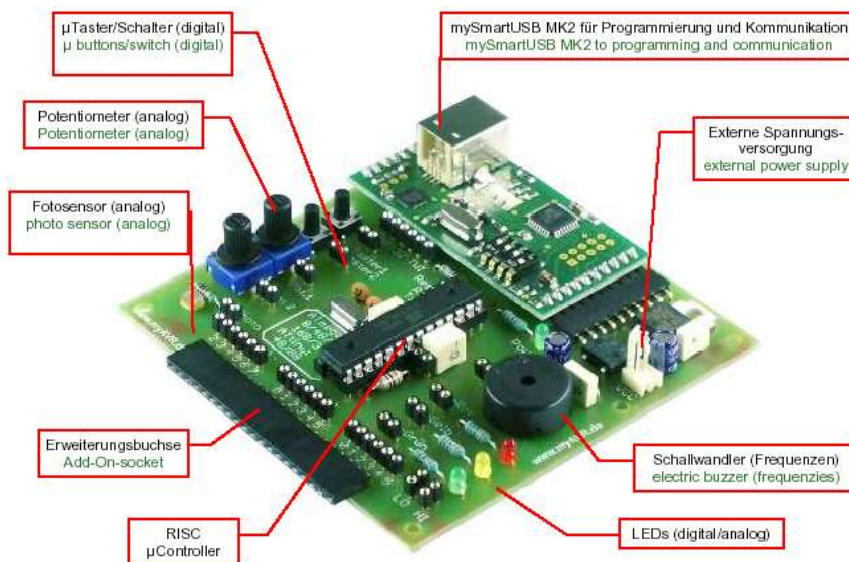


Abbildung 7: Das Experimentierboard myAVR MK2

Das Board ist neben dem Mikrocontroller AVR bestückt mit

- zwei Analogwertgebern (Potentiometer),
- zwei Tastern zur Eingabe von Digitalsignalen,
- drei LEDs zur Ausgabe von Digitalsignalen,
- einem Fotosensor und
- einer Messerleiste zur Erweiterung von Ein-/ Ausgabesignalen,
- einer ISP-Schnittstelle mit USB-Anschluss.

Diese werden zu Simulationszwecken und zum Austesten von Programm-Entwicklungen verwendet. Das Board ist sehr übersichtlich konzipiert und leicht zu programmieren.

2.2 Der Aufbau des Mikrocontrollers AtmelMega8

Das folgende Bild zeigt den inneren Aufbau des Mikrocontrollers/RISC-Prozessors AtmelMega8. Sie werden die wichtigsten Baugruppen und ihre Funktion nun kennen lernen.

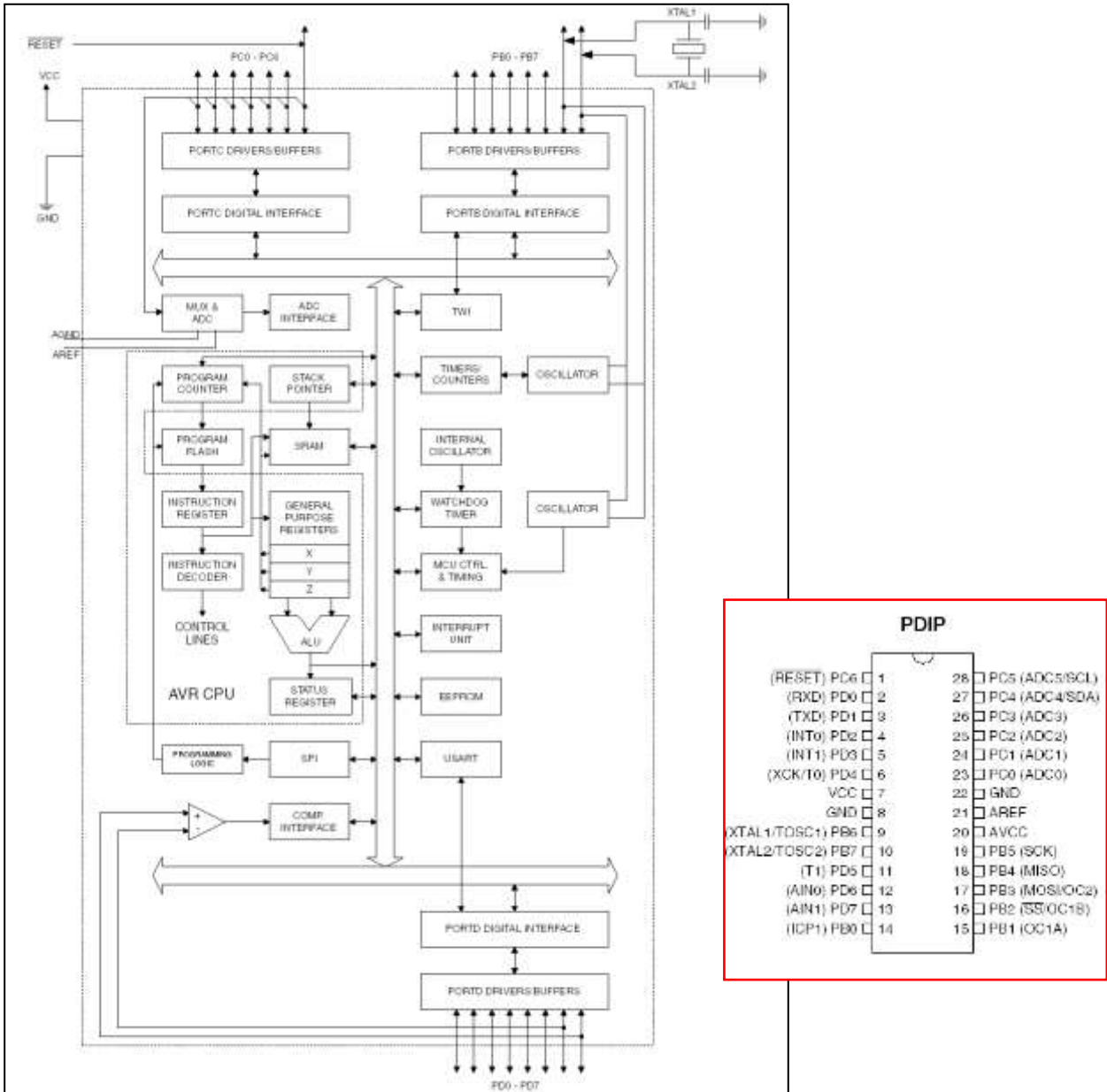


Abbildung 8: Innerer Aufbau des Mikrocontrollers und Sockelbeschriftung

2.3 Allgemeine Betriebsdaten

Zu den allgemeinen Betriebsdaten, die für den Hardwareentwickler wichtig sind, zählen der interne Aufbau eines Prozessorsystems und auch die physikalischen Grunddaten, die in den folgenden Betrachtungen beschrieben werden

2.3.1 Die Betriebsspannung

Der Mikrocontroller wird mit einer Betriebsspannung zwischen 4 und 6 Volt betrieben, manche Ausführungsformen auch mit 2,7 bis 6 Volt. Die Spannungen können also sehr variabel sein. Gerade bei Anwendungen mit einer autarken Spannungsversorgung sind Spannungsschwankungen in gewissen Grenzen möglich. Der Betriebsstrom ist abhängig von der Betriebsspannung und ist beispielsweise bei 5V zwischen 10 und 12 mA. Diese Stromangabe ist ein Mittelwert und ist abhängig von der jeweiligen Programmausführung.

2.3.2 Der Systemtakt

Der Takt kann an zwei Pins (9 und 10) extern durch einen Quarz oder Keramikschwinger erzeugt werden. Der Mikrocontroller kann aber auch mit einem internen Takt betrieben werden. Die Taktung bzw. die Taktfrequenz mit dem internen RC-Generator ist allerdings sehr abhängig von der Betriebsspannung und der Betriebstemperatur.

Die interne Taktfrequenz ist ca. 1 MHz. Diese Frequenz ist jedoch nicht allzu genau. Werden genaue Frequenzen gefordert, wird deshalb der Takt als externer Takt durch einen Quarz realisiert. Eine häufig verwendete Taktfrequenz ist 4 MHz bzw. 3,68 MHz. Diese wird auch im verwendeten Experimentierboard eingesetzt. Der Systemtakt kann bis zu 16 MHz betragen. Die Stromverbrauchswerte verändern sich dadurch ebenfalls.

2.4 Speicherorganisation und Größe der Speicher

Die Speicher sind in drei Gruppen organisiert, die eigene Charaktere haben für unterschiedliche Zwecke genutzt werden.

Im Flash-Speicher werden die Programme abgelegt. Der Programmzähler hat direkten Zugriff auf diesen Speicherbereich.

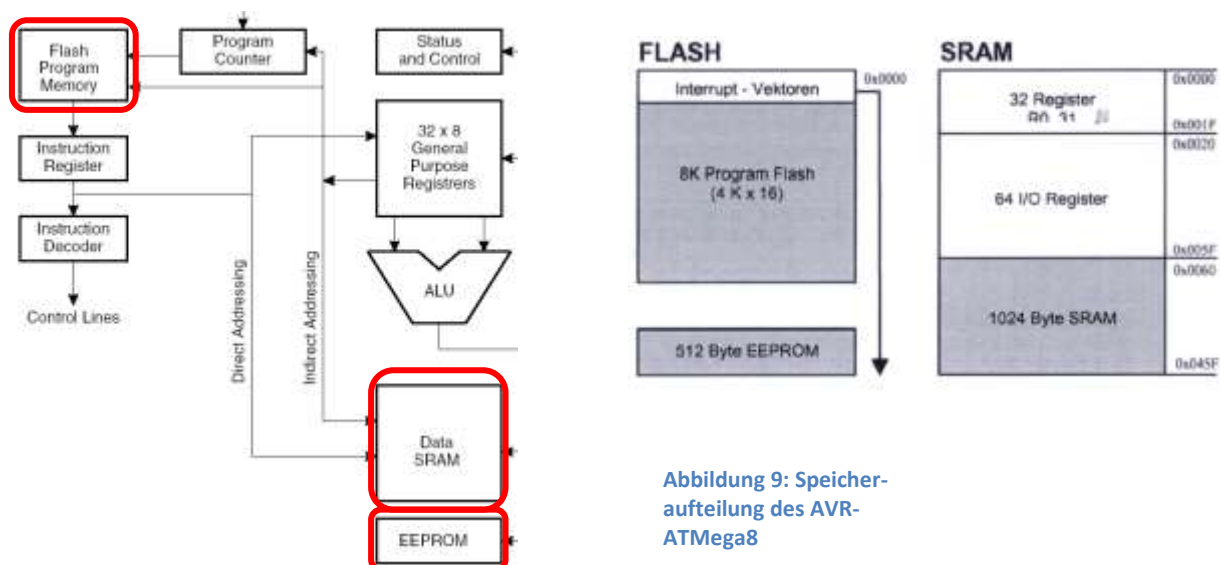


Abbildung 9: Speicher-
aufteilung des AVR-
ATMega8

Flash-Speicher haben den Vorteil, dass auch nach Abschaltung der Speisepannung der Inhalt nicht verloren geht.

Speichergröße des Flash-Speichers beim AtmelMega8: 4K x 16

Neben den Programmen werden auch die Interrupt-Vektoren in diesem Speicherbereich abgelegt. Mehr dazu im Kapitel „Interrupt-Verarbeitung“.

Im **EEPROM** werden hauptsächlich Konstanten des Anwenders abgelegt. Dieser Speicherteil wird durch indirekte Adressierung vom Anwender genutzt. Auch dazu mehr in späteren Kapiteln. EEPROMs behalten Ihre Information auch ohne Betriebsspannung. Der Schreib-Lesezyklus ist langlebiger als beim Flash.

Speichergröße des EEPROMs 512 Byte

Im **SRAM** werden Daten gespeichert, die nur temporären Charakter haben. Deren Inhalt also nach einer bestimmten Zeit wieder überschrieben oder neu beschrieben werden kann. Zwischenergebnisse, Rücksprungadressen von Unterprogrammen usw. sind hier gespeichert. Dieser Speicher ist auch für den Programmierer durch indirekte Adressierung zugänglich. Ein Teil des SRAM-Speichers ist für die Register reserviert. Register sind Speicherplätze, die durch den Programmierer direkt angesprochen und genutzt werden können.

Speichergröße des SRAM 1024 Byte

2.5 Register

Register sind zunächst Speicher mit einer Kapazität von 8 Bit. Die Nummerierung der Bits beginnt mit 0 und endet mit 7.

Registerinhalt / Bits								
7	6	5	4	3	2	1	0	Bezeichnung
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Wertigkeit

2.5.1 Die Besonderheit von Registern:

Sie können direkt durch Befehle aus dem Befehlssatz angesprochen werden.

Viele Operationen mit dem Inhalt eines Registers können durch einen einzigen Befehlsbyte ausgeführt werden.

- Register sind direkt mit dem Rechenwerk des Controllers verbunden.
- Register können sowohl Quelle von Daten als auch Ziel des Ergebnisses einer Operation sein.
- Register können Ganzzahlen von dezimal 0 bis 255 aufnehmen oder Zahlen mit Vorzeichen von -128 bis +127 aufnehmen.

Der Registerzahl ist bei den Mikrocontrollern und Controllertypen verschieden. Der AVR bietet dem Programmierer 32 Register an. Sie werden mit R0 bis R32

bezeichnet. Diese Bezeichnungen können durch Registerumbenennungen im Programm auch andere Namen bekommen.

Im AVR sind allerdings die Register noch einmal in deren Bedeutung aufgeteilt. Die Register und deren Bedeutung zeigt die folgende Tabelle:

Registerbezeichnung	Bedeutung
R0 Adresse 0000 (hexadezimal)	Übernimmt Datenverkehr zwischen Flash und Register
R1 bis R15 Adressen 0001 bis 000F (hexadezimal)	Reserviert für spezielle Aufgaben im AVR
R16 bis R25 Adressen 0010 bis 0019 (hexadezimal)	Für den Programmierer frei verfügbar
R26 und R27 Adressen 001A bis 001B(hexadezimal)	Für den Programmierer frei verfügbar – die beiden Register sind jedoch als 16Bit Zeigerregister (auch Pointer genannt) zusammenschaltbar und haben dann die Bezeichnung X
R28 und R29 Adressen 001C bis 001D (hexadezimal)	Für den Programmierer frei verfügbar – die beiden Register sind jedoch als 16Bit Zeigerregister zusammenschaltbar und haben dann die Bezeichnung Y
R30 und R31 Adressen 001E bis 001F (hexadezimal)	Für den Programmierer frei verfügbar – die beiden Register sind jedoch als 16Bit Zeigerregister zusammenschaltbar und haben dann die Bezeichnung Z

Tabelle 1: Die Bedeutung der Register R0 bis R31 im AVR

Register haben eine feste Adresse im Speicher, durch die sie angesprochen werden können. Da der Umgang mit den Registeradressen etwas umständlich ist, werden diese Adressen in die Registernamen R0 bis R32 im Assembler umbenannt. Die Adressen und Registernamen sind in der Tabelle wiedergegeben. Wie die Register angesprochen werden können, werden Sie bei der Programmierung in Assembler kennen lernen.

2.5.2 Besondere Register

Für den Datenverkehr zwischen verschiedenen Stellen im AVR und zur Kontrolle von Baugruppen wie AD-Wandler oder Zähler des AVR stellt der Mikrocontroller spezielle PORTs und Register zur Verfügung, deren Bedeutung Sie noch kennen lernen werden. Einen Ausschnitt zeigt die folgende Tabelle, die Sie komplett im Anhang des Studienbriefes finden:

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0F (0x0F)	SREG	I	T	H	S	V	N	Z	C
0x0E (0x0E)	SPH	–	–	–	–	–	SP10	SP9	SP8
0x0D (0x0D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x0C (0x0C)	Reserved								
0x0B (0x0B)	ICSR	INT1	INT0	–	–	–	–	IVSEL	IVCE
0x0A (0x0A)	GIFR	INTF1	INTF0	–	–	–	–	–	–
0x09 (0x09)	TMSK	OCIE2	TOIE2	TCIE1	OCIE1A	OCIE1B	TCIE1	–	TOIE0
0x08 (0x08)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TON1	–	TOV0
0x07 (0x07)	SPMCR	SPMIE	RWWRF	–	RWWRF	BLBSF	PGWRT	PGERS	SPMEN
0x06 (0x06)	TWCR	TWINT	TWRA	TWSTA	TWSTO	TWNC	TWEN	–	TWIE
0x05 (0x05)	MCLCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
0x04 (0x04)	MJCSR	–	–	–	–	WDNF	BCRF	EXTRF	PORF
0x03 (0x03)	TCCR0	–	–	–	–	–	CS02	CS01	CS00
0x02 (0x02)	TCNT0	Timer/Counter0 (8 Bits)							
0x01 (0x01)	OSCCAL	Oscillator Calibration Register							
0x00 (0x00)	SFCSR	–	–	–	–	ACME	PUD	PSR2	PSR10
0x0F (0x0F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
0x0E (0x0E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0x0D (0x0D)	TCNT1	Timer/Counter1 – Counter Register High byte							
0x0C (0x0C)	TCNT1L	Timer/Counter1 – Counter Register Low byte							
0x0B (0x0B)	OCR1AH	Timer/Counter1 – Output Compare Register A High byte							
0x0A (0x0A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low byte							
0x09 (0x09)	OCR1BH	Timer/Counter1 – Output Compare Register B High byte							
0x08 (0x08)	OCR1BL	Timer/Counter1 – Output Compare Register B Low byte							
0x07 (0x07)	ICR1H	Timer/Counter1 – Input Capture Register High byte							
0x06 (0x06)	ICR1L	Timer/Counter1 – Input Capture Register Low byte							
0x05 (0x05)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
0x04 (0x04)	TCNT2	Timer/Counter2 (8 Bits)							
0x03 (0x03)	OCR2	Timer/Counter2 Output Compare Register							

Tabelle 2: Die Register mit besonderer Bedeutung im AVR

2.6 Die PORTs

Die Schnittstelle zur Außenwelt des Mikrocontrollers bilden die Ports mit jeweils 8 Bit Datenbreite. Der Aufbau der PORTs ist dem der Register gleich. Es stehen insgesamt drei Ports zur Verfügung, wobei zu beachten ist, dass die Bits der Ports auch mehrfach belegt sind (siehe Sockelbeschaltung). Diese Mehrfachbelegung der PORTs führt dazu, dass je PORT für den Anwender nur 6 Bits für den Anschluss von Aktoren oder Sensoren zur Verfügung stehen.

PORTX								
PORTX.7	PORTX.6	PORTX.5	PORTX.4	PORTX.3	PORTX.2	PORTX.1	PORTX.0	Bezeichnung
2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	Wertigkeit

X steht allgemein für die PORT-Bezeichnungen B, C und D.

Die Bezeichnungen der PORTs sind:

- PORT B mit den Bitbezeichnungen B0 bis B7
- PORT C mit den Bitbezeichnungen C0 bis C7
- PORT D mit den Bitbezeichnungen D0 bis D7

Reservierte nicht zugängliche Bits der PORTs:

- PORT B.6 und PORTB.7 sind für den Anschluss des externen Quarz vorgesehen
- PORT C.6 ist mit dem Reset-Signal belegt
- PORT C.7 ist für den Anwender nicht zugänglich
- PORT D.0 und PORT D.1 sind für die serielle Schnittstelle reserviert (UART – *Universal Asynchronous Receiver Transmitter*)

Die PORTs sind mit einem Strom von 3mA bis 20 mA bei einem Spannungspiegel von 5V belastbar, genauer gesagt: jedes einzelne Bit eines PORTs. Somit können beispielsweise Leuchtdioden direkt – über einen Vorwiderstand betrieben – angesteuert werden. Um den Mikrocontroller nicht zu gefährden, muss auf diese maximalen Stromwerte geachtet werden!

Beispiel für die Dimensionierung eines PORTs für den Anschluss einer Leuchtdiode mit einer Durchlass-Spannung von 1,2V, die an PORTC.0 angeschlossen werden soll.

