Assembler

Monday, 5 March 2018 13:52

Eine kleine Befehlssammlung:

Q: http://www.microchip.com/webdoc/avrassembler/avrassembler.wb_instruction_list.html

Syntax	Übersetzung	Funktion