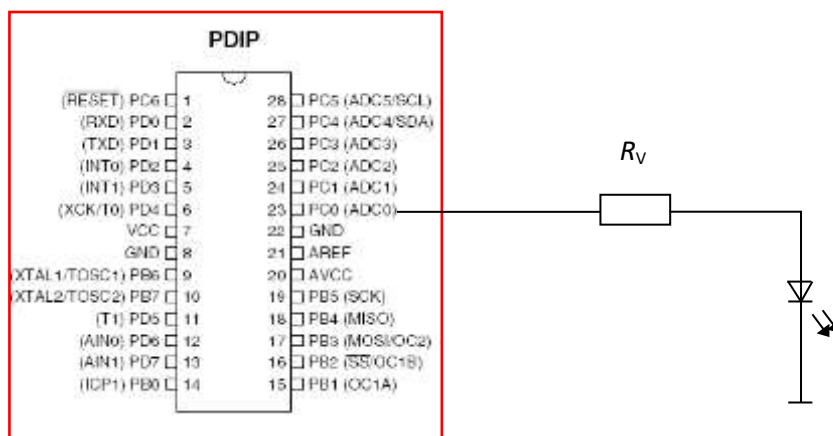


Abbildung 10: Anschluss einer LED an den PORTC

Soll die Leuchtdiode mit maximal 10mA betrieben werden, gilt für die Berechnung des Vorwiderstandes R_V

$$R_V = \frac{U_{AVR} - U_D}{I_{\max}} = \frac{5V - 1,2V}{10mA} = 380\Omega$$

2.7 Datendirektionsregister

Die Datenflussrichtung muss beim AVR programmiert werden, ehe die PORTs benutzt werden können. Die Programmierung der PORTs erfolgt über jeweils drei I/O-Register. Die Datenflussrichtung der PORTs wird durch ein nach- bzw. vorgeschaltetes Register bestimmt. Bei der Programmierung muss beachtet

werden, dass zuerst die Datenflussrichtung bestimmt wird, erst danach kann der Port bedient werden.

Die Datendirektionsregister bezeichnet man im AVR mit:

- DDRB Register für PORT B
- DDRC Register für PORT C
- DDRD Register für PORT D

Datenausgaben erfolgen über das PORT-Register; Dateneingaben über das entsprechende PIN-Register.

PORTX								
PORTX.7	PORTX.6	PORTX.5	PORTX.4	PORTX.3	PORTX.2	PORTX.1	PORTX.0	Daten-Ausgabe
PINX.7	PINX.6	PINX.5	PINX.4	PINX.3	PINX.2	PINX.1	PINX.0	Daten-Eingabe
DDRX.7	DDRX.6	DDRX.5	DDRX.4	DDRX.3	DDRX.2	DDRX.1	DDRX.0	Daten-Direktionsregister

Welches Bit in welchem Register aktiviert wird, bestimmt das zugehörige Datendirektionsregister, kurz DDR genannt. Der folgende Tabellenausschnitt zeigt die Register und deren Adressen.

0x18 (0x38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x17 (0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x16 (0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x15 (0x35)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x14 (0x34)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x13 (0x33)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x12 (0x32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x11 (0x31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x10 (0x30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0

1 an einer Stelle im DDR bedeutet Ausgabe, also PORT-aktiviert

Beispiel:

Je eine Hälfte des PORTB soll auf Eingabe, die andere Hälfte als Ausgabekanal dienen:

Bit 2^0 bis 2^3 auf Eingabe, Bit 2^4 bis 2^7 auf Ausgabe, damit muss das zugehörige DDR die folgende Struktur aufweisen:

PORT B								Bedeutung
DDRB.7	DDRB.6	DDRB.5	DDRB.4	DDRB.3	DDRB.2	DDRB.1	DDRB.0	DDR
1	1	1	1	0	0	0	0	Registerinhalt

Technische Aspekte:

Die Ein- und Ausgänge des AVR sind in C-MOS-Technik ausgelegt und deshalb sehr empfindlich. Schon kleinste Ströme und Spannungen werden als Signal erkannt und dementsprechend im Mikrocontroller ausgewertet.

Werden einpolige Schaltelemente zur Signalerzeugung verwendet, ist zumindest ein Pegel nicht definiert und damit als offener Eingang zu sehen. Die korrekte Schaltung für einen Eingang mit einpoligem Schalter sieht so aus:

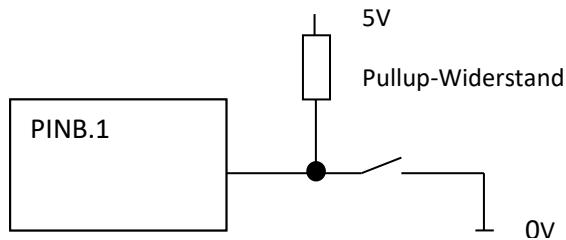


Abbildung 11: Anschluss eines Tasters an PORTB.1 mit Pullup-Widerstand

In dieser Schaltungsvariante wird ein 1-Signal durch einen sogenannten Pullup-Widerstand realisiert. Wird der Schalter geschlossen, wird ein eindeutiges 0-Signal generiert. undefinierte Zustände werden somit vermieden. Um den Strom möglichst gering zu halten, wird ein Widerstandswert von ca. 1 bis 2 kOhm verwendet.

Der externe Pullupwiderstand kann beim AVR auch per Programm realisiert werden. Das werden Sie in späteren Kapiteln noch sehen.

Signale prellen oft

Signale, die durch einen einfachen einpoligen Schalter oder Taster realisiert werden, prellen. Damit ist gemeint, dass der mechanische Kontakt in der Regel nicht ideal schließt oder öffnet.

Dieses prellen eines Kontaktes wird jedoch im Mikrocontroller durchaus registriert und kann zu Programmfehlfunktionen führen. Eine Verbesserung kann durch einen parallel zum Kontakt geschalteten Kondensator mit einem Wert von ca. 1 nF erreicht werden.

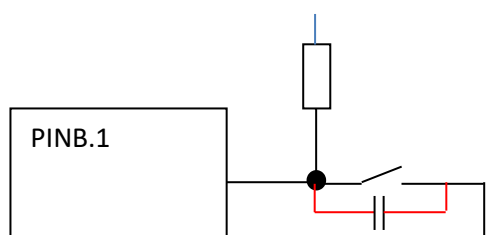


Abbildung 12: Tastenanschluss mit Entprellung durch einen Kondensator

Hier wollen wir die Hardware-Betrachtungen erst einmal unterbrechen und uns der Programmierung in Assembler widmen. Wir werden die Hardware wieder betrachten, wenn die Funktionen der Baugruppen besprochen werden.

Zuvor lesen Sie bitte die folgende Zusammenfassung und lösen Sie die Aufgabe in der Lernerfolgskontrolle.

Z

2.8 Zusammenfassung

- Entwicklungssysteme für Mikrocontroller werden mit Hilfe einer Programmiersprache programmiert. Assembler oder C++ sind die oft verwendeten Sprachen.
- Als Betriebsspannung der Mikrocontroller werden Spannungsbereiche zwischen 2,7V und 6V verwendet.
- Der Systemtakt kann intern oder extern erzeugt werden.
- Mikrocontroller haben verschiedene Speicher. In der Regel sind dies EEPROM, Flash und SRAM in unterschiedlicher Größe. Das SRAM dient als Programmspeicher.
- Speicher mit besonderen Aufgaben werden als Register bezeichnet. Register sind direkt mit dem Rechenwerk des Mikrocontrollers verbunden.
- Die Register mit den Bezeichnungen R0 bis R15 haben im AVR besondere Verwendungen. Für den Programmierer sind die Register mit den Bezeichnungen R16 bis R31 frei zugänglich.
- Sollen Daten vom Mikrocontroller zu externen Baugruppen fließen oder umgekehrt, stehen die PORTs zur Verfügung. Sie werden im AVR mit den Buchstaben B, C und D bezeichnet.
- Zu einem PORT gehören immer drei Register: PORT, PIN und DDR. Im DDR wird die Datenflussrichtung festgelegt.
- Sensoren, wie beispielsweise Schalter müssen eindeutige Signalpegel liefern, sonst kann es zu Programmfehlern bei der Ausführung kommen.
- Aktoren, wie beispielsweise eine einfache Leuchtdiode können mit Strömen bis 20 mA vom Controller direkt versorgt werden.

L

2.9 Lernerfolgskontrolle

- Welche Speichertypen werden im Mikrocontroller Atmega8 eingesetzt und welche Aufgaben haben diese?
- Welche Baugruppen beinhaltet der Mikrocontroller Atmega8. Listen Sie die Ihnen bis jetzt bekannten Baugruppen und benennen Sie die Aufgaben dieser Baugruppen.
- Welche gängigen Taktfrequenzen werden bei Mikrocontrollern eingesetzt?
- Welche Aufgaben haben die Register R16 bis R31?
- Erklären Sie die Aufgaben des Datendirektionsregisters DDR?
- Welchen Inhalt muss das DDRB haben, wenn die Stellen 2^0 bis 2^6 als Ausgabe und die Stelle 2^7 als Eingabe programmiert werden sollen?

3 Mikrocontroller Praxis 1

Mikrocontroller werden vornehmlich in Assembler und in C++ programmiert. Das haben Sie bereits gehört. Wir werden uns in den folgenden Betrachtungen mit der Programmierung in Assembler beschäftigen. Dazu benötigen Sie ein entsprechendes Assemblerprogramm. Wenn Sie das in der Hardware beschriebene Entwicklungssystem „myAVR Board MK2 mit Atmega8“ benutzen, steht Ihnen zu Testzwecken eine Software mit der Bezeichnung „Workpad“ kostenlos zur Verfügung.

Die Software können Sie auf folgender Internetseite als Demoversion downloaden:

<http://shop.myavr.de/index.php?sp=download.sp.php&suchwort=dl62>

Ideal ist es, wenn Sie natürlich das Entwicklungsboard ebenfalls zur Verfügung haben.

3.1 Das Workpad Entwicklungssystem in Betrieb nehmen

Ein einfaches Entwicklungssystem für Assembler und C ist das myAVR Workpad. Wenn Sie das myAVR Workpad von der beschriebenen Seite auf Ihren PC geladen und installiert haben, wird es sich wie folgt melden:



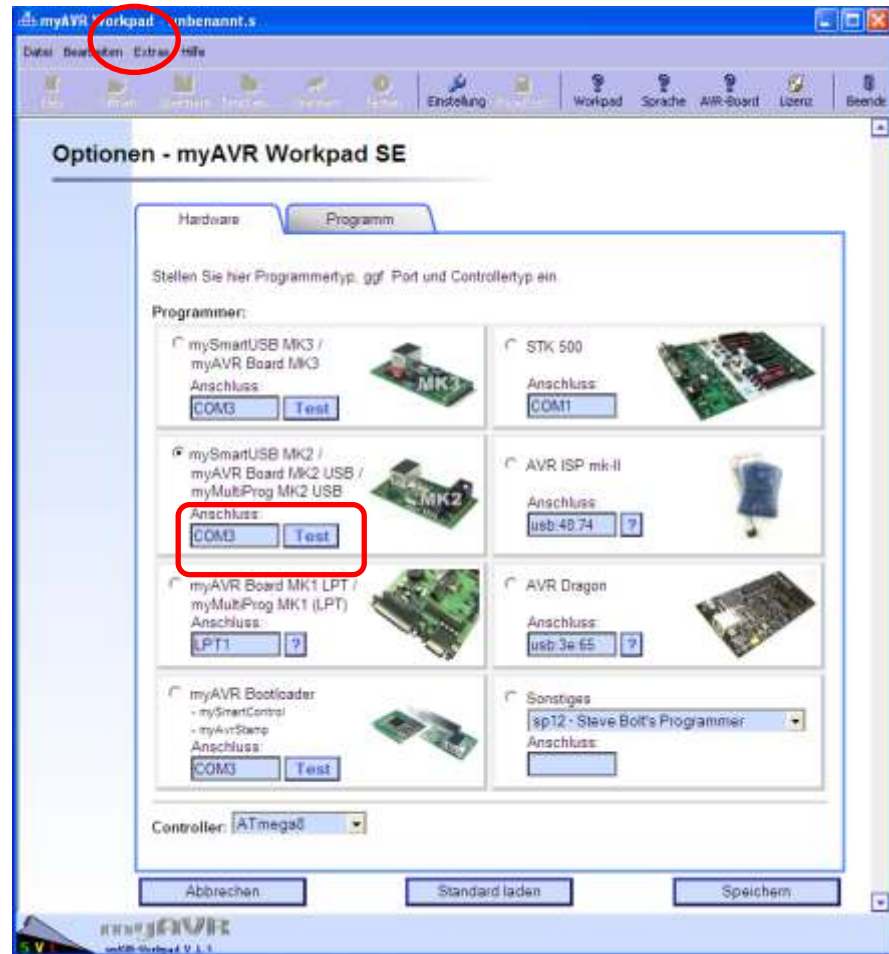
Abbildung 13: Der Eröffnungsbildschirm des my-AVR Workpad

Das myAVR Workpad kann nun mit dem Entwicklungsboard my-AVR Board MK2 USB / verbunden werden. Dazu muss allerdings die entsprechende Einstellung für die USB-Schnittstelle eingerichtet werden. Verbinden Sie das Board mit Ihrem PC und rufen Sie dann folgendes auf:

Menüleiste: **Extras** und dann **Einstellungen**

Sie sollten jetzt das folgende Bild erhalten:

Abbildung 14: Schnittstelle für den COM-Port einstellen und Systemtest. Die Einstellung der Schnittstelle ist abhängig vom PC



Wenn Sie Ihr Entwicklungssystem mit der USB-Schnittstelle mit Ihrem PC verbunden haben, muss die COM-Schnittstelle eingestellt werden. Diese kann in Ihrem System auf COM1 bis COMX liegen. Um herauszufinden, welche COM-Schnittstelle Ihr PC benötigt, klicken Sie auf **Test**.

Falls das System einen Treiber für die COM-Schnittstelle fordert, können Sie diesen auf der nachfolgend genannten Internetseite downloaden und installieren:

<http://shop.myavr.de/index.php?sp=download.sp.php&suchwort=dl46>

Nun kann der Test noch einmal durchgeführt werden. Danach sollte sich das Entwicklungssystem wie folgt melden:

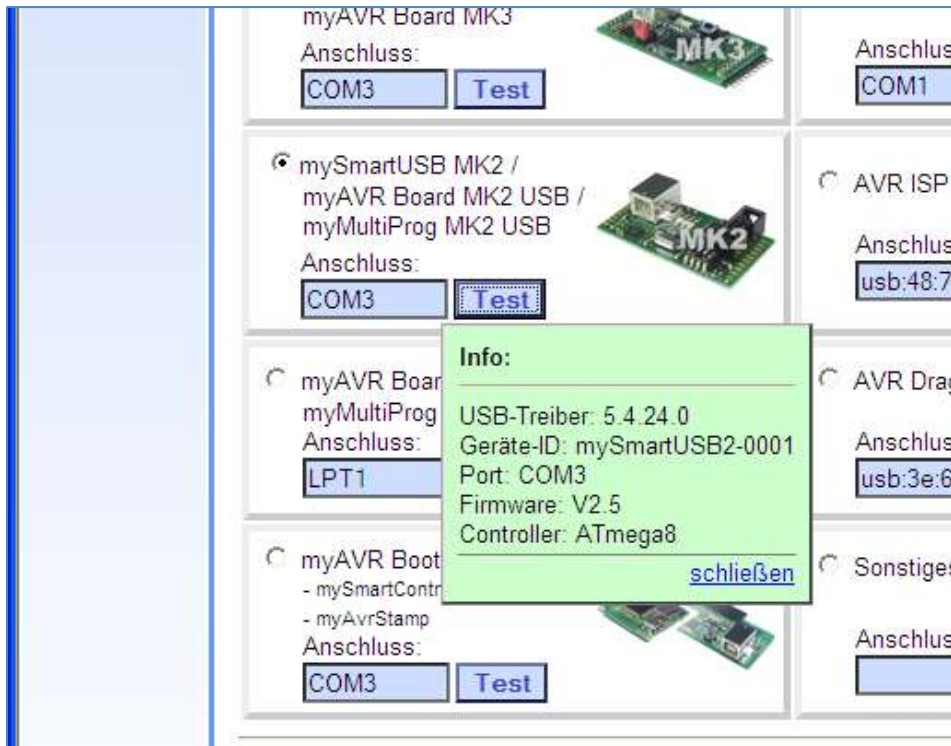


Abbildung 15: Erfolgreicher Test nach Installation des USB-Treibers

Hinweis: Für Windows 7 muss das Workpad plus und der entsprechende Treiber installiert sein. Mit Workpad SE arbeitet das System unter Windows 7 nicht.

Damit sind Sie nun in der Lage, erste Programme zu entwickeln. Allerdings fehlen Ihnen dazu die ersten Befehle in Assembler. Diese wollen wir nun erarbeiten.

4 Mikrocontroller Programmierung in Assembler

Die Programmierung in Assembler ist die universellste Art einen Mikrocontroller zu programmieren. Für den Programmierer ist es wichtig, sich auch in der verwendeten Hardware – also dem Prozessor und den verschiedenen Baugruppen – auszukennen.

Ein Mikrocontroller verarbeitet intern Befehle immer im sogenannten Maschinencode oder Binärcode (vergl. ALU). Ein solcher Maschinenbefehl könnte beispielsweise aus 16 einzelnen Bits bestehen:

```
1001 1011 0000 1000
```

Diese Bitkombinationen sind natürlich auch für den Programmierer sehr unübersichtlich und lassen sich nicht nur schwer lesen, sondern auch schwer merken. Zumal, wenn man bedenkt, dass es im AVR Mikrocontroller ca. 130 verschiedener Befehle gibt.

4.1 Der Assembler

Für die Umsetzung in die Maschinenbefehle stellt der Assembler dem Programmierer etwas übersichtlichere Befehlsformen zur Verfügung, mit denen leichter zu arbeiten ist. Der Assembler ist sozusagen ein Übersetzungsprogramm zwischen Programmierer und Mikrocontroller.

Die Befehle des Assemblers werden Mnemonics genannt. Halbsprechende Abkürzungen also, unter denen man sich die Arbeitsweise eines Befehls vorstellen kann.

Ein Beispiel für einen Maschinenbefehl in Assembler ist MOV. MOV steht für „move“ und bedeutet „bewege“. Oder ein anderes Beispiel: LDI steht für „load immediately“ und bedeutet „lade direkt“.

Ist das Programm in diesen mnemonischen Befehlen geschrieben, kann der Assembler jeden einzelnen Befehl in die Maschinensprache in der Binärcodierung übersetzen und diese dann dem Mikrocontroller mitteilen. Neben der Übersetzungsarbeit leistet der Assembler bei der Programmentwicklung noch weitaus mehr. Zum Beispiel bei der Berechnung von Sprungadressen und anderen mathematischen Operationen.

Grundsätzlich wird ein Assemblerprogramm mit einem Texteditor geschrieben und liegt im einfachen ASCII-Format vor.

4.1.1 Textlayout des Quellcodes

Der Quellcode, so wird die Textformulierung im Editor genannt, wird der Übersichtlichkeit halber in Spalten aufgeteilt, die durch Tabulatoren erzeugt werden.

- In der ersten Spalte stehen Marken, das sind in der Regel Sprungadressen, die auch als Label bezeichnet werden. Eine Sprungadresse muss immer mit einem Buchstaben beginnen und endet mit einem Doppelpunkt.
- Die zweite Spalte enthält die Maschinenbefehle.
- Die dritte Spalte enthält die zum Maschinenbefehl gehörenden Operanden. Die Reihenfolge der Operanden ist immer zuerst das Ziel und dann die Quelle.
- Beispiel: MOV R15, R17 bedeutet, dass der Inhalt des Registers mit der Bezeichnung R17 in das Register mit der Bezeichnung R15 kopiert wird.
- In der letzten Spalte werden dann die Kommentare zur jeweiligen Befehlszeile geschrieben, falls dies zum Verständnis des Programms beiträgt. Kommentare - also Texte, die den Assembler nicht interessieren bei der Übersetzung des Quellcodes - beginnen mit einem Semikolon.

Sehen wir uns das einfach im Editor des myAVR Workpad an. Rufen Sie dazu das Workpad jetzt auf.

4.2 Der Editor des myAVR Workpads

Der Eröffnungsbildschirm erscheint wie es das Bild zeigt

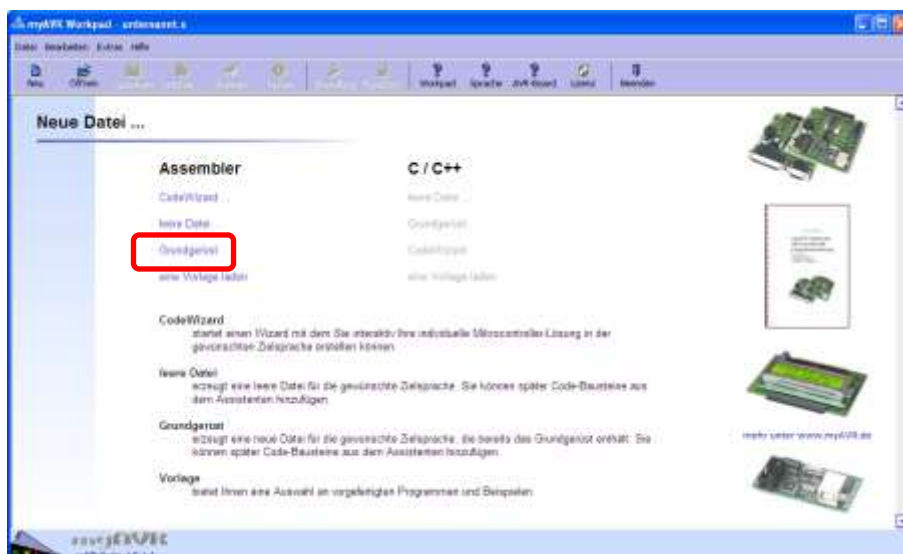


Abbildung 16: Hier wird das Grundgerüst gewählt. Grundeinstellungen werden damit übernommen

Damit Sie zu Beginn der Programmierarbeit nicht alle Grunddefinitionen selbst eingeben müssen, wählen Sie bitte die Funktion „Grundgerüst“

Sie erhalten dann folgenden Bildschirm

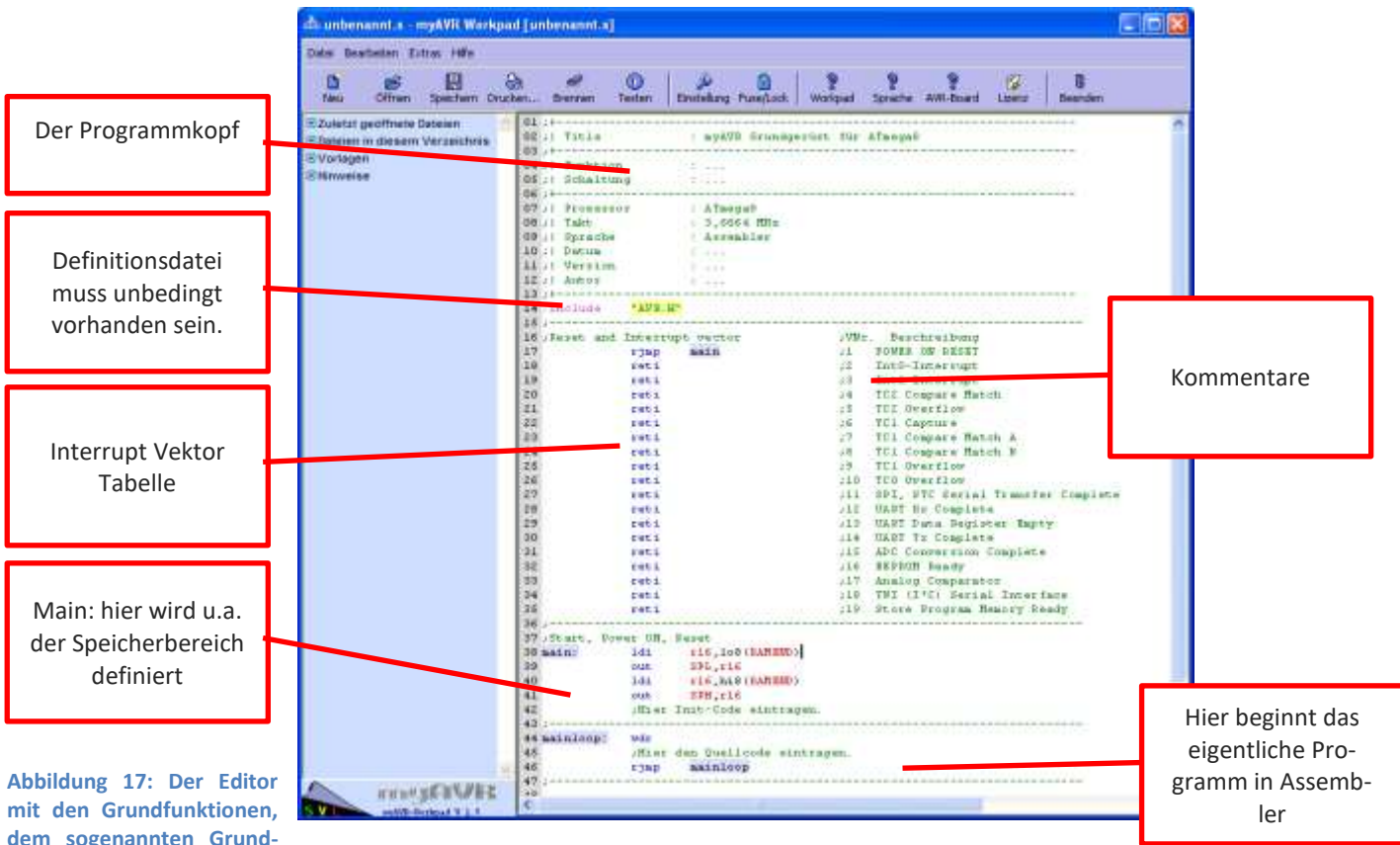


Abbildung 17: Der Editor mit den Grundfunktionen, dem sogenannten Grundgerüst

4.3 Programmübertragung auf das Entwicklungsboard

Wenn Sie die eben gezeigte Seite auf Ihrem Bildschirm haben, können Sie auch gleich einmal die Übertragung des Programms auf das Entwicklungsboard machen. Dazu betätigen Sie in der Menüleiste den Punkt „Brennen“. Mit Brennen ist hier die Übersetzung des Programms und die Übertragung des Programms auf das Board gemeint. Im unteren Drittel des Editors sollte nun ein grünes Feld erscheinen, in dem beispielsweise folgende Zeilen zu lesen sind:

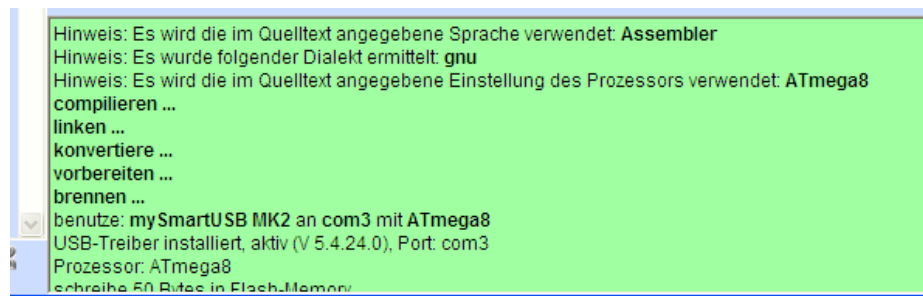


Abbildung 18: So wird ein erfolgreich übertragenes Programm angezeigt

Damit ist das Programm übertragen. Wir haben lediglich das Grundgerüst übertragen, also werden wir auch noch keine Funktion feststellen können.

Um nun ein Programm schreiben zu können, benötigen Sie Kenntnisse über den Aufbau der Befehle und der Befehlsstruktur des AVR. Die ersten Befehle werden Sie gleich kennen lernen.

4.4 Klassifizierung der Assembler-Befehle

Eine grobe Einteilung der Befehle lässt sich in die folgender Gliederung darstellen:

Befehlsgruppe	Beispiel
Arithmetisch- logische Befehle	Addition zweier Registerinhalte oder AND-Verknüpfung mit eines Registerinhalts mit einer Konstanten
Lade- und Transferbefehle	Register mit einem Wert laden. Wert in ein anderes Register übertragen
Sprungbefehle	Programmschleife, Sprung zu einer Sprungmarke
Einzelbit Befehle	Einzelbits in einem Register oder PORT setzen oder rücksetzen

Wir werden im Folgenden nun die einzelnen Befehle des AVR in Bedeutung und Ausführung beschreiben und in einem kleinen darauf folgenden Programm bzw. Programmentwicklung gleich austesten.

4.5 Einen konstanten Wert in ein Register laden (Transferbefehl)

Der entsprechende Befehl lautet:

Befehl	Operand 1	Operand 2
LDI	RegX	Konstante
Load immediate	R16.....R32	0 - 255

Die Schreibweise von konstanten Werten kann im Assembler in verschiedenen Zahlensystemen erfolgen:

Dezimalzahlen – da die Register in der Regel 8-Bit-Register sind, können Zahlenwerte zwischen 0 und 255 geladen werden. Die Dezimalzahlen werden direkt ohne weitere Zusätze geschrieben.

Binärzahlen – sollen Binärzahlen verwendet werden, muss dies dem Assembler mitgeteilt werden. Dies geschieht durch Vorsetzen der Zeichen **0b** vor die eigentliche Binärzahl. Dadurch wird dem Assembler mitgeteilt, dass Binärzahlen verwendet werden.

Beispiel: 0b0000 0001

Hexadezimalzahlen – sollen Hexadezimalzahlen verwendet werden, muss dies ebenfalls dem Assembler mitgeteilt werden. Dies geschieht durch Vorsetzen der Zeichen **0x** vor die eigentliche Hexadezimalzahl.

Beispiel: 0xFE

Je nach Anwendung kann es sinnvoll sein, zwischen den Zahlensystemen innerhalb eines Programms zu wechseln.

4.6 Datenausgabe an PORT oder Datendirektionsregister DDR

Der entsprechende Befehl lautet:

Befehl	Operand 1	Operand 2
OUT	PORTX oder DDRX	RegX
Ausgabe	PORTB, PORTC, PORTD, DDRB, DDRC, DDRD	Reg16....Reg31

Die Bedienung eines PORTs oder eines Datendirektionsregisters DDR erfolgt mit Hilfe eines der allgemeinen Register. Es können alle Register, die frei zugänglich sind, verwendet werden. Also die Register mit den Bezeichnung R16 bis R32.

Nun fehlt noch ein Befehl und Sie können das erste kleine Programm schon selbst schreiben.

Im AVR gibt es keinen direkten Befehl für das Programmende, wie in anderen Programmiersprachen. In der Regel beginnt man das Programm wieder von Anfang an durch einen Sprung zu einer bestimmten Sprungmarke.

Anmerkung: das Wort PORT und DDR muss mit Großbuchstaben geschrieben werden, sonst versteht es der Assembler nicht.

4.7 Der unbedingte relative Sprung

Soll ein Programm an einer bestimmten Sprungmarke fortgesetzt werden, kann dazu der Befehl **rjmp** mit nachfolgend genannter Sprungmarke verwendet werden.

Der Befehl lautet:

Befehl	Operand 1	Operand 2
rjmp	Adresse	kein
Relative jump	Sprungmarke wie <i>main-loop</i> :	

Nun kann das erste Programm geschrieben werden. Beispielsweise die Ansteuerung eines Aktors in Form einer Leuchtdiode als Kontrollleuchte.

4.8 Erstes Programm in Assembler „LED steuern“

Programmfunktionsbeschreibung:

Eine Leuchtdiode soll an PORTB,0 angeschlossen werden und per Programm eingeschaltet werden. Eigentlich ein recht einfaches Programm – dennoch müssen einige grundsätzliche Dinge berücksichtigt werden, die wichtig sind:

- Der PORTB, Stelle 0 muss erst als Ausgabekanal definiert werden.
- Der PORT kann nicht direkt bedient werden, der Ausgabewert muss erst in ein Register geladen werden.
- Erst dann kann der Registerwert in den PORT übertragen werden.
- Das Programm wird im Editor geschrieben.
- Das Programm wird „gebrannt“ also in den AVR übertragen und über-
setzt.
- Das Programm kann im AVR ausgeführt werden.

Diese Schritte werden wir nun gemeinsam durchführen. Dazu benötigen wir das Workpad und darin das Grundgerüst. Außerdem sollten Sie das Entwicklungsboard angeschlossen haben.

Auf dem Entwicklungsboard verbinden Sie den PORTB, Stelle 0 mit der Leuchtdiode mit einem Verbinder (am besten mit einem steifen Draht von einer Telefonleitung).

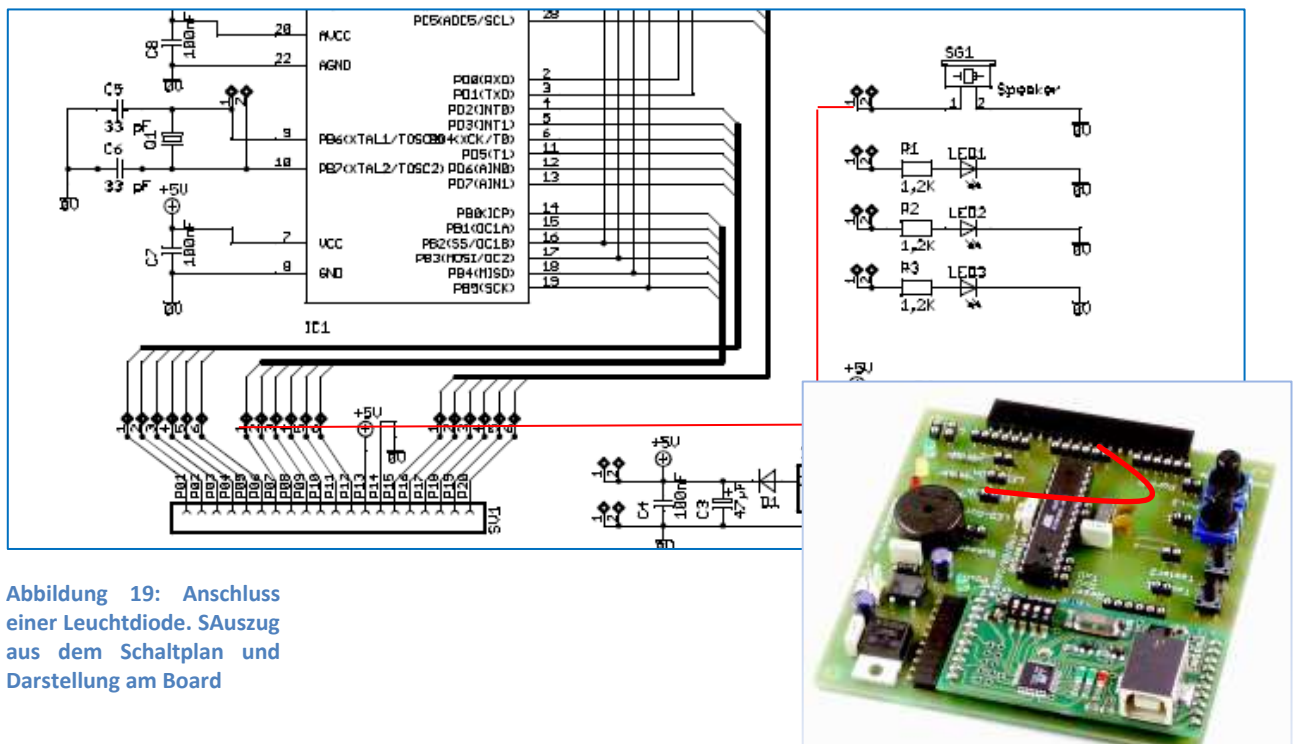


Abbildung 19: Anschluss einer Leuchtdiode. S. Auszug aus dem Schaltplan und Darstellung am Board

Die Verbindung der Leuchtdiode sehen Sie im Schaltplan und daneben die Verbindung auf dem Board.

Nun kann das Programm erstellt werden. Dazu verwenden Sie das Workpad.

Zuerst muss die Datenflussrichtung von PORTB bestimmt werden. Dazu steht uns das zum PORTB gehörende Datendirektionsregister DDRB zur Verfügung. Sie erinnern sich: eine 1 bedeutet Datenausgabe, eine 0 bedeutet Dateneingabe. Jedes einzelne Bit kann im Datendirektionsregister DDR als Ein oder Ausgabe gesetzt werden.

Die Programmierung der Datenflussrichtung wird im Programmteil *main* erledigt. Die entsprechenden Befehlszeilen können wie folgt lauten:

<i>Befehl</i>	<i>Operand 1</i>	<i>Operand 2</i>	<i>Kommentar</i>
LDI	R16	0b11111111	Konstante für alle Bits in PORTB auf Ausgabe stellen
OUT	DDRB	R16	

Damit ist die Datenflussrichtung programmiert. Nun muss durch das Hauptprogramm (mainloop) die Leuchtdiode eingeschaltet werden.

Die entsprechenden Befehlszeilen lauten:

Befehl	Operand 1	Operand 2	Kommentar
LDI	R16	0b00000001	Stelle 20 =1 schaltet die Leuchtdiode ein
OUT	PORTB	R16	

Mit dem OUT-Befehl werden immer alle 8 Bits eines PORTs bedient. Wir benötigen nur die Stelle 0 mit einem 1 Signal, die anderen Stellen sind nicht von Bedeutung und werden auf 0 gesetzt.

Fehlt nur noch der unbedingte relative Sprungbefehl zum Beginn des Programms:

Befehl	Operand 1	Operand 2	Kommentar
rjmp	mainloop	kein	Rücksprung zu mainloop , die tatsächliche Adresse wird vom Assembler berechnet und beim Assemblieren an den AVR übergeben

Im Editor des Workpad sieht das nun so aus:

```

27      reti                ;11 SPI, STC Serial Transfer Complete
28      reti                ;12 UART Rx Complete
29      reti                ;13 UART Data Register Empty
30      reti                ;14 UART Tx Complete
31      reti                ;15 ADC Conversion Complete
32      reti                ;16 EEPROM Ready
33      reti                ;17 Analog Comparator
34      reti                ;18 TWI (I²C) Serial Interface
35      reti                ;19 Store Program Memory Ready
36      ;-----
37
38 main:      ldi      r16,lo8(RAMEND)
39            out      SPL,r16
40            ldi      r16,hi8(RAMEND)
41            out      SPH,r16
42            ldi      r16,0b11111111 ; Gesamter DDRB auf Ausgabe
43            out      DDRB,r16      ; stellen
44
45      ;-----
46 mainloop: ldi      r16,0b00000001 ; Stelle 0 des PORTB = 1
47            out      PORTB,r16    ; schaltet die Leuchtdiode ein
48            rjmp     mainloop     ; diese Schleife wird ständig wiederholt
49

```

Abbildung 20: Ausschnitt aus dem Editor des Workpad. Die neuen Zeilen sind umrandet dargestellt

Die neuen Programmzeilen zeigt der umrandete Teil des Programms; die anderen Programmzeilen entstammen dem „Grundgerüst“.

4.9 Übung 1

Erstellen Sie ein Programm, das alle drei auf dem Board befindlichen Leuchtdioden einschaltet. Die Leuchtdioden sollen mit dem PORTC an den Stellen 0, 2 und 3 verbunden werden.

4.10 Darstellung eines Programms in einem Programmablaufplan PAP

Das eben entwickelte Programm zur Ansteuerung einer Kontrollleuchte war recht übersichtlich. Werden Programme umfangreicher, dann kann eine graphische Darstellung die Übersicht wesentlich erleichtern. Dazu kann man sich eines Programmablaufplans bedienen. Programme für Programmablaufpläne finden Sie auch im Internet. Eine Fundstelle ist:

<http://www.gso-koeln.de/papdesigner/Download.html>

Bitte installieren Sie dieses Programm auf Ihrem Computer, es ist kostenfrei.

Es ist leicht zu installieren und für fast alle Computer und Betriebssysteme verwendbar. Ist es installiert, werden Sie sehen, dass es sehr leicht zu bedienen ist und Ihre Programme fortan sehr gut dokumentieren wird. In den Übungen werden fortan zu allen Assemblerprogrammen auch die entsprechenden Programmablaufpläne erscheinen.



Abbildung 21: Die Programm-dokumentation mit Hilfe des PAP-Designers

Ziel des Programmablaufplans ist es, jemandem mit wenigen graphischen Elementen den Zusammenhang des Programmablaufs und die Entwicklungsschritte der Programmentwicklung klar zu stellen. Dementsprechend wird auch der Text in den einzelnen graphischen Darstellungen keine direkte Übersetzung eines Assembler-Programms darstellen, sondern vielmehr den Ablauf oder eine Funktion kurz beschreiben. Vergleichen Sie bitte den im Bild dargestellten Programmablaufplan mit dem Assemblerprogramm. Beides zusammen gibt einem „Außenstehenden“ Klarheit.

Elemente des Programmablaufplanes PAP








Element des PAP	Bedeutung	Erklärung allgemein
	Allgemeine Anweisung	Prozess
	Datenein- und ausgaben	Daten
	Unterprogramm	Vordefinierter Prozess
	Entscheidung	Verzweigung
	Verbindungsstelle	Verbindungsstelle
	Programmstart und Programmende	Grenzstelle
	Endlosschleife	

Abbildung 22: Allgemein gültige Symbole zur Programmdarstellung im Programmablaufplan

4.11 Zweites Programm in Assembler „Taste steuert LED“

Wir haben im Programm „LED steuern“ eine an PORTB.0 angeschlossene LED mit einem 1 Signal angesteuert. Jetzt wollen wir das etwas erweitern und die LED durch einen Taster ansteuern.

Anmerkung: Zur Erinnerung, es sind nicht alle 8-Bit eines PORTs für den Programmierer frei zugänglich. Bei PORTD fehlen die Bits 0 und 1, bei PORTB und C fehlen die Stellen 6 und 7. Das sollten Sie bei der folgenden Programmentwicklung berücksichtigen.



Abbildung 23: Der Anschlussplan des Tasters und der Leuchtdiode am AVR-Board. Der Pullup-Widerstand muss zusätzlich verdrahtet werden.

4.11.1 Anschluss der Hardware

Zunächst muss der Anschluss für den Taster und für die Leuchtdiode festgelegt werden. Empfehlenswert ist hier, zwei gleich aufgebaute PORTs zu verwenden. Wenn Sie in das Anschlussbild oben sehen, fällt sicher gleich auf, dass bei PORTB und PORTC jeweils die Bits 6 und 7 fehlen. Sie sind demnach identisch aufgebaut. Wählen wir PORTB für die Eingabe also den Anschluss des Tasters, dann können wir PORTC für den Anschluss der Leuchtdiode verwenden. Die Verbindungen finden Sie oben im Bild eingezeichnet.

Taster und LED sind jeweils an Bit 5 der PORTs angeschlossen. Das hat den Sinn, dass der eingelesene Wert (8Bit) an PORTB direkt über ein Register an den PORTC ausgegeben werden kann.

4.11.2 Pullup-Widerstand

Der Taster – wir haben das schon angemerkt – ist nur einpolig und deshalb nur mit dem 0-Potential verbunden. Ein offener – also nicht betätigter – Taster würde ein unbestimmtes Signal an den Eingang liefern, und das ist nicht gut.



Abbildung 24: Die Programmdokumentation mit Hilfe des PAP-Designers

Deshalb ziehen wir durch den Pullup-Widerstand das Potential auf 5V. Der Widerstandswert kann ca. 2 kOhm betragen. Der Widerstand ist nicht auf dem Board installiert. Er muss also zusätzlich angeschlossen werden. Wir kommen auf diesen Widerstand noch einmal zurück, denn dieser kann im AVR auch programmiert werden. Dazu aber später mehr.

4.11.3 Der Programmablaufplan

Wir verwenden wieder das Grundgerüst des Assemblers und stellen das im PAP nicht dar.

Im ersten Programmschritt, wieder im Abschnitt main, wird die Datenflussrichtung im PORTB und PORTC festgelegt.

Im Hauptprogramm – mainloop – wird kontinuierlich der Wert des PORTB in das Register R16 eingelesen und an den PORTC weitergegeben. Sicher nicht die eleganteste Art der Tastenabfrage, für die ersten Programmschritte aber vollkommen ausreichend.

Ein Befehl fehlt momentan noch zur Umsetzung des Programms. Das ist der Befehl für das Einlesen eines Wertes, der an einem PORT ansteht.

4.11.4 Der IN-Befehl

Befehl	Operand 1	Operand 2
IN	RegX	PINB, PINC, PIND
Eingabe	Reg16....Reg31	PORTB, PORTC, PORTD

Es klingt etwas seltsam, dass bei der Dateneingabe von einem PORT, dieser plötzlich PIN heißt. Aber so haben es die Entwickler nun mal festgelegt. Das Zielregister als ersten Operanden, kann wieder ein Register zwischen R16 und R31 sein. Quelle der Einleseaktion können die PORTs B, C oder D sein.

Für unser Beispiel würde sich folgende Einlesezeile ergeben

```
in    r16, PINB           ;Einlesen des PORTB durch PIN!
```

Damit wird der Wert, der an PORTB ansteht, in das Register R16 übertragen.

Rufen Sie nun das Workpad auf und vollziehen Sie die Programmentwicklung mit uns.

Wie zeigen von jetzt an nur den Textausschnitt des Editors.


```

;-----
main:    ldi     r16,lo8(RAMEND)
        out     SPL,r16
        ldi     r16,hi8(RAMEND)
        out     SPH,r16
        ldi     r16,0b00000000    ;Konstante für Eingabe
        out     DDRB,r16          ;an PORTB
        ldi     r16,0b11111111    ;Konstante für Ausgabe
        out     DDRC,r16          ;an PORTC
;-----
mainloop: wdr
        in      r16,PINB          ;Einlesen des PORTB durch PIN!
        out     PORTC,r16        ;Ausgabe des Wertes an PORTC
        rjmp    mainloop
;-----

```

In „main“ werden die Datenflussrichtungen der PORTs B und C festgelegt.

In „mainloop“ wird der an PORTB anstehende Wert in das Register R16 übertragen und dann durch den OUT-Befehl an PORTC übergeben.

Das ist schon alles. Sie können nun das Programm in das AVR-Entwicklungsboard übertragen und testen.

4.11.5 Internen Pullup-Widerstand setzen

Für einen einpoligen Taster oder Schalter einen extra Pullup-Widerstand einzulöten ist für viele Anwendungen umständlich. Aus diesem Grund haben die Entwickler des Mikrocontrollers diesen Widerstand schon im Mikrocontrollerchip integriert. Da der Widerstand nicht immer benötigt wird, kann man ihn durch eine Befehlsfolge im Assemblerprogramm setzen (einschalten) oder rücksetzen (ausschalten).

Zuerst muss der PORT auf Eingabe programmiert werden. In unserem Beispiel wurde das durch folgende Programmzeile realisiert:

```
ldi     r16,0b00000000    ;Konstante für Eingabe
out     DDRB,r16          ;an PORTB
```

Jetzt sind alle Bits des PORTB auf Eingabe programmiert. Wird jetzt eine Datenausgabe auf den PORTB in Form von 1-Signalen ausgegeben, wird der interne Pullup-Widerstand bzw. die internen Pullup-Widerstände eingeschaltet.

Wir setzen also diese Zeile in das Programm ein:

```
ldi     r16,0b00000000    ;Konstante für Eingabe
out     DDRB,r16          ;an PORTB
ldi     r16,0b11111111    ;alle Pullup Widerstände an PORTB setzen
out     PORTB,r16
```

Damit entfällt das externe Einsetzen eines Widerstands.

Das vollständige Programm „Taster steuert LED“ sehen Sie im folgenden Bild.

```

01 ;+-----+
02 ;| Title       : Taster steuert LED
03 ;+-----+
04 ;| Funktion    : ...
05 ;| Schaltung   : ...
06 ;+-----+
07 ;| Prozessor   : ATmega8
08 ;| Takt        : 3,6864 MHz
09 ;| Sprache     : Assembler
10 ;| Datum       : 20.06.2011
11 ;| Version     : 01
12 ;| Autor      : Edgar Hoch
13 ;+-----+
14 .include      "AVR.H"
15 ;+-----+
16 ;Reset and Interrupt vector          ;VNr. Beschreibung
17         rjmp   main                 ;1  POWER ON RESET
18         reti   ;2  Int0-Interrupt
19         reti   ;3  Int1-Interrupt
20         reti   ;4  TC2 Compare Match
21         reti   ;5  TC2 Overflow
22         reti   ;6  TC1 Capture
23         reti   ;7  TC1 Compare Match A
24         reti   ;8  TC1 Compare Match B
25         reti   ;9  TC1 Overflow
26         reti   ;10 TC0 Overflow
27         reti   ;11 SPI, STC Serial Transfer Complete
28         reti   ;12 UART Rx Complete
29         reti   ;13 UART Data Register Empty
30         reti   ;14 UART Tx Complete
31         reti   ;15 ADC Conversion Complete
32         reti   ;16 EEPROM Ready
33         reti   ;17 Analog Comparator
34         reti   ;18 TWI (I2C) Serial Interface
35         reti   ;19 Store Program Memory Ready
36 ;+-----+
37 ;Start, Power ON, Reset
38 main:    ldi    r16,lo8(RAMEND)
39         out    SPL,r16
40         ldi    r16,hi8(RAMEND)
41         out    SPH,r16
42         ldi    r16,0b00000000      ;Konstante für Eingabe
43         out    DDRB,r16           ;an PORTE
44         ldi    r16,0b11111111     ;alle Pullup-Widerstände an PORTE setzen
45         out    PORTE,r16
46         ldi    r16,0b11111111     ;Konstante für Ausgabe
47         out    DDRC,r16           ;an PORTC
48 ;+-----+
49 mainloop: wdr
50         in     r16,PINB           ;Einlesen des PORTE durch PIN!
51         out    PORTC,r16         ;Ausgabe des Wertes an PORTC
52         rjmp   mainloop
53 ;+-----+

```

4.12 Zusammenfassung



- Für die Programmentwicklung ist es notwendig, sich für eine Programmiersprache zu entscheiden. Häufig verwendete Programmiersprachen sind Assembler oder C++.
- Der Assembler bietet dem Programmierer Befehle an, die in sogenannter Mnemonik dargestellt werden. Beispielsweise die Mnemonik „ldi“, die für load immediate steht, also lade direkt.
- Ein Assemblerbefehl besteht aus dem Befehl selbst und ein oder mehrere Operatoren.
- Beim Schreiben von Programmen in Assembler wird eine bestimmte Reihenfolge eingehalten. Von links nach rechts geschrieben, bedeutet das **Sprungmarke, Befehl, Ziel, Quelle, Kommentar**. Dabei sind Ziel und Quelle beispielsweise die Operatoren.
- Bisher behandelte Befehle des Assemblers sind LDI, IN, OUT und rjmp.
- Die Schreibweise der Befehle kann in klein oder Großbuchstaben erfolgen. Lediglich bei der Schreibweise von PORT oder PIN müssen Großbuchstaben verwendet werden.
- Beim Anschluss eines Tasters ist darauf zu achten, dass kein undefiniertes Eingangssignal entsteht. Einpolige Taster müssen mit einem Pullup-Widerstand versehen werden, damit keine undefinierten Signale entstehen.
- Der Pullup-Widerstand muss nicht als externes Bauelement mit dem Taster angeschlossen werden. Dieses Bauelement kann im Assemblerprogramm durch eine Befehlsfolge aktiviert werden.

Jetzt sind Sie wieder einmal gefordert. Bearbeiten Sie folgende Übung.

4.13 Übung 2

Erstellen Sie ein Programm, das mit zwei Tastern zwei Leuchtdioden auf dem Board steuern kann.



Schließen Sie die Taster an PORTC.0 und PORTC.1 an. Die Leuchtdioden sollen an PORTB.0 und PORTB.1 angeschlossen werden

- Schreiben Sie einen Programmablaufplan mit dem Papdesigner
- Schreiben Sie das Assemblerprogramm mit dem Workpad
- Überprüfen Sie das Programm mit dem Entwicklungsboard



4.14 Lernerfolgskontrolle

Damit Sie sicher sind, dass Sie die bisherigen Ausführungen und Beschreibungen auch verstanden haben, möchten wir Ihnen ein paar Fragen stellen. Gegebenenfalls sollten Sie im Studienbrief noch einmal eine entsprechende Stelle nachlesen, wenn Sie die Antwort nicht gleich parat haben.

- Was benötigen Sie, um einen Mikrocontroller zu programmieren?
- Benennen Sie die Befehlsgruppen des AVR
- Was müssen Sie beachten, wenn Daten aus einem PORT eingelesen werden sollen?
- Können Daten durch einen Ladebefehl direkt in einen PORT transferiert werden?
- Welche graphische Darstellung wird für eine Eingabe oder Ausgabe von Daten im PAP verwendet?
- Welche Aufgaben hat ein Assembler. Geben Sie eine kurze Beschreibung.
- Welche Aufgabe hat das Datendirektionsregister DDR? Geben Sie eine kurze Beschreibung.
- Geben Sie alle Befehle wieder, die notwendig sind, um Daten von PORTB in das Register R16 zu übertragen.

5 Mikrocontroller Praxis 2

In diesem Teil des Studienbriefes widmen wir uns den sogenannten Bitmanipulationsbefehlen des Assemblers. In den letzten beiden Programmbeispielen sind als Aktor bzw. als Sensor ein Taster bzw. eine LED verwendet worden. Um beispielsweise die LED zu aktivieren, reicht ein Bit eines PORTs völlig aus. Die bisher besprochenen Befehle bezogen sich jedoch immer auf alle 8 Bits eines Registers oder eines PORTs. Das kann in Steuerungsaufgaben teilweise hinderlich sein.

Für Steuerungsaufgaben ist es meist sinnvoller, Befehle zu verwenden, die sich auf genau ein Bit eines Registers oder eines PORTs beziehen. Diese Befehle sollen im Folgenden untersucht und angewendet werden.

5.1 Bitmanipulationsbefehle

Bitmanipulationen, damit ist das Setzen oder Rücksetzen eines Bits gemeint, können in Registern und in PORTs stattfinden. In einem Befehl wird das Register oder der PORT genannt, welche Stelle die Operation betrifft und was mit dieser Stelle gemacht werden soll. Aber sehen wir uns das im Einzelnen genauer an.

5.1.1 Setzen/Rücksetzen eines Bits in einem Register oder PORT

Der Setzbefehl lautet:

<i>Befehl</i>	<i>Operand 1</i>	<i>Operand 2</i>
SBI	RegX oder PORTX	0..7
Set Bit in Setzte ein Bit in...	Register R16...R31 oder PORTB, PORTC, PORTD oder DDRB, DDRC, DDRD oder andere Register	Stelle 0 bis 7

Ein Bit eines Registers oder eines PORTs wird auf den Wert 1 gesetzt. Als ersten Operanden wird das Register oder der PORT benannt, als zweiten Operanden wird die Stelle genannt, auf die der Befehl wirkt.

Wichtig: Die Bitmanipulationsbefehle lassen sich nur auf Register und PORTs anwenden, die eine hexadezimale Adresse kleiner 0x32 besitzen. Wir werden in den folgenden Betrachtungen dies berücksichtigen.

Das ist momentan noch nicht von großer Bedeutung. Bei der Anwendung der Register zum Analog-Digital-Umsetzer schon, denn diese liegen außerhalb des angesprochenen Bereichs.

Der Rücksetzbefehl lautet:

Befehl	Operand 1	Operand 2
CBI	RegX oder PORTX	0..7
Clear Bit in Rücksetze ein Bit in...	Register R16...R31 oder PORTB, PORTC, PORTD oder DDRB, DDRC, DDRD oder andere Register	Stelle 0 bis 7

Ein Bit eines Registers oder eines PORTs wird auf den Wert 0 gesetzt. Als erster Operand wird das Register oder der PORT benannt, als zweiter Operand wird die Stelle genannt, auf die der Befehl wirkt.

5.1.2 Programmentwicklung 1

Beispiel:

Im ersten Programmbeispiel haben wir eine Leuchtdiode an PORTC angeschlossen und zwar an der Stelle 2. Durch das Laden einer Konstanten in ein Register und anschließende Übergabe des Registerwertes mit dem OUT-Befehl wurde die Leuchtdiode aktiviert. Mit dem Bitmanipulationsbefehl SBI kann das eleganter gelöst werden. Es wird nur ein einziger Befehl benötigt, der wie folgt programmiert wird:

```
sbi          PORTC,0 ;Bit0 PORTC auf Wert 1 - LED einschalten
```

Damit kann die Leuchtdiode, die an Bit 0 des PORTC angeschlossen ist, aktiviert werden.

Natürlich erspart uns der Bitmanipulationsbefehl nicht, das zum PORT zugehörige DDR für Datenein- oder -ausgabe zu deklarieren. Aber man kann auch hierfür ebenfalls Bitmanipulationsbefehle einsetzen.

Das vollständige Programm zur Aktivierung einer Leuchtdiode sieht nun durch Einfügen von zwei Zeilen ins Grundgerüst wie folgt aus.

```

01 ;+-----+
02 ;| Title      : LED aktivieren durch Bitmanipulation
03 ;+-----+
04 ;| Funktion   : ...
05 ;| Schaltung  : ...
06 ;+-----+
07 ;| Prozessor  : ATmega8
08 ;| Takt       : 3,6864 MHz
09 ;| Sprache    : Assembler
10 ;| Datum      : 21.6.2011
11 ;| Version    : ...
12 ;| Autor      : Edgar Hoch
13 ;+-----+
14 .include     "AVR.H"
15 ;+-----+
16 ;Reset and Interrupt vector      ;VNr. Beschreibung
17     rjmp     main                ;1  POWER ON RESET
18     reti     ;2  Int0-Interrupt
19     reti     ;3  Int1-Interrupt
20     reti     ;4  TC2 Compare Match
21     reti     ;5  TC2 Overflow
22     reti     ;6  TC1 Capture
23     reti     ;7  TC1 Compare Match A
24     reti     ;8  TC1 Compare Match B
25     reti     ;9  TC1 Overflow
26     reti     ;10 TC0 Overflow
27     reti     ;11 SPI, STC Serial Transfer Complete
28     reti     ;12 UART Rx Complete
29     reti     ;13 UART Data Register Empty
30     reti     ;14 UART Tx Complete
31     reti     ;15 ADC Conversion Complete
32     reti     ;16 EEPROM Ready
33     reti     ;17 Analog Comparator
34     reti     ;18 TWI (I2C) Serial Interface
35     reti     ;19 Store Program Memory Ready
36 ;+-----+
37 ;Start, Power ON, Reset
38 main:   ldi     r16,lo8(RAMEND)
39         out     SPL,r16
40         ldi     r16,hi8(RAMEND)
41         out     SPH,r16
42 ;+-----+
43         sbi     DDRC,0            ;Bit0 für PORTC auf Ausgabe setzen
44 ;+-----+
45 mainloop: wdr
46         sbi     PORTC,0          ;Bit0 PORTC auf Wert 1 - LED einschalten
47         rjmp    mainloop
48 ;+-----+
49

```

Wenn Sie das Programm starten, wird die angeschlossene Leuchtdiode eingeschaltet.

5.1.3 Bedingte Sprungbefehle mit Skip

Sollen Werte, die extern oder intern an einem PORT oder einem Register des Mikrocontrollers anliegen, interpretiert werden, können dafür Sprungbefehle verwendet werden. Recht einfache Sprungbefehle sind die Skip-Befehle.

Der Befehlsaufbau

Befehl	Operand 1	Operand 2
SBIC	RegX oder PORTX	0..7
Skip if Bit ist cleared Sprung, wenn das genannte Bit=0 ist	Register R16...R31 oder PORTB, PORTC, PORTD oder DDRB, DDRC, DDRD oder andere Register	Stelle 0 bis 7

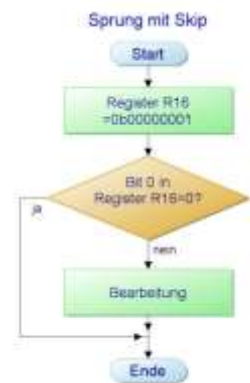


Abbildung 25: Grafische Darstellung des Skip-Befehls

Die Sprungweite ist allerdings begrenzt, genauer gesagt: der nächste folgende Befehl im Listing wird übersprungen. Die Sprungweite ist also nur ein Befehl,

die Sprungrichtung ist immer vorwärts. Will man im Programm an eine beliebige Stelle springen, muss eine Kombination von Skip und rjmp angewendet werden.

Als Ursache dafür, ob ein Sprung ausgeführt wird oder nicht, dient ein Bit in einem Register oder einem PORT. Die Darstellung des Sprungbefehls im PAP zeigt das nebenstehende Bild.

Die zum Bild gehörende Programmzeile könnte so lauten:

```
sbic    r16,0    ;Sprung, wenn die Stelle0 in R16=0 ist
```

Der nachfolgende Befehl wird dann übersprungen, wenn die Stelle 0 in R16 = 0 ist. Der nachfolgende Befehl wird ausgeführt, wenn die Stelle 0 in Register R16=1 ist.

Sozusagen die Umkehrung des eben besprochen Befehls ist wie folgt aufgebaut:

Befehl	Operand 1	Operand 2
SBIS	RegX oder PORTX	0..7
Skip if Bit set Sprung, wenn das genannte Bit=0 ist	Register R16...R31 oder PORTB, PORTC, PORTD oder DDRB, DDRC, DDRD oder andere Register	Stelle 0 bis 7

Der nachfolgende Sprung wird ausgeführt, wenn die Bedingung in den Operanden 1 und 2 erfüllt sind, hier also das Bit = 1 ist.

Mit diesen Kenntnissen kann nun ein Programm für eine Tastenabfrage generiert werden. Durch den Skip-Befehl soll die Taste darauf abgefragt werden, ob sie gedrückt oder nicht gedrückt ist.

Vergleichbar ist dieser Assemblerbefehl mit einer *if---then---else*-Anweisung in anderen Programmiersprachen.

Beispielprogramm

Ein Taster soll an PORTB, Stelle0 angeschlossen werden, eine Leuchtdiode schließen wir an PORTB, Stelle 1 an.

Der Taster soll nun abgefragt werden, ob er betätigt ist oder nicht. Und das mit einem Skip-Befehl.

Den Programmablaufplan finden Sie im Bild.



Abbildung 26: Tasterabfrage mit Hilfe des Skip-Befehls, der einen Befehl überspringen kann

Die Programmschritte und Erklärung:

Zunächst muss die Datenflussrichtung festgelegt werden. Wie immer tun Sie das im DDR. Weil die Dateneingabe auf PORTB und die Datenausgabe auf PORTB erfolgen soll, werden die einzelnen Stellen entsprechend im DDRB eingestellt. Das erfolgt mit dem Bitmanipulationsbefehl cbi oder sbi.

Bevor der Taster abgefragt wird, schalten wir die LED „vorsichtshalber“ erst einmal aus. Die Tasterabfrage erfolgt dann mit Hilfe des Skip-Befehls. Ist die Taste gedrückt, wird im nachfolgenden Befehl die Leuchtdiode aktiviert – ist das nicht der Fall, bleibt die Leuchtdiode ausgeschaltet bzw. der Befehl LED einschalten wird übersprungen.

In ein Assemblerprogramm umgesetzt sieht das dann so aus (wir zeigen fortan nur noch die main und mainloop – ohne Programmkopf und Interrupt-Verkto-Tabelle, damit die Darstellung übersichtlicher wird. (Beim Programmieren verwenden Sie natürlich immer das komplette Grundgerüst).

```

;-----
main:
    ldi    r16,hi8(RAMEND)
    out    SPH,r16
    ldi    r16,lo8(RAMEND)    ;Stack Initialisierung
    out    SPL,r16           ;Init Stackpointer
;-----
    cbi    DDRB,0           ;PORTB0 auf Eingang
    sbi    PORTB,0         ;PORTB0 Pullup
    sbi    DDRB,1           ;PORTB1 auf Ausgang
;-----
mainloop:
    cbi    PORTB,1         ;Wert bei Taste nicht gedrückt
    sbis   PINB,0          ;Taste auswerten
    sbi    PORTB,1         ;Wert bei Taste gedrückt
    rjmp   mainloop
;-----

```

Testen Sie das Programm aus. Sie werden feststellen, dass das Programm einen kleinen Schönheitsfehler hat. Welchen? Nun die Leuchtdiode leuchtet nicht wie erwartet in der vollen Leuchtstärke, sondern etwas dunkler.

Woran liegt das?

Wenn Sie die Programmschritte nochmals verfolgen, dann stellt sich schnell heraus, dass der Programmschritt, in dem die Leuchtdiode ausgeschaltet wird, immer erfolgt – egal, ob die Taste gedrückt ist oder nicht. Das bedeutet, dass

bei gedrückter Taste – zwar nur für eine kurze Zeit- die Leuchtdiode in jedem Schleifendurchlauf einmal ausgeschaltet wird.

5.1.4 Programoptimierung

Die Änderung des Programms zeigt der Programmablaufplan. Hier wird deutlich, dass bei gedrückter Taste das Ausschalten der Leuchtdiode nicht mehr erfolgt.

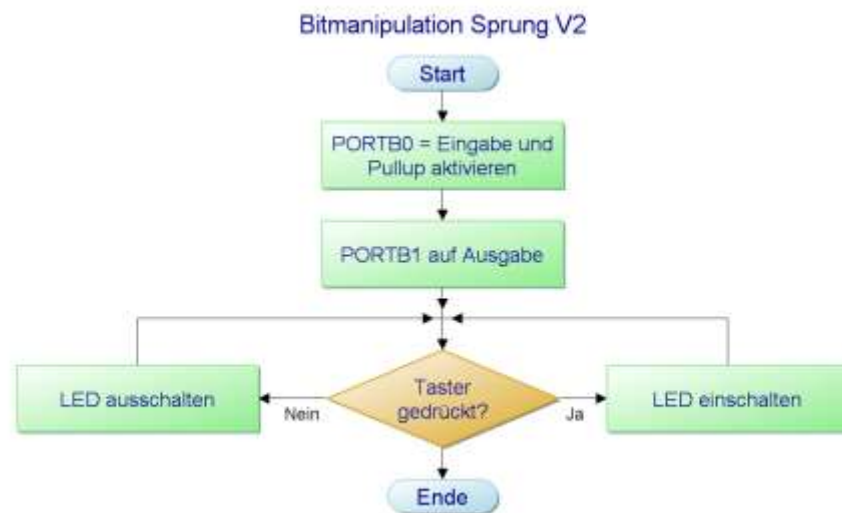


Abbildung 27: Die Leuchtdiode wird in diesem Programm nur ausgeschaltet, wenn die Taste nicht gedrückt ist

Jetzt gilt es, die Schritte des Programmablaufplans in das Assemblerprogramm zu übertragen. Das Einstellen der PORTs kann aus dem vorigen Programm übernommen werden.

```

;-----
main:    ldi     r16, lo8(RAMEND)
        out     SPL, r16
        ldi     r16, hi8(RAMEND)
        out     SPH, r16
  
```

In der mainloop müssen zwei neue Sprungmarken definiert werden. Anders ist das Programm nicht umzusetzen, weil im Gegensatz zum Programmablaufplan im Assemblerprogramm keine parallelen Vorgänge ablaufen können. Sie können an beliebiger Stelle im Assemblerprogramm die beiden Sprungstellen definieren. Beispielsweise so:

```

LEDON:   sbi     PORTB, 1           ;LED einschalten
        rjmp   mainloop           ;Rücksprung zu mainloop
LEDOFF:  cbi     PORTB, 1           ;LED ausschalten
        rjmp   mainloop           ;Rücksprung zu mainloop
  
```

Als Sprungmarken haben wir LEDON und LEOFF gewählt. Die Namen für die Sprungmarken können beliebig gewählt werden. Ist der Vorgang beendet – also die Leuchtdiode ein oder ausgeschaltet – wird durch einen relativen Sprung zurück zur mainloop verzweigt. Beachten Sie bitte, dass der Name der Sprungmarke mit einem Doppelpunkt endet. Bei der Verwendung der Sprungmarke im rjmp muss der Doppelpunkt weggelassen werden. Der Assembler würde diesen Fehler aber melden.

Der Aufruf der Sprungmarken erfolgt nach der Tastenabfrage wie folgt:

```
mainloop:  wdr
           sbic   PINB,0           ;Abfrage ob Taste gedrückt
           rjmp  LEDOFF           ;Bei nicht gedrückter Taste aus
           rjmp  LEDON            ;Bei gedrückter Taste ein
```

Das komplette Programm in der richtigen Befehlsfolge sieht nun so aus:

```
;-----
main:      ldi    r16,lo8(RAMEND)
           out    SPL,r16
           ldi    r16,hi8(RAMEND)
           out    SPH,r16
;-----
           cbi    DDRB,0           ;PORTB Stelle 0 als Eingabe
           sbi    PORTB,0          ;pullup-Widerstand setzen
           sbi    DDRB,1           ;PORTB Stelle 1 als Ausgabe
;-----
mainloop:  wdr
           sbic   PINB,0           ;Abfrage ob Taste gedrückt
           rjmp  LEDOFF           ;Bei nicht gedrückter Taste aus
           rjmp  LEDON            ;Bei gedrückter Taste ein
;-----
LEDON:    sbi    PORTB,1           ;LED einschalten
           rjmp  mainloop         ;Rücksprung zu mainloop
LEDOFF:   cbi    PORTB,1           ;LED ausschalten
           rjmp  mainloop         ;Rücksprung zu mainloop
```

Testen Sie das Programm auf dem Entwicklungsboard aus. Sie werden sehen, dass die Leuchtstärke der Leuchtdiode bei gedrückter Taste jetzt zufriedenstellen ist.

Mag sein, dass Sie den Fehler aus der ersten Programmvariante als nicht so gravierend empfinden. Die Leuchtstärke der LED ist zugegebenermaßen auch nicht so wichtig. Bedenken Sie aber, dass ein solcher Fehler bei der Ansteuerung von weiteren Baugruppen oder externen schnell schaltenden Bauelementen sich wohl viel stärker auswirken kann. Deshalb sollte man solche Fehler im Programm auf jeden Fall vermeiden.



5.2 Übung 3

Programmbeschreibung

Über zwei Taster soll eine LED gesteuert werden. Die LED wird eingeschaltet, wenn der Taster 1 betätigt wird; Die LED wird ausgeschaltet, wenn der Taster 2 betätigt wird. Ganz einfach also. Den Lösungsansatz in Form eines Programmablaufplans für finden Sie im Bild.

Die beiden Taster schließen Sie an PORTB0 und PORTB1 an, die LED soll mit dem PORTB5 verbunden sein.

- Schreiben Sie das Assemblerprogramm mit dem Workpad
- Testen Sie das Programm mit dem Entwicklungsboard

Dieses Programm wird in der Praxis in etwas abgewandelter Form auch bei einer Garagentorsteuerung verwendet, um beispielsweise einen Motor einzuschalten und mit einem Endschalter wieder auszuschalten.

Die Lösung finden Sie im Anhang

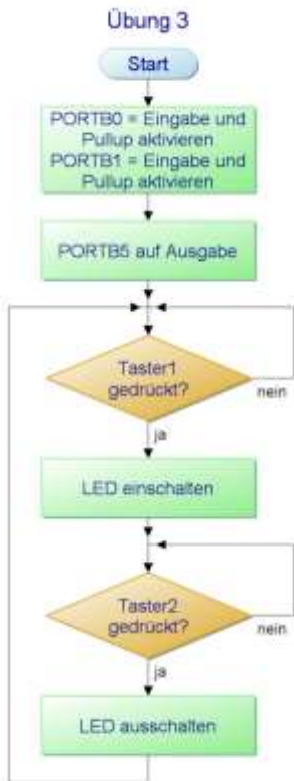


Abbildung 28: Programmablaufplan für das Schalten einer LED mit zwei Tastern

5.3 Weitere Skip-Befehle

Der Vollständigkeit halber wollen wir an dieser Stelle noch zwei weitere Skip-Befehle vorstellen, die sich jedoch nur auf die Register R0 bis R31 beziehen, also auf die PORTs nicht wirksam sind.

5.3.1 SBRC – Skip if Bit in Register is Cleared

Der Sprungbefehl ist dem bislang bekannten SBIC sehr ähnlich. Er ist jedoch nur auf Bitkonstruktionen der Register R0 bis R31 anwendbar. Er wird ausgeführt, wenn ein im Operand 2 angegebenes Bit rückgesetzt ist

Befehl	Operand 1	Operand 2
SBRC	RegX	0..7
Skip if Bit is cleared Sprung, wenn das genannte Bit=0 ist	Register R0...R31	Stelle 0 bis 7

5.3.2 SBRS – Skip if Bit in Register is Set

Der Sprungbefehl wird ausgeführt, wenn ein im Operand 2 angegebenes Bit gesetzt ist.

Befehl	Operand 1	Operand 2
SBRS	RegX	0..7
Skip if Bit set Sprung, wenn das genannte Bit=1 ist	Register R16...R31	Stelle 0 bis 7

5.4 Zusammenfassung



Zu Bitmanipulationsbefehlen finden Sie hier eine kurze Zusammenfassung:

Bitmanipulationsbefehle werden hauptsächlich Steuerungszwecken benutzt. Das ist auch leicht einzusehen, wenn Sie bedenken, dass ein Aktor oder ein Sensor meist mit einer Datenleitung gesteuert werden kann. Beispiele hierfür sind Taster oder Leuchtdioden wie wir sie in den letzten Programmen erfahren haben.

Die wichtigsten Befehle sind die Setz- bzw. Rücksetzbefehle. Sie lauten sbi und cbi. Die Operanden sind dann Register oder PORTs. Der zweite Operand benennt das entsprechende Bit im PORT oder Register.

Neben den Setz- bzw. Rücksetzbefehlen sind die Sprungbefehle sehr wichtig. Aufgrund des Zustandes eines Bits in einem Operanden wird ein Sprung ausgeführt oder nicht. Mit diesen Befehlen können Zustandsabfragen sehr leicht erstellt werden. Wird der Sprung ausgeführt, dann wird der übernächste Befehl abgearbeitet; es kann immer nur vorwärts gesprungen werden.

5.5 Lernerfolgskontrolle



- Erklären Sie kurz, was zu tun ist, wenn in PORTB die Stellen 0, 3 und 5 mithilfe von Setzbefehlen auf Ausgabe zu stellen ist.
- Erklären Sie kurz, wie mit welchen Befehlen eine Tastenabfrage realisiert werden kann. Die Taste wird an PORTC, Stelle 5 angeschlossen.

6 Mikrocontroller Praxis 3

In diesem Teil des Studienbriefes widmen wir uns Programmteilen, die in Programmen immer wieder an verschiedenen Stellen gebraucht und aufgerufen werden müssen. Diese Programmteile möchte man natürlich nicht immer neu schreiben.

Nehmen Sie einmal an, dass Sie eine Kontrolllampe in einem Sekundentakt zum Leuchten bringen wollen. Einen möglichen Programmablaufplan sehen Sie im Bild. Die Zeitverzögerung von einer Sekunde muss in diesem Programm zweimal aufgerufen werden, bzw. programmiert werden. Solche Programme werden Unterprogramme genannt und Sie haben noch einige Besonderheiten aufzuweisen.

Ein weiterer Aspekt ist die Mehrfachverwendung von Programmsequenzen wie Rechenfunktionen, Bedienung der USB-Schnittstelle usw. Programmteile, die also immer wieder verwendet werden oder sogar als Programmteile von anderen Programmieren eingekauft werden.

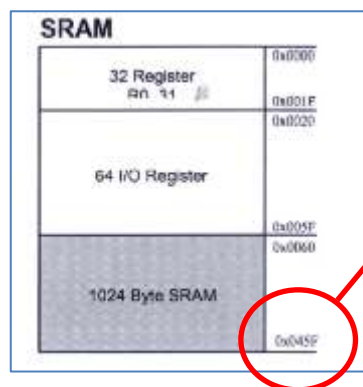


Abbildung 29: Programmablaufplan für den „Sekundentakt“

6.1 Unterprogramme

Unterprogramme – also Programmteile, die in verschiedenen Situationen im Programmablauf immer wieder gebraucht werden – dienen auch der Übersichtlichkeit in der Programmstruktur.

Unterprogramme müssen sich von verschiedenen Stellen aus aufrufen lassen und auch den Weg zurück in den Programmteil finden, der sie aufgerufen hat. Klingt etwas kompliziert, ist es aber nicht.



Auf diese Adresse zeigt der Stackpointer, der das Ende des SRAM signalisiert und durch die vier ersten Befehle in „main“ definiert wird.

Abbildung 30: Speicherteilung des AVR-ATmega8

Wichtiger Helfer beim Zurückfinden in den Programmteil, aus dem ein Unterprogramm aufgerufen wurde, ist der Stack. Kurz zur Erinnerung an unsere Betrachtungen zur Speicherorganisation: der Atmega8, unser verwendeter AVR Controller, besitzt 1024 Byte Speicherplatz im SRAM. Und genau dort im oberen Bereich ist der Stack angesiedelt. Im Stack wird die Rücksprungadresse bei Unterprogrammaufrufen gespeichert.

Bislang haben wir im Grundgerüst ein paar Zeilen zu Beginn von „main“ als selbstverständlich hingenommen und nicht weiter betrachtet. Sehen wir uns diese Zeilen nochmals etwas genauer an – hier sind sie noch einmal:

```
main:    ldi    r16,lo8(RAMEND) ; Definition des Stacks
        out    SPL,r16    ; niederwertiger Teil
        ldi    r16,hi8(RAMEND) ; Definition des Stacks
        out    SPH,r16    ; höherwertiger Teil
```

Diese Programmzeilen definieren das Ende des SRAMs. Die beiden 8-Bit Register SPL und SPH bilden zusammen eine 16-Bit Adresse und zwar genau die Adresse, die Sie in der Speicherdarstellung als letzte Adresse in unserem System finden: 0x045F.

6.1.1 Stackpointer

Diese letzte Adresse des SRAMs, die in den beiden Registern SPL und SPH definiert wird, nennt man Stackpointer. Das ist also ein Zeigerregister, das aus zwei Teilen besteht: dem niederwertigen Teil einer Adresse in SPL und dem höherwertigen Teil in SPH.

Wie es zur Berechnung dieser Adresse kommt, sehen wir uns in einem späteren Kapitel des Themas Mikrorechner-technik an.

6.1.2 Der Stack

Der Stack ist ein Speicherbereich des SRAM, in dem Rücksprungadressen festgehalten werden. Der Stackpointer zeigt auf die aktuelle Adresse im SRAM. Theoretisch könnten alle 1024 Adressen des SRAM für den Stack in Anspruch genommen werden. Sie werden jedoch noch sehen, dass dieser Speicherbereich auch noch für andere Zwecke in Anspruch genommen wird.

6.2 Der Unterprogrammaufruf

Der Aufruf eines Unterprogramms kann durch zwei verschiedene Befehle ausgeführt werden. Der erste Befehl ist „rcall“, der zweite Befehl ist „ical“.

6.2.1 Aufruf mit rcall

Der relative Aufruf eines Unterprogramms erfolgt durch die Angabe der Sprungdistanz von der aufrufenden Adresse bis zur tatsächlichen Adresse des Unterprogramms. Die Berechnung der Distanz übernimmt der Assembler, darum muss man sich nicht kümmern. Wichtig zu wissen ist jedoch, dass die Sprungdistanz beim relativen Unterprogrammaufruf maximal -2048 bis +2048 Adressen betragen. Für größere Distanzen muss der Befehl ical verwendet werden.

Befehl	Operand 1	Operand 2
rcall	Sprungmarke	kein
Relativer Unterprogrammaufruf	Kann beliebig definiert werden, muss jedoch mit einem Buchstaben beginnen	kein

Wird ein Unterprogramm von einer beliebigen Stelle eines Programms – wir nennen es künftig Hauptprogramm – aufgerufen, wird auf dem Stack die Rücksprungadresse automatisch festgehalten und der Stackpointer zeigt auf die nächste beschreibbare Stelle im Stack.

6.2.2 Zurück mit ret

Beim Rücksprung vom Unterprogramm in das Hauptprogramm wird dieser Prozess „rückwärts“ durchgeführt. Die Rücksprungadresse wird aus dem Stack gelesen, in den Programmzähler übergeben und der Stackpointer wird wieder zurückgesetzt. Der entsprechende Befehl lautet:

Befehl	Operand 1	Operand 2
ret	kein	kein
Rücksprung ins Hauptprogramm, Adresse befindet sich im Stack	kein	kein

6.2.3 Aufruf mit icall

Müssen größere Distanzen als +/- 2048 Adressen vom Aufruf des Unterprogramms bis zum Unterprogramm selbst überwunden werden, wird dies mit dem Befehl icall realisiert. Das „i“ steht hier für indirekt und das bedeutet, dass die Adresse durch die Register R30 und R31 gebildet werden. Dieser Befehl kann sozusagen den gesamten Speicherbereich des Mikrocontrollers umfassen.

Die Rückkehr aus dem mit icall aufgerufenen Unterprogramm erfolgt wie beim rcall mit dem Befehl ret.

Befehl	Operand 1	Operand 2
icall	Sprungmarke	kein
Indirekter Unterprogrammaufruf	kein	kein

Diese größeren Distanzen werden in unserem System nicht benötigt. Wir werden deswegen auf icall auch nicht weiter eingehen. Es soll Ihnen lediglich zur Information dienen.

6.2.4 Praktische Umsetzung

Der zu Beginn unserer Betrachtungen gezeigte Programmablaufplan würde mit Verwendung eines Unterprogramms wie folgt aussehen:

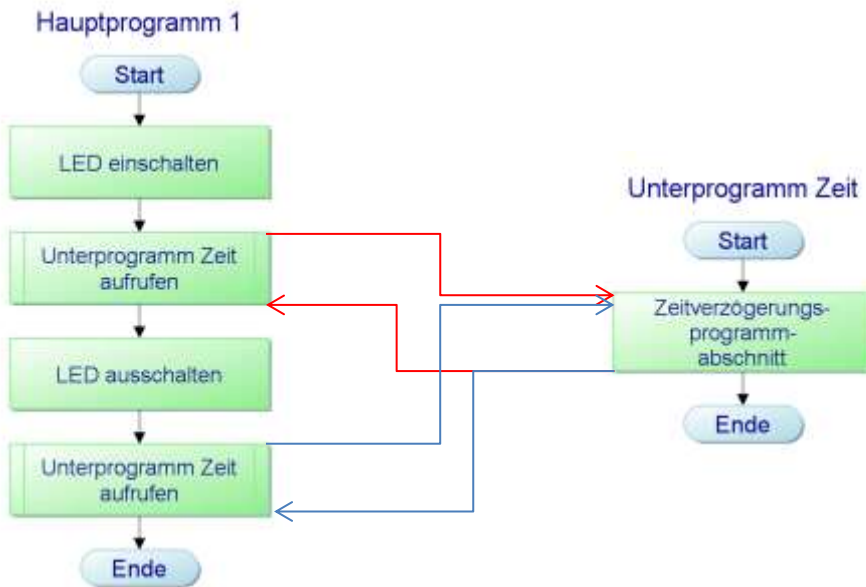


Abbildung 31: Das Hauptprogramm und das Unterprogramm werden in zwei Plänen dargestellt. Die Verbindungslinien werden normalerweise nicht dargestellt – sie dienen nur der ersten Orientierung

Die Umsetzung der Zeitverzögerung können Sie momentan noch nicht entwickeln, das werden wir später tun. Das Programmgerüst soll nun jedoch schon einmal gezeigt werden.

Dazu brauchen wir noch eine Festlegung für den Anschluss der Leuchtdiode:

Die LED soll mit dem PORTB, Stelle 5 verbunden werden.

Damit beginnt das Assemblerprogramm wie folgt (wieder ohne komplettes Grundgerüst dargestellt):

```

main:      ldi      r16,lo8(RAMEND)    ;Stackpointer Initialisierung
          out      SPL,r16
          ldi      r16,hi8(RAMEND)
          out      SPH,r16
          sbi      DDRB,5        ;PORTB5 auf Ausgabe stellen

;-----
mainloop:  wdr
          sbi      PORTB,5       ;Leuchtdiode einschalten
          rcall   Zeit           ;Unterprogramm Zeit aufrufen
          cbi      PORTB,5       ;Leuchtdiode ausschalten
          rcall   Zeit           ;Unterprogramm Zeit aufrufen
          rjmp    mainloop

;-----

Zeit:     .....                ;Programmzeilen für die Zeitver-
          ret                    ;zögerung
          ret                    ;Rücksprung zum Hauptprogramm
  
```

Programmbeschreibung

Sehen Sie sich die Programmzeilen im Assemblerprogramm an. Zu Beginn von „main“ wird der Stackpointer initialisiert. Danach wird PORTB Stelle 5 als Ausgabe definiert.

Im Hauptprogramm „mainloop“ wird zunächst mit dem sbi-Befehl sbi PORTB,5 die LED eingeschaltet. Danach wird die Zeitverzögerung als Unterprogramm aufgerufen. Die Sprungmarke haben wir mit „Zeit:“ gewählt.

Ist das Zeitprogramm abgearbeitet worden, geht das Programm zurück zum Hauptprogramm. Dort wird die LED durch cbi PORTB,5 ausgeschaltet und erneut das Unterprogramm Zeit aufgerufen. Ist das wiederum abgearbeitet, beginnt alles erneut.

Die Programmzeilen für die Zeitverzögerung müssen noch entwickelt werden. Stellen wir das noch ein wenig zurück, wir entwickeln das später.

Z

6.3 Zusammenfassung

- Werden Programmteile oder Programmsequenzen an unterschiedlichen Stellen von Programmen immer wieder gebraucht, werden diese als Unterprogramme geschrieben.
- Bei einem Unterprogrammaufruf wird die Aufrufstelle als Rücksprungadresse vom Mikrocontroller im Stack festgehalten.
- Der Stack muss als Adressbereich definiert werden. Die Endadresse des Stack wird im Stackpointer festgehalten.
- In unserem verwendeten System ist die letzte Hexadezimaladresse im SRAM der Beginn des Stacks.
- Unterprogrammaufrufe werden mit dem Befehl rcall realisiert. Die Distanz zum Unterprogramm darf nicht grösser als +/- 2048 sein.
- Die Berechnung der Sprungdistanz übernimmt der Assembler.
- Das Ende eines Unterprogramms bildet der Rücksprungbefehl ret. Durch ret wird die auf dem Stack automatisch gespeicherte Rücksprungadresse gelesen und in den Programmzähler übergeben.

L

6.4 Lernerfolgskontrolle

- Wozu werden Unterprogramme geschrieben? Kurze Erklärung
- Welche Sprungdistanz zum Unterprogramm kann mit dem Befehl rcall maximal überwunden werden?
- Wie lauten die Befehle für den

Unterprogrammaufruf: _____

Rücksprung zum Hauptprogramm: _____

- Wie wird der Anfang eines Unterprogramms definiert?

7 Mikrocontroller Praxis 4

Kehren wir nach dem „Ausflug“ in die Bitmanipulationen, die für die Steuerungstechnik sehr wichtig sind, zurück zu 8-Bit-Funktionen oder Byte-Funktionen und Byte-Befehle. Dies betrifft vornehmlich Zählerfunktionen, also arithmetische Befehle und logische Befehle. So haben wir diese bei der Aufteilung der Befehlsgruppen ganz zu Beginn unserer Betrachtungen auch benannt.

7.1 Zählbefehle mit Registern

In den frei zugänglichen Registern (R16 bis R31) können Werte aufwärts oder abwärts gezählt werden. Dazu sind die Register mit einem Anfangswert zu laden, von dem aus dann bis zum Maximalwert 0xFF mit der Schrittweite 1 gezählt werden kann. Ist der Maximalwert erreicht, wird der Registerwert automatisch auf den Wert Null gesetzt.

Zum Zählen gibt es zwei Befehle `inc reg` und `dec`.

7.1.1 Inc reg

`Inc reg` addiert zu einem Registerinhalt den Wert 1. Inkrementieren kann also auch mit „Addiere 1 zum derzeitigen Registerinhalt und speichere den aktuellen Wert im Register“ übersetzt werden.

<i>Befehl</i>	<i>Operand 1</i>	<i>Operand 2</i>
Inc	Reg	kein
Inkrementiere den Registerinhalt	R16 – R31	kein

7.1.2 Dec reg

Geradezu umgekehrt funktioniert der Befehl `dec reg`. Hiermit kann der aktuelle Wert eines Registers um den Wert 1 subtrahiert werden. Das Ergebnis wird als neuer Wert im Register gespeichert.

<i>Befehl</i>	<i>Operand 1</i>	<i>Operand 2</i>
Dec	Reg	kein
Dekrementiere den Registerinhalt	R16 – R31	kein

7.2 Das Statusregister SREG

Ergebnisse einer arithmetischen oder logischen Operation werden in einem Statusregister festgehalten, das beim AVR als SREG bezeichnet wird. Das geschieht automatisch, der Programmierer muss sich also nicht bemühen. Jedes einzelne Bit dieses Registers hat eine Bedeutung.

Einige dieser Stellen im SREG sollen nun genauer beleuchtet werden, weil diese Stellen später auch die Schaltstellen bilden, ob Sprungbefehle ausgeführt werden oder nicht.

Aufbau des SREG

SREG							
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
I	T	H	S	V	N	Z	C

Bit 7 → Global Interrupt enable, signalisiert, dass Interrupts zugelassen sind.

Bit 6 → Bit Copy Storage, kann als Zwischenspeicher benutzt werden.

Bit 5 → Half Carry Flag, wird gesetzt, wenn bei einer Rechenoperation ein Übertrag von der Stelle 2^3 auf 2^4 stattgefunden hat.

Bit 4 → Sign bit, zeigt das Vorzeichen einer Zahl an.

Bit 3 → Two’s Complement Overflow Flag, wird gesetzt, wenn bei der Bildung eines 2er Komplements ein Übertrag stattgefunden hat.

Bit 2 → Negative Flag, wird gesetzt, wenn das Rechenergebnis eine negative Zahl ist.

Bit 1 → Zero Flag, wird gesetzt, wenn das Rechenergebnis den Wert Null annimmt.

Bit 0 → Carry Flag, wird gesetzt, wenn an der Stelle 2^7 ein Übertrag stattfindet.

Die für Sie momentan wichtigsten Bits oder auch Flags genannt sind in der ersten drei Bits zu finden: Carry, Zero und Negative

Untersuchung einer Rechenoperation als Beispiel

Nehmen wir an, im Register R16 sei ein Wert 0xFF gespeichert. Zu diesem Wert wird nun eine 1 addiert, dann ist das Ergebnis 0.

1111 1111	Wert im Register R16
+0000 0001	Addition einer Eins
<hr/>	
0000 0000	Ergebnis = 0

In diesem Fall sind die drei Flags – also Bit 0, Bit1 und Bit 2 des SREG – wie folgt gesetzt bzw. rückgesetzt:

- Carry-Flag: gesetzt, weil es einen Übertrag von der Stelle 2^7 gegeben hat.
- Zero-Flag: gesetzt, weil das Ergebnis der Addition Null ist.
- Negative- Flag: nicht gesetzt, weil Null eine positive Zahl ist.

7.3 Bedingte Sprungbefehle in Abhängigkeit des Ergebnisses einer arithmetischen oder logischen Operation

Die Bits im eben beschriebenen SREG können sozusagen als Schaltelemente bei der Ausführung oder Nichtausführung von Sprüngen fungieren. Wir sprechen von bedingten Sprungbefehlen. Bei diesen Sprüngen kann das Programm zu einer beliebigen Sprungmarke verzweigt werden.

Die für uns wichtigsten Sprungbefehle sollen nun kurz erklärt werden.

7.3.1 BRNE Sprung bei not equal

BRNE kann mit „Sprung erfolgt, wenn das Ergebnis nicht Null ist“, übersetzt werden. BR steht hier für Branch. Dieser Sprungbefehl hat folgende Struktur:

Befehl	Operand 1	Flag
BRNE	Sprungmarke	Z
Sprünge zur Sprungmarke, wenn das Ergebnis der letzten arithmetischen oder logischen Operation nicht Null als Ergebnis liefert.	beliebig	Zero

Am Einfachsten kann man ein Sprungbefehl im PAP darstellen wie auf dem Rand dargestellt. Das Register R16 wird mit dem Hexadezimalwert 0x0F geladen und dann mit Hilfe des dec-Befehls um den Wert 1 erniedrigt. Der nachfolgende Sprungbefehl wird immer dann ausgeführt, wenn das Ergebnis noch nicht den Wert Null erreicht hat. Ist das Ergebnis dann nach dem 15ten Schleifendurchlauf Null, dann ist das Programm beendet.

Ein entsprechendes Assemblerprogramm kann wie folgt aussehen:

```
mainloop:  wdr
           ldi    R16,0x0F           ;Register R16 mit Anfangswert
LOOP1:    dec    R16                 ;Registerwert - 1
           brne  LOOP1              ;Sprung, solange Ergebnis nicht 0
ende:     rjmp  ende                ;Programmende
;-----
```

Die Sprungmarke haben wir einfach LOOP1 genannt und ein Programmende ist in den AVR-Befehlssätzen nicht enthalten, deswegen wurde ein unbedingter Sprung auf die Sprungmarke ende gewählt.

Das Programm selbst hat keinen eigentlichen Zweck, es wird lediglich eine Programmschleife abgearbeitet. Zur Abarbeitung des Programms benötigt der Mikrocontroller jedoch Zeit. Wenn wir also noch einmal zu unseren Überlegungen beim Unterprogramm Zeit zurück gehen, dann macht das eventuell schon Sinn. Nur: wie viel Zeit braucht ein Mikrocontroller für die Bearbeitung eines Befehls?



Abbildung 32: Programmablaufplan für die Darstellung einer Programmschleife

Das wollen wir an dieser Stelle mal etwas genauer wissen.

7.3.2 Zeitbetrachtungen

Bei der Bearbeitung eines Befehls ist der Zeitfaktor abhängig von der Taktfrequenz des Systems. Die Taktfrequenz vieler Systeme ist:

$$f=3,68 \text{ MHz}$$

oder anders ausgedrückt:

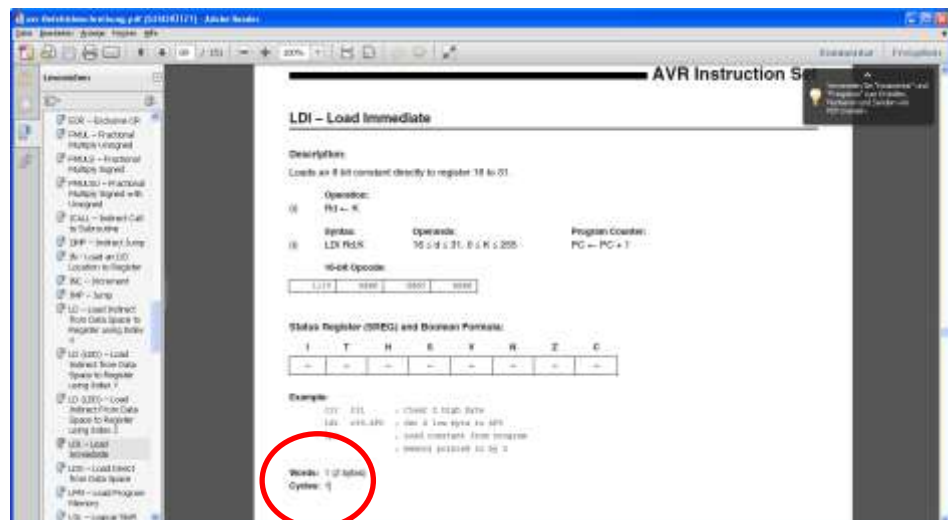
die Zeitdauer für eine Taktperiode berechnet sich mit

$$T = \frac{1}{f} = \frac{1}{3,68 \cdot 10^6 \text{ Hz}} = 0,2717 \mu\text{s}$$

Das ist also die Zeit, die der Mikrocontroller für die Bearbeitung eines Befehlszyklus benötigt.

Will man die Zeitdauer für die Bearbeitung eines Assemblerprogramms wie beispielsweise das eben besprochene berechnen, muss man etwas genauer die einzelnen Befehle hinsichtlich der benötigten Zyklenzahl betrachten. Diese Zyklenzahl finden Sie in der Befehlsbeschreibung des AVR. Einen Ausschnitt finden Sie im folgenden Bild für den LDI-Befehl.

Abbildung 33: Ausschnitt aus der Original-Befehlsbeschreibung für den atmega8 des Herstellers AVR. Diese Beschreibung finden Sie auf der CD zum Studienbrief



Diese Befehlsbeschreibung sollten Sie bei der Programmierung in Assembler eigentlich immer parat haben, wenn Sie einmal etwas nachschlagen wollen.

Betrachtet man nun nochmals die Programmschleife und notiert sich die benötigten Befehlszyklen, kommt man auf folgende Berechnung:

```

LOOP1:   ldi    R16,0x0F    ;benötigt 1 Befehlszyklus
         dec    R16      ;benötigt 1 Befehlszyklus
         brne   LOOP1    ;benötigt bei Ausführung des Sprungs
                                     ;2 Befehlszyklen sonst nur einen.

```

Wenn man die Befehlszyklen für dieses Beispiel zusammen rechnet, kommt man auf folgendes Ergebnis:

1 Befehlszyklus für das Laden des Registers (ldi)

1 Befehlszyklus für das Dekrementieren des Registers, dieser wird aber 15 Mal ausgeführt, also 15 Befehlszyklen (dec)

2 Befehlszyklen für den Sprungbefehl, wenn er ausgeführt wird. Er wird 15 Mal ausgeführt, also benötigen wir 2 mal 15 = 30 Befehlszyklen (brne)

1 Befehlszyklus für den nicht ausgeführten Sprungbefehl am Schluss der Programmschleife (brne)

Also sind es insgesamt **47 Befehlszyklen**, die für das Laden des Registers und die Programmschleife LOOP1 benötigt werden. Als Zeit betrachtet, kommt man auf folgende Berechnung:

$$\text{Gesamtzeit} = T \cdot \text{Befehlszyklen} = 47 \cdot 0,27 \mu\text{s} = 12,69 \mu\text{s}$$

(T= Ausführungszeit für einen Befehlszyklus)

Also immer noch eine recht kurze Zeit, die – wenn eine Zeitverzögerung von einer Sekunde erreicht werden soll – viel zu kurz ist.

Will man zeitliche Abläufe in einem Mikrocontrollersystem sichtbar machen – beispielsweise eine Leuchtdiode in einem definierten Zeitabstand blinken lassen – dann kann man das durch die Beschäftigung des Mikrocontrollers mit mehreren Schleifendurchläufen erreichen. Man bremst den Mikrocontroller sozusagen aus.

Nutzt man den Registerinhalt von R16 in der Schleife voll aus, lädt also das Register mit dem Wert 0xFF dann wird die Zeit schon länger:

$$\text{Gesamtzeit} = T \cdot \text{Befehlszyklen} = T \cdot 2 \cdot (R16) \cdot 3 = 0,27 \mu\text{s} \cdot 2 \cdot 255 \cdot 3 = 413,1 \mu\text{s}$$

7.3.3 Verschachtelung mehrerer Programmschleifen

Das ist die maximale Zeit, wenn das Register R16 voll ausgenutzt wird. Wenn diese Zeit auch nicht ausreicht, hilft eine Schleifenverschachtelung. Eine solche Verschachtelung mit zwei Registern zeigt der Programmablaufplan.

Die Zeiten für die Programmschleifen multiplizieren sich jetzt:

Die innere Schleife (Register 16) benötigt 413,1 Mikrosekunden, das wissen wir. Wird nun diese Schleife durch die äußere Programmschleife 255 mal auf-

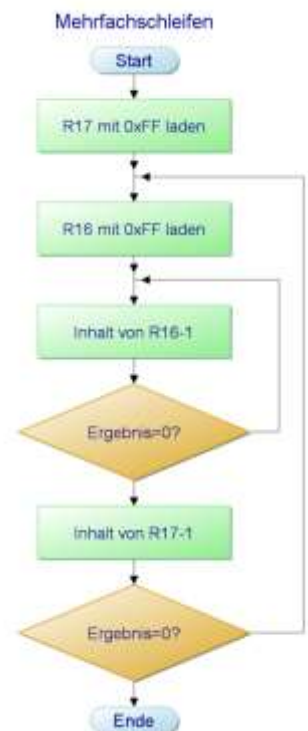


Abbildung 34: Mehrfachprogrammschleifen

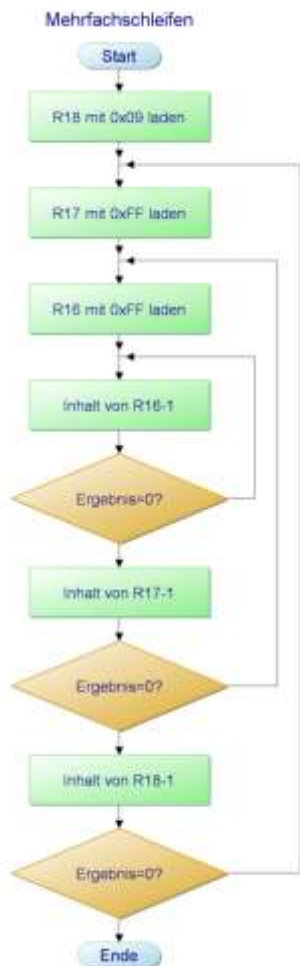


Abbildung 35: Mehrfachprogramm-schleifen. Die dritte Schleife bringt nochmals eine Multiplikation der Zeitwerte

gerufen, haben wir eine Gesamtzeit von ca. 105,34 Millisekunden oder 0,10534 Sekunden. (Bei dieser Betrachtung wurden die Ladezyklen und die Zyklen bei für die 2. Schleife nicht mit berücksichtigt – für unsere Betrachtungen ist das jedoch ausreichend)

Will man auf eine Sekunde Zeitverzögerung kommen, bleibt nur, eine weitere Schleife einzubauen. Eine Lösung finden Sie wieder im Programmablaufplan.

Die eingebaute dritte Zeitschleife bringt noch einmal eine Multiplikation der 0,105 Sekunden aus der zweiten Schleife. Wenn R18 mit der Zahl 9 geladen wir, kommen wir so auf ca. eine Sekunde Zeitverzögerung.

7.4 Übung 4

Bitte setzen Sie den Programmablaufplan nun selbst in ein Assemblerprogramm um und testen Sie das Programm aus. Damit Sie den Test auch verfolgen können, setzen Sie zu Beginn des Programms eine Leuchtdiode. Den PORT für die Leuchtdiode wählen Sie bitte selbst.

Am Programmende schalten Sie die Leuchtdiode wieder aus. Die Leuchtdiode müsste ca. eine Sekunde leuchten und dann wieder dunkel werden.

7.5 Weitere bedingte Sprungbefehle

Neben der Abfrage auf Null gibt es noch weitere bedingte Sprungbefehle. Die für uns wichtigen werden gleich noch erklärt. Die folgende Tabelle, die Sie in der AVR-Befehlsbeschreibung auch finden, gibt eine Übersicht.

Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z*(N ⊕ V) = 0	BRLT ⁽¹⁾	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BRREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE ⁽¹⁾	Rd > Rr	Z*(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO ⁽¹⁾	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BRREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH ⁽¹⁾	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BRREQ	Not zero	Z = 0	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd

Abbildung 36: Tabelle der bedingten Sprungbefehle mit Zuordnung der beeinflussten Flags

7.5.1 BREQ Sprung if equal

Dieser Befehl ist sozusagen die Umkehrung des BRNE. Der Sprung wird ausgeführt, wenn das Ergebnis einer arithmetischen oder logischen Operation den Wert Null liefert. BR steht hier für Branch.

Befehl	Operand 1	Flag
BREQ	Sprungmarke	Z
Springe zur Sprungmarke, wenn das Ergebnis der letzten arithmetischen oder logischen Operation Null als Ergebnis liefert.	beliebig	Zero=1

7.5.2 BRCS Sprung if Carry Set

Dieser Sprung wird ausgeführt, wenn das Carry-Flag nach einer arithmetisch-logischen Operation den Wert 1 annimmt. Ist dies nicht der Fall, wird der Sprungbefehl ignoriert.

Befehl	Operand 1	Flag
BRCS	Sprungmarke	C
Springe zur Sprungmarke, wenn das Ergebnis der letzten arithmetischen oder logischen Operation einen Übertrag liefert.	beliebig	Carry=1

7.5.3 BRCC Sprung if Carry Cleared

Auch wieder eine Umkehrung Befehls BRCS. Der Sprung wird ausgeführt, wenn kein Übertrag an der Stelle 7 auf Stelle 8 stattfindet.

Befehl	Operand 1	Flag
BRCC	Sprungmarke	C
Springe zur Sprungmarke, wenn das Ergebnis der letzten arithmetischen oder logischen Operation keinen Übertrag liefert.	beliebig	Carry=0

Und noch zwei weitere Sprungbefehle, die manchmal recht nützlich sind. Sprungbefehle die auf „kleiner als“ oder „gleich oder größer als“ reagieren. Hierzu wird das Vorzeichenflag N|V und das Carry-Flag C verwendet.

7.5.4 BRLO Sprung if lower

Findet beispielsweise als arithmetische Operation eine Subtraktion statt, kann mit diesem Befehl festgestellt werden, ob Subtrahend größer war als der Minuend. In diesem Fall wird der Sprungbefehl ausgeführt. Hierbei muss beachtet werden, dass der Mikrocontroller im Zweierkomplement die Subtraktion ausführt.

<i>Befehl</i>	<i>Operand 1</i>	<i>Flag</i>
BRLO	Sprungmarke	C
Springe zur Sprungmarke, wenn das Ergebnis der letzten arithmetischen oder logischen Operation einen Übertrag liefert.	beliebig	Carry=1

7.5.5 BRGE Sprung if Greater or Equal

Findet eine Subtraktion oder Addition - beispielsweise zweier Register - statt, dann signalisiert das N|V-Flag, ob beide Werte gleich waren oder der zweite Wert größer war als der erste.

<i>Befehl</i>	<i>Operand 1</i>	<i>Flag</i>
BRGE	Sprungmarke	C
Springe zur Sprungmarke, wenn das Ergebnis der letzten arithmetischen oder logischen Operation Gleichheit oder größer als liefert	beliebig	N V=0

Neben diesen Sprungbefehlen gibt es noch eine ganze Reihe weiterer bedingter Sprungbefehle, die Sie aus der Tabelle entnehmen können. Mit den hier besprochenen bedingten Sprungbefehlen kommen wir in der Programmentwicklung zunächst einmal aus. Bei Bedarf werden wir auf die weiteren Sprungbefehle zurückkommen.

Die folgende Zusammenfassung gibt Ihnen wieder einmal einen Überblick über das Gelernte.

7.6 Zusammenfassung

- Die einfachste Art einer arithmetischen Operation ist das Inkrementieren eines Registerinhalts. Der entsprechende Befehl lautet `inc` und `dec`.
- Die Ergebnisbeschreibung wird im SREG abgelegt. SREG ist ein 8Bit Register in dem jedes einzelne Bit eine Bedeutung hat. Für uns wichtig sind die sogenannten Flags. Drei typische Flags sind das Zero-Flag, das Carry-Flag, und das Neg-Flag.
- Will man aufgrund dieser Ergebnisbeschreibung einen bestimmten Vorgang auslösen, sind hierfür die sogenannten bedingten Sprungbefehle zu verwenden. Von diesen Sprungbefehlen bietet der `atmega8` eine ganze Menge. Für uns momentan wichtig sind die beiden Branch-Befehle `brne` und `breq`. Die Sprünge werden aufgrund der Zustandsbeschreibung des Zero-Flags im SREG ausgeführt oder eben nicht.
- Für die Bearbeitung eines Befehls ist die Taktfrequenz wichtig. Wir werden durch den verwendeten Quartz immer mit einer Taktfrequenz von 3,68MHz arbeiten. Die Zeit für die Abarbeitung einer Taktperiode erhält man, wenn man den Reziprokwert (Umkehrwert) der Frequenz bildet. Bei 3,68MHz ist das eine Zeit von 0,27Mikrosekunden.
- Für die Berechnung der Zeit, die der Mikroprozessor für einen Abschnitt eines Assemblerprogramms zur Bearbeitung benötigt, muss man die Zyklenzahl der Befehle kennen. Die meisten Befehle arbeiten mit einem Zyklus einige, wie beispielsweise Sprungbefehle, benötigen 2 Zyklen zur Bearbeitung.



7.7 Lernerfolgskontrolle

Bitte beantworten Sie nun noch die folgenden Fragen zu diesem Kapitel des Studienbriefs. Die Lösungen finden Sie im Anhang.

- Erklären Sie kurz, was der Befehl `breq LOOP` bewirkt, wenn die vorangegangene arithmetische Operation das Ergebnis Null geliefert hat.
- Welche Bearbeitungszeit benötigt eine Programmsequenz, in der insgesamt 1720 Befehlszyklen ausgeführt werden und die Taktfrequenz des Mikrocontrollers 3,68 MHz beträgt?
- Erklären Sie kurz, wie Sie durch den Befehl `inc Reg 10` Schleifendurchläufe programmieren. Nach den 10 Schleifendurchläufen soll das Programm enden.



8 Mikrocontrollerpraxis 5

Im Teil 5 der Mikrocontrollerpraxis sehen wir uns einige arithmetische Befehle etwas genauer an. Und zum Schluss dieses Studienbriefes werden wir das Zeitverzögerungsprogramm, das wir schon bei der Unterprogrammtechnik begonnen haben, zu Ende führen.

8.1 Arithmetische Befehle

Will man zum Inhalt eines Registers eine 1 addieren, dann kann dies durch die Anwendung des Befehls „INC“ umgesetzt werden. Soll vom Inhalt eines Registers der Wert 1 subtrahiert werden, dann wird der Befehl „DEC“ angewendet, das kennen Sie ja schon. Für die Addition und Subtraktion größerer Werte stehen die Befehle Add, SUB, ADI und SUI zur Verfügung.

8.1.1 Addition mit ADD

Der Inhalte zweier Register werden addiert. Das Ergebnis wird im Zielregister abgelegt.

<i>Befehl</i>	<i>Operand 1</i>	<i>Operand 2</i>
ADD	Rx	Ry
Addiere zum Register Rx den Inhalt von Register Ry, das Ergebnis wird in Rx abgelegt	Beliebiges Register R0 bis R31	Beliebiges Register R0 bis R31
Beeinflusste Flags sind: Z,N,V,C,H		

Auch bei diesen Befehlen wurde die schon bekannte Reihenfolge Befehl, Ziel und Quelle eingehalten. Will man beispielsweise den Inhalt des Registers R17 zum Registerinhalt R18 addieren wird der Befehl wie folgt geschrieben:

```
add    R18, R17    ; Registerinhalt R18+R17 →R18
```

8.1.2 Addition mit ADI

Dieser Befehl addiert zum Inhalt eines als Ziel definierten Registers einen konstanten Wert. I- steht hier also wieder einmal für Immediate also direkt. Der Befehl ist wie folgt aufgebaut.

<i>Befehl</i>	<i>Operand 1</i>	<i>Operand 2</i>
ADI	Rx	k
Addiere zum Register Rx einen konstanten Wert. Das Ergebnis wird in Rx abgelegt	Beliebiges Register R0 bis R31	Konstanter Wert zwischen 0 und 0xFF
Beeinflusste Flags sind: Z,N,V,C,H		

Die Werte für den 2. Operanden kann man wieder in binärer, dezimaler oder hexadezimaler Schreibweise angeben.

Die gleichen Befehle gibt es für die Subtraktion: Subtraktion zweier Register und Subtraktion einer Konstanten von einem Register.

8.1.3 Subtraktion mit SUB

Hierbei ist der Minuend das Zielregister, der Subtrahend ist das Quellregister. Das Ergebnis wird im Zielregister abgelegt. Auch hier wurde also die übliche Schreibweise eingehalten.

Befehl	Operand 1	Operand 2
SUB	Rx	Ry
Subtrahiere vom Register Rx den Wert von Ry. Das Ergebnis wird in Rx abgelegt	Beliebiges Register R0 bis R31	Beliebiges Register R0 bis R31
Beeinflusste Flags sind: Z,N,V,C,H		

8.1.4 Subtraktion mit SUI

Bei der Subtraktion SUI wird vom Inhalt eines Registers ein konstanter Wert abgezogen und das Ergebnis im Register abgelegt. Die Subtraktion wird im Zweierkomplement ausgeführt. Bei der Stellung der Flags sollte dies berücksichtigt werden.

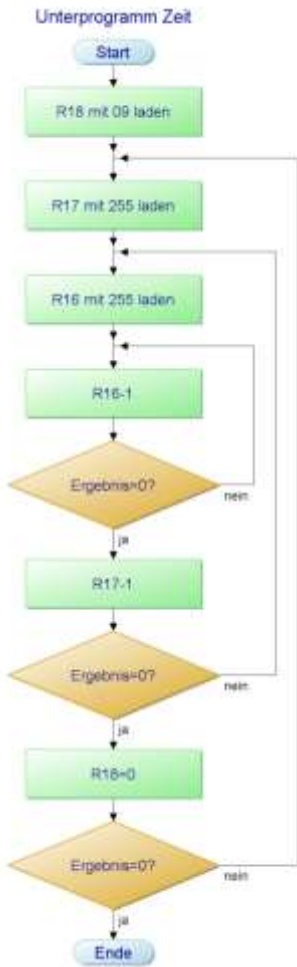
Befehl	Operand 1	Operand 2
SUI	Rx	k
Subtrahiere vom Register Rx einen konstanten Wert. Das Ergebnis wird in Rx abgelegt	Beliebiges Register R0 bis R31	Konstanter Wert zwischen 0 und 0xFF
Beeinflusste Flags sind: Z,N,V,C,H		

8.1.5 Besondere Subtraktion mit CPI

Will man wissen, ob der Inhalt eines Registers genau gleich groß ist wie eine Konstante, dann könnte dies durch die Anwendung des Subtraktionsbefehls SUI erreicht werden. Sind nämlich Minuend und Subtrahend in der Subtraktion gleich groß, wird als Ergebnis Null herauskommen.

Allerdings hat die Subtraktion einen kleinen Nachteil: Der Inhalt des Registers ist nach der Ausführung des Befehls gleich Null. Der alte Wert ist also überschrieben worden.

Eine Subtraktion – sozusagen ohne Ergebnis – kann durch den Befehl CPI erstellt werden. C steht hier für Compare – also Vergleich.



Blinklicht 1 Sekunde



Abbildung 37: Das komplette Programm im PAP: Hauptprogramm und Unterprogramm Zeit

Der Befehlsaufbau ist wie folgt:

Befehl	Operand 1	Operand 2
CPI	Rx	k
Subtrahiere vom Register Rx einen konstanten Wert. Das Ergebnis wird nicht abgelegt	Beliebiges Register R0 bis R31	Konstanter Wert zwischen 0 und 0xFF
Beeinflusste Flags sind: Z,N,V,C,H		

Eine weitere Variante des Vergleichsbefehls ist der Befehl CP, der die Inhalte zweier Register subtrahiert – ebenfalls ohne das Ergebnis abzulegen.

Befehl	Operand 1	Operand 2
CP	Rx	Ry
Subtrahiere vom Register Rx den Wert von Ry. Das Ergebnis wird nicht abgelegt	Beliebiges Register R0 bis R31	Beliebiges Register R0 bis R31
Beeinflusste Flags sind: Z,N,V,C,H		

Soviel zu den arithmetischen Befehlen des AVR atmega8.

Zum Schluss dieses Studienbriefes widmen wir uns noch einmal der Programmentwicklung „Zeitverzögerung“ das Sie ja in der Übung 4 teilweise schon realisiert haben. Wir benötigen „nur“ einen kleinen Umbau. Das heißt, das Zeitverzögerungsprogramm wird als Unterprogramm „umfunktioniert“.

8.2 Programmentwicklung Blinklicht

Das Hauptprogramm

Zu Beginn des Programms muss- wie immer- der PORT definiert werden, mit dem die Leuchtdiode verbunden werden soll. Wir haben hier den PORTB, Stelle 0 gewählt.

Ist der PORT definiert, kann durch einen sbi-Befehl die LED eingeschaltet und das Unterprogramm Zeit aufgerufen werden.

Ist das Unterprogramm Zeit durchgearbeitet worden, wird im Hauptprogramm weitergearbeitet. Genauer gesagt, die LED wird durch einen cbi-Befehl ausgeschaltet.

Nun muss ein zweites Mal das Unterprogramm Zeit bemüht werden, wenn die Ausschaltzeit genauso lang sein soll wie die Einschaltzeit.

Das Unterprogramm Zeit

Das Unterprogramm besteht aus drei in sich verschachtelten Schleifen. Damit multiplizieren sich die Zeiten der Einzelschleifen. Nur so können größere Zeitverzögerungen realisiert werden.

Dass dies eine etwas umständliche und zudem etwas ungenaue Zeitverzögerung ergibt, soll uns hier nicht stören. Sie werden im zweiten Studienbrief genauere und einfachere Methoden kennen lernen.

Das Assemblerprogramm

```

;+-----+
;| Title           : Blinklicht 1 Sekunde
;+-----+
;| Funktion        : ...
;| Schaltung       : ...
;+-----+
;| Prozessor       : ATmega8
;| Takt            : 3,6864 MHz
;| Sprache         : Assembler
;| Datum          : 23.6.2011
;| Version        : 01
;| Autor          : Edgar Hoch
;+-----+
.include "AVR.H"
;+-----+
;Reset and Interrupt vector          ;VNr.Beschreibung
    rjmp    main                    ;1  POWER ON RESET
    reti    ;2  Int0-Interrupt
    reti    ;3  Int1-Interrupt
    reti    ;4  TC2 Compare Match
    reti    ;5  TC2 Overflow
    reti    ;6  TC1 Capture
    reti    ;7  TC1 Compare Match A
    reti    ;8  TC1 Compare Match B
    reti    ;9  TC1 Overflow
    reti    ;10 TC0 Overflow
    reti    ;11 SPI, STC Serial Transfer
    reti    ;12 UART Rx Complete
    reti    ;13 UART Data Register Empty
    reti    ;14 UART Tx Complete
    reti    ;15 ADC Conversion Complete
    reti    ;16 EEPROM Ready
    reti    ;17 Analog Comparator
    reti    ;18 TWI (IC) Serial Interface
    reti    ;19 Store Program Memory Ready
;-----+
;Start, Power ON, Reset
main:    ldi    r16,lo8(RAMEND)      ;Stackpointer Initialisierung
        out    SPL,r16
        ldi    r16,hi8(RAMEND)
        out    SPH,r16
        sbi    DDRB,0              ;PORTB5 auf Ausgabe stellen
        ;Hier Init-Code eintragen.

;-----+-----Hauptprogramm-----+
mainloop: wdr
        sbi    PORTB,0             ;Leuchtdiode einschalten
        rcall  Zeit               ;Unterprogramm Zeit aufrufen

```

```

        cbi     PORTB,0           ;Leuchtdiode ausschalten
        rcall  Zeit             ;Unterprogramm Zeit aufrufen
        rjmp   mainloop

;-----Unterprogramm Zeit-----

Zeit:   ldi     r18,9             ;R18 mit Konstanten laden
loop3:  ldi     r17,255          ;R17 mit Konstanten laden
loop2:  ldi     r16,255          ;R16 mit Konstanten laden
loop1:  dec     r16              ;R16-1
        brne   loop1            ;1.Schleife durchlaufen
        dec     r17              ;R17-1
        brne   loop2            ;2.Schleife durchlaufen
        dec     r18              ;R18-1
        brne   loop3            ;3.Schleife durchlaufen
        ret                    ;zurück zum Hauptprogramm
    
```

Z

8.3 Zusammenfassung

- Für arithmetische Operationen stehen im AVR mehrere Befehle zur Verfügung. Die wichtigsten haben Sie kennen gelernt es sind dies die Addition zweier Register und die Addition eines Registers mit einem konstanten Wert. ADD und ADI sind die mnemonische Bezeichnungen.
- Für die Subtraktion stehen die Befehle SUB und SUI zur Verfügung. Es sollte beachtet werden, dass die Subtraktion in Zweierkomplement-Technik ausgeführt wird, wenn es darum geht, die Flags im SREG zu definieren bzw. für Sprungbefehle auszunutzen.
- Eine Besonderheit der Subtraktionsbefehle sind die Vergleichsbefehle mit CP bzw. CPI. Diese Befehle arbeiten wie die Subtraktionsbefehle, haben jedoch kein Ergebnis. Sie setzen bzw. rücsctzen „nur“ die Flags im SREG.

L

8.4 Lernerfolgskontrolle

- Nehmen Sie an, dass im Register R16 die Hexadezimalzahl 0x12 gespeichert ist und im Register R17 die Hexadezimalzahl 0x0F. Es wird danach die folgende Befehlszeile ausgeführt.

```
ADD    R16,R17
```

Welchen Inhalt haben nach der Befehlsausführung die beiden Register?

- Welchen Wert hat das Register R16, wenn folgender Befehl ausgeführt wird:

```
SUI    R16,0x0A
```

- Welche Zustände haben die Flags C, Z in diesem Fall?

Damit haben Sie den ersten Studienbrief zum Thema Mikrorechnertechnik durchgearbeitet. Sie haben die Grundlagen der Programmierung eines Mikrocontrollers kennen gelernt. Im zweiten Studienbrief widmen wir uns der Programmierung der Baugruppen Analog-Digital-Umsetzer, Timer und Counter und der seriellen Datenübertragung an externe Baugruppen. Weiter werden die Interrupt-Behandlungen und die Interrupt Programmierung ein wichtiges Thema werden.

9 Lösungen zu Lernerfolgskontrollen und Übungen

Lernerfolgskontrolle 1.8

Erarbeiten Sie die Unterschiede „Von-Neumann-Architektur“ und „Harvard-Architektur“.

In der „Von-Neumann-Architektur“ werden alle Baugruppen des Systems mit dem Adressbus, dem Datenbus und dem Steuerbus verbunden. Alle Baugruppen sind demnach von den Bussystemen abhängig. Dies kann sich mitunter auf die Verarbeitungsgeschwindigkeit auswirken. Diese Architektur findet man häufig in Mikrocomputersystemen.

In der „Harvard-Architektur“ wird diese strenge Bushierarchie aufgelöst. Bestimmte Baugruppen werden direkt miteinander verbunden- es können mehrere Bussysteme gleichzeitig bedient werden. Das erhöht für Steuerungsaufgaben die Verarbeitungsgeschwindigkeit. Diese Architektur findet man hauptsächlich bei Mikrocontrollersystemen und Risc-Prozessoren.

Lernerfolgskontrolle 2.9

- Welche Speichertypen werden im Mikrocontroller Atmega8 eingesetzt und welche Aufgaben haben diese?
Es werden die Speichertypen EEPROM, Flash, und SRAM verwendet. Im Flash wird das Programm abgelegt, im EEPROM allgemeine Daten und im SRAM solche Daten, die zur Programmausführung nach Spannungsabschaltung nicht mehr gebraucht werden.
- Welche Baugruppen beinhaltet der Mikrocontroller Atmega8? Listen Sie die Ihnen bis jetzt bekannten Baugruppen und benennen Sie die Aufgaben dieser Baugruppen.
Als wesentliche Baugruppen sind die ALU in ihr finden alle arithmetischen und Logischen Verknüpfungen statt und zugehörige Register und PORTs . Register sind schnelle Speicher mit sehr kurzen Datenübertragungswegen, die auch Ergebnisse aufnehmen. PORTs übernehmen den Datenverkehr zwischen angeschlossenen Aktoren und Sensoren. Als besondere Register sind die Datendirektionsregister DDR zu nennen, welche die Datenflussrichtung bestimmen . Daneben sind die Baugruppen Speicher zu nennen, welche für die Programmspeicherung und die Speicherung von temporären Daten verwendet werden.
- Welche gängigen Taktfrequenzen werden bei Mikrocontrollern eingesetzt?
Das ist recht unterschiedlich – für den Atmega8 wird eine Frequenz zwischen 1MHz und 16MHz gewählt
- Welche Aufgaben haben die Register R16 bis R31?

Die Bestimmung dieser Register obliegt dem Programmierer. Er kann sie frei einsetzen.

- Erklären Sie die Aufgaben des Datendirektionsregisters DDR?
Im DDR wird die Datenflussrichtung bestimmt. Also ob ein PORT und darin besonders die einzelnen Stellen als Dateneingabe oder Datenausgabe arbeiten sollen.
- Welchen Inhalt muss das DDRB haben, wenn die Stellen 2^0 bis 2^6 als Ausgabe und die Stelle 2^7 als Eingabe programmiert werden sollen?
Der Wert ist binär 0xb01111111

Nur der Wert für die Stelle 7 ist eine Null. Null bedeutet Eingabe, Eins bedeutet Ausgabe

Übung 1 (Kap. 4.9)

Erstellen Sie ein Programm, das alle drei auf dem Board befindlichen Leuchtdioden einschaltet. Die Leuchtdioden sollen mit dem PORTC an den Stellen 0, 2 und 3 verbunden werden.

```

;+-----+
;| Title           : Lösung zur Übung 1
;+-----+
;| Funktion        : ...
;| Schaltung       : ...
;+-----+
;| Prozessor       : ATmega8
;| Takt            : 3,6864 MHz
;| Sprache         : Assembler
;| Datum          : ...
;| Version        : ...
;| Autor          : ...
;+-----+
.include          "AVR.H"
;+-----+
;Reset and Interrupt vector          ;VNr.  Beschreibung
      rjmp      main                ;1   POWER ON RESET
      reti     ;2   Int0-Interrupt
      reti     ;3   Int1-Interrupt
      reti     ;4   TC2 Compare Match
      reti     ;5   TC2 Overflow
      reti     ;6   TC1 Capture
      reti     ;7   TC1 Compare Match A
      reti     ;8   TC1 Compare Match B
      reti     ;9   TC1 Overflow
      reti     ;10  TC0 Overflow
      reti     ;11  SPI, STC Serial Transfer C
complete
      reti     ;12  UART Rx Complete
      reti     ;13  UART Data Register Empty
      reti     ;14  UART Tx Complete
      reti     ;15  ADC Conversion Complete
      reti     ;16  EEPROM Ready
      reti     ;17  Analog Comparator
      reti     ;18  TWI (IC) Serial Interface

```

```

                reti                                ;19 Store Program Memory Ready
;-----
main:          ldi    r16,lo8(RAMEND)
              out    SPL,r16
              ldi    r16,hi8(RAMEND)
              out    SPH,r16
              ldi    r16,0b11111111    ; Gesamter DDRB auf Ausgabe
              out    DDRC,r16          ; stellen
;-----
mainloop:     ldi    r16,0b00001101    ; Stelle 0 des PORTB = 1
              out    PORTB,r16        ; schaltet die Leuchtdiode ein
              rjmp   mainloop         ; diese Schleife wird ständig w
iederholt
;-----

```

Übung 2 (Kap.4.12)

Erstellen Sie ein Programm, das mit zwei Tastern zwei Leuchtdioden auf dem Board steuern kann. Schließen Sie die Taster an PORTC.0 und PORTC.1 an. Die Leuchtdioden sollen an PORTB.0 und PORTB.1 angeschlossen werden

- Schreiben Sie ein Programmablaufplan mit dem Papdesigner
- Schreiben Sie das Assemblerprogramm mit dem Workpad
- Überprüfen Sie das Programm mit dem Entwicklungsboard

Lösung:

Assemblerprogramm

```

;+-----+
;| Title           : Übung 2
;+-----+
;| Funktion        : ...
;| Schaltung       : ...
;+-----+
;| Prozessor       : ATmega8
;| Takt            : 3,6864 MHz
;| Sprache         : Assembler
;| Datum          : 20.06.2011
;| Version        : 01
;| Autor          : Edgar Hoch
;+-----+
.include      "AVR.H"
;-----
;Reset and Interrupt vector      ;VNr. Beschreibung
rjmp   main          ;1 POWER ON RESET
reti   ;2 Int0-Interrupt
reti   ;3 Int1-Interrupt
reti   ;4 TC2 Compare Match
reti   ;5 TC2 Overflow
reti   ;6 TC1 Capture
reti   ;7 TC1 Compare Match A
reti   ;8 TC1 Compare Match B
reti   ;9 TC1 Overflow
reti   ;10 TC0 Overflow
reti   ;11 SPI, STC Serial Transfer Complete
reti   ;12 UART Rx Complete
reti   ;13 UART Data Register Empty

```

```

    reti                ;14  UART Tx Complete
    reti                ;15  ADC Conversion Complete
    reti                ;16  EEPROM Ready
    reti                ;17  Analog Comparator
    reti                ;18  TWI (IC) Serial Interface
    reti                ;19  Store Program Memory Ready
;-----
;Start, Power ON, Reset
main:    ldi    r16,lo8(RAMEND)
        out    SPL,r16
        ldi    r16,hi8(RAMEND)
        out    SPH,r16
        ldi    r16,0b00000000    ;Konstante für Eingabe
        out    DDRC,r16          ;an PORTC
        ldi    r16,0b11111111    ;alle Pullup-
Widerstände an PORTC setzen
        out    PORTC,r16
        ldi    r16,0b11111111    ;Konstante für Ausgabe
        out    DDRB,r16          ;an PORTB
;-----
mainloop: wdr
        in     r16,PINB          ;Einlesen des PORTB durch PIN!
        out    PORTC,r16        ;Ausgabe des Wertes an PORTC
        rjmp   mainloop
;-----

```



Lernerfolgskontrolle 4.13

Damit Sie sicher sind, dass Sie die bisherigen Ausführungen und Beschreibungen auch verstanden haben, möchten wir Ihnen ein paar Fragen stellen. Gegebenenfalls sollten Sie im Studienbrief noch einmal eine entsprechende Stelle nachlesen, wenn Sie die Antwort nicht gleich parat haben.

- Was benötigen Sie, um einen Mikrocontroller zu programmieren?
Ein Entwicklungsprogramm wie beispielsweise ein Assembler oder C++. Ev. Noch ein Entwicklungsboard.
- Benennen Sie die Befehlsgruppen des AVR
Arithmetisch-Logische Befehle, Einzelbit-Befehle, Lade- und Transferbefehle, Sprungbefehle
- Was müssen Sie beachten, wenn Daten aus einem PORT eingelesen werden sollen?
Dass der PORT entsprechend im DDR als Einlesekanal deklariert wird und dass das Register PIN statt PORT verwendet werden muss.
- Können Daten durch einen Ladebefehl direkt in einen PORT transferiert werden?
Nein, das geht nicht. Man muss erst ein Register laden und dann den Wert an den PORT übergeben.
- Welche graphische Darstellung wird für eine Eingabe oder Ausgabe von Daten im PAP verwendet?
Das ist die Raute.
- Welche Aufgaben hat ein Assembler. Geben Sie eine kurze Beschreibung.

Er übersetzt die Mnemonik in Binärbefehle, die der Mikrocontroller versteht. Außerdem übernimmt er die Berechnung von Adressen bei Sprungbefehlen.

7. Welche Aufgabe hat das Datendirektionsregister DDR? Geben Sie eine kurze Beschreibung.

Das Register übernimmt die Bestimmung der Datenflussrichtung. Jedes einzelne Bit eines PORTs kann als Eingabe oder Ausgabe programmiert werden.

8. Geben Sie alle Befehle wieder, die notwendig sind, um Daten von PORTB in das Register R16 zu übertragen.

Port B muss im DDRB als Einlesekanal deklariert werden. Das Einlesen der Daten wird dann durch den In-Befehl realisiert.

Übung 3 (Kap 5.2)

Programmbeschreibung

Über zwei Taster sollen soll eine LED gesteuert werden. Die LED wird eingeschaltet, wenn der Taster 1 betätigt wird; Die LED wird ausgeschaltet, wenn der Taster 2 betätigt wird. Ganz einfach also. Den Lösungsansatz in Form eines Programmablaufplans für finden Sie im Bild.

Die beiden Taster schließen Sie an PORTB0 und PORTB1 an, die LED soll mit dem PORTB5 verbunden sein.

- Schreiben Sie das Assemblerprogramm mit dem Workpad
- Testen Sie das Programm mit dem Entwicklungsboard

```
-----  
; Titel          : zwei Taster steuern LED (Übung3)  
-----  
; Funktion      :  
; Schaltung     :  
-----  
; Prozessor     : ATmega8  
; Takt          : 3,6864 MHz  
; Sprache       : Assembler  
; Datum        : 21.6.2011  
; Version      :  
; Autor        : Edgar Hoch  
-----  
.equ    F_CPU, 3686400  
.include "AVR.H"  
-----  
;Reset and Interruptvektoren      ;VNr.  Beschreibung  
begin:  
    rjmp    main                    ; 1    POWER ON RESET  
    reti                    ; 2    Int0-Interrupt  
    reti                    ; 3    Int1-Interrupt  
    reti                    ; 4    TC2 Compare Match  
    reti                    ; 5    TC2 Overflow  
    reti                    ; 6    TC1 Capture  
    reti                    ; 7    TC1 Compare Match A  
    reti                    ; 8    TC1 Compare Match B  
    reti                    ; 9    TC1 Overflow  
    reti                    ;10   TC0 Overflow
```

```
re-
ti          ;11   SPI, STC Serial Transfer Complete
reti       ;12   UART Rx Complete
reti       ;13   UART Data Register Empty
reti       ;14   UART Tx Complete
reti       ;15   ADC Conversion Complete
reti       ;16   EEPROM Ready
reti       ;17   Analog Comperator
reti       ;18   TWI (IC) Serial Interface
reti       ;19   Store Program Memory Read
y
;-----
main:
ldi        r16,hi8(RAMEND)
out        SPH,r16
ldi        r16,lo8(RAMEND) ;Stack Initialisierung
out        SPL,r16        ;Init Stackpointer
;-----
cbi        DDRB,0        ;PORTB0 auf Eingang
sbi        PORTB,0       ;PORTB0 Pullup
cbi        DDRB,1        ;PORTB1 auf Eingang
sbi        PORTB,1       ;PORTB1 Pullup
sbi        DDRB,5        ;PORTB5 auf Ausgang
;-----
mainloop:
LOOP1:     sbic        PINB,0        ;Sprung bei Taste1 gedrückt
rjmp      LOOP1
sbi        PORTB,5       ;LED einschalten
LOOP2:     sbic        PINB,1        ;Sprung bei Taste2 gedrückt
rjmp      LOOP2
cbi        PORTB,5       ;LED ausschalten
rjmp      mainloop      ;Hauptprogramm von vorne starten
;-----
```

Lernerfolgskontrolle 5.5

Erklären Sie kurz, was zu tun ist, wenn in PORTB die Stellen 0, 3 und 5 mithilfe von Setzbefehlen auf Ausgabe zu stellen ist.

Sbi DDRB,0 ; Stellt den PORTB Stelle 0 auf Ausgabe

Sbi DDRB,3 ; Stellt den PORTB Stelle 0 auf Ausgabe

Sbi DDRB,5 ; Stellt den PORTB Stelle 0 auf Ausgabe

Erklären Sie kurz, wie mit welchen Befehlen eine Tastenabfrage realisiert werden kann. Die Taste wird an PORTC, Stelle 5 angeschlossen.

Zunächst muss das DDRC auf Eingabe gestellt werden. Danach kann dann mit dem Skip-Befehl sbic oder sbis der Tastenwert abgefragt werden. Der Sprungbefehl erfolgt immer in Kombination mit einem relativen Sprungbefehl rjmp.

Lernerfolgskontrolle 6.4

- Wozu werden Unterprogramme geschrieben? Kurze Erklärung
Damit Programme und Programmstrukturen übersichtlicher werden und damit Programmteile und Codesequenzen, die an verschiedenen Stellen immer wieder gebraucht werden, nicht immer neu geschrieben werden müssen.
- Welche Sprungdistanz zum Unterprogramm kann mit dem Befehl rcall maximal überwunden werden?
Maximal zwei mal 2048
- Wie lauten die Befehle für den
Unterprogrammaufruf: _____ rcall
Rücksprung zum Hauptprogramm: _____ ret
- Wie wird der Anfang eines Unterprogramms definiert?
Durch eine Sprungmarke – sie darf nicht mit einer Zahl beginnen und endet mit einem Doppelpunkt

Übung 4 (Kap 7.4)

Bitte setzen Sie den Programmablaufplan nun selbst in ein Assemblerprogramm um und testen Sie das Programm aus. Damit Sie den Test auch verfolgen können, setzen Sie zu Beginn des Programms eine Leuchtdiode. Den PORT für die Leuchtdiode wählen Sie bitte selbst.

Am Programmende schalten Sie die Leuchtdiode wieder aus. Die Leuchtdiode müsste ca. eine Sekunde leuchten und dann wieder dunkel werden.

```
;+-----+
;| Title           : Übung 4
;+-----+
;| Funktion        : ...
;| Schaltung       : ...
;+-----+
;| Prozessor       : ATmega8
;| Takt           : 3,6864 MHz
```



```

;| Sprache      : Assembler
;| Datum       : ...
;| Version     : ...
;| Autor       : Edgar Hoch
;+-----+
.include      "AVR.H"
;-----+
;Reset and Interrupt vector          ;VNr.  Beschreibung
    rjmp     main                    ;1   POWER ON RESET
    reti    ;2   Int0-Interrupt
    reti    ;3   Int1-Interrupt
    reti    ;4   TC2 Compare Match
    reti    ;5   TC2 Overflow
    reti    ;6   TC1 Capture
    reti    ;7   TC1 Compare Match A
    reti    ;8   TC1 Compare Match B
    reti    ;9   TC1 Overflow
    reti    ;10  TC0 Overflow
re-
ti
    ;11  SPI, STC Serial Transfer Complete
    reti    ;12  UART Rx Complete
    reti    ;13  UART Data Register Empty
    reti    ;14  UART Tx Complete
    reti    ;15  ADC Conversion Complete
    reti    ;16  EEPROM Ready
    reti    ;17  Analog Comparator
    reti    ;18  TWI (IC) Serial Interface
    reti    ;19  Store Program Memory Ready
;-----+
;Start, Power ON, Reset
main:    ldi    r16,lo8(RAMEND)
        out    SPL,r16
        ldi    r16,hi8(RAMEND)
        out    SPH,r16
        sbi    DDRB,0                ;PORTB auf Ausgabe Bit0
;-----+
mainloop: wdr
        sbi    PORTB,0                ;LED einschalten
        ldi    R18,0x09                ;R18 mit Anfangswert laden
loop3:   ldi    R17,0xFF                ;R17 mit Anfangswert laden
loop2:   ldi    R16,0xFF                ;R16 mit Anfangswert laden
loop1:   dec    R16                    ;R16-1
        brne   loop1                  ;Sprung, wenn Ergebnis nicht Null
        dec    R17                    ;R17-1
        brne   loop2                  ;Sprung, wenn Ergebnis nicht Null
        dec    R18                    ;R18-1
        brne   loop3                  ;Sprung, wenn Ergebnis nicht Null
        cbi    PORTB,0
ende:   rjmp   ende
;-----+

```

Lernerfolgskontrolle 7.6

- Erklären Sie kurz, was der Befehl `brq LOOP` bewirkt, wenn die vorangegangene arithmetische Operation das Ergebnis Null geliefert hat.
Der Sprung zur Sprungmarke `Loop` wird ausgeführt
- Welche Bearbeitungszeit benötigt eine Programmsequenz, in der insgesamt 1720 Befehlszyklen ausgeführt werden und die Taktfrequenz des Mikrocontrollers 3,68 MHz beträgt?

$$T=0,27 \text{ Mikrosekunden} * 1720=0,46 \text{ Millisekunden}$$

- Erklären Sie kurz, wie Sie durch den Befehl `inc` 10 Schleifendurchläufe programmieren. Nach den 10 Schleifendurchläufen soll das Programm enden. Weil das ZeroBit in den Befehlen `breq` oder `brne` abgefragt wird, muss das Register, in dem der `inc` Befehl ausgeführt wird, auf einen Anfangswert geladen werden, dass es nur noch 10 Schritte sind, bis der Wert Null erreicht wird. In einer Dezimalzahl ausgedrückt, muss das Register mit dem Anfangswert 245 geladen werden. Damit sind es noch 10 Schritte bis der Wert Null erreicht wird und der Sprungbefehl entsprechend reagieren kann.

Lernerfolgskontrolle 8.4

- Nehmen Sie an, dass im Register R16 die Hexadezimalzahl 0x12 gespeichert ist und im Register R17 die Hexadezimalzahl 0x0F. Es wird danach die folgende Befehlszeile ausgeführt.

```
ADD R16,R17
```

Welchen Inhalt haben nach der Befehlsausführung die beiden Register? Register R17 hat nach wie vor den Wert 0x0F, Das Register R16 beinhaltet das Ergebnis der Addition und hat den Wert 0x21.

- Welchen Wert hat das Register R16, wenn folgender Befehl ausgeführt wird:
SUI R16,0x0A
Das Register R16 beinhaltet das Ergebnis der Subtraktion $0x12 - 0x0A = 0x08$
- Welche Zustände haben die Flags C, Z in diesem Fall?
Beide Flags haben den Wert Null. Es gab keinen Übertrag und das Ergebnis der Subtraktion ist nicht Null.

10 Anhänge

10.1 Registerübersicht des AVR atmega8

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x0F (0x0F)	SREG	-	-	-	-	-	-	-	-	3
0x2E (0x2E)	SPH	-	-	-	-	-	SP10	SP9	SP8	11
0x30 (0x30)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
0x3C (0x3C)	Reserved									
0x3E (0x3E)	GICR	INT1	INT0	-	-	-	-	IVSEL	IVCE	47, 63
0x3A (0x3A)	GIFR	INTF1	INTF0	-	-	-	-	-	-	66
0x38 (0x38)	TIMSK	OCIE2	TOIE2	OCIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0	78, 108, 128
0x36 (0x36)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0	71, 101, 128
0x37 (0x37)	SPMCR	SPME	RWWSB	-	RWWSRE	BLSSET	POWRT	POERS	SPMEN	210
0x38 (0x38)	TWCR	TW&T	TWEA	TWSTA	TWST0	TWVC	TWEN	-	TWIE	168
0x36 (0x36)	MCLCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	31, 64
0x34 (0x34)	MUCUSR	-	-	-	-	WDRF	BORF	EXTRF	PORF	39
0x33 (0x33)	TCCR0	-	-	-	-	-	CS02	CS01	CS00	70
0x32 (0x32)	TCNT0	TimerCounter0 (8 Bits)								70
0x31 (0x31)	OSCCAL	Oscillator Calibration Register								29
0x30 (0x30)	SPDR	-	-	-	-	ACME	PUD	PSR2	PSR10	85, 73, 121, 190
0x2F (0x2F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	26
0x2E (0x2E)	TCCR1B	ICN1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	26
0x2D (0x2D)	TCNT1H	TimerCounter1 – Counter Register High byte								29
0x2C (0x2C)	TCNT1L	TimerCounter1 – Counter Register Low byte								29
0x2B (0x2B)	OCR1AH	TimerCounter1 – Output Compare Register A High byte								29
0x2A (0x2A)	OCR1AL	TimerCounter1 – Output Compare Register A Low byte								29
0x29 (0x29)	OCR1SH	TimerCounter1 – Output Compare Register B High byte								29
0x28 (0x28)	OCR1BL	TimerCounter1 – Output Compare Register B Low byte								29
0x27 (0x27)	ICR1H	TimerCounter1 – Input Capture Register High byte								100
0x26 (0x26)	ICR1L	TimerCounter1 – Input Capture Register Low byte								100
0x26 (0x26)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	115
0x24 (0x24)	TCNT2	TimerCounter2 (8 Bits)								117
0x23 (0x23)	OCR2	TimerCounter2 Output Compare Register								117
0x22 (0x22)	ASCR	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB	117
0x21 (0x21)	WDTCR	-	-	-	WDFE	WDE	WDF2	WDF1	WDF0	41
0x20 ¹⁾ (0x20 ¹⁾)	UBRRH	URSEL	-	-	-	-	UBRR[15]			105
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCS21	UCS20	UCPOL	103
0x1F (0x1F)	EEARH	-	-	-	-	-	-	-	EEAR8	18
0x1E (0x1E)	EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	18
0x1D (0x1D)	EEDR	EEPROM Data Register								18
0x1C (0x1C)	EEDCR	-	-	-	-	EERIE	EEMWE	EEWE	EERE	18
0x1B (0x1B)	Reserved									
0x1A (0x1A)	Reserved									
0x19 (0x19)	Reserved									
0x18 (0x18)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	63
0x17 (0x17)	DDRB	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	63
0x16 (0x16)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	63
0x15 (0x15)	PORTC	-	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	63
0x14 (0x14)	DDRC	-	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	63
0x13 (0x13)	PINC	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	63
0x12 (0x12)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	63
0x11 (0x11)	DDRD	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0	63
0x10 (0x10)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	63
0x0F (0x0F)	SPDR	SPI Data Register								128
0x0E (0x0E)	SPGR	SPHF	WCOL	-	-	-	-	-	SP2X	128
0x0D (0x0D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	128
0x0C (0x0C)	UDR	USART I/O Data Register								150
0x0B (0x0B)	UCSRA	RXC	TXC	UDRE	FE	DOR	FE	UDX	MPCM	151
0x0A (0x0A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCS22	RXB8	TXB8	152
0x09 (0x09)	UBRRL	USART Baud Rate Register Low byte								155
0x08 (0x08)	ACSR	ACD	ACBG	ACD	ACI	ACIE	ACIC	ACIS1	ACIS0	161
0x07 (0x07)	ADMUX	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0	202
0x06 (0x06)	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	204
0x05 (0x05)	ADCH	ADC Data Register High byte								206
0x04 (0x04)	ADCL	ADC Data Register Low byte								206
0x03 (0x03)	TWDR	Two-wire Serial Interface Data Register								170
0x02 (0x02)	TWAR	TWAR6	TWAS5	TWAA4	TWAS3	TWAS2	TWAA1	TWAS0	TWGC0E	170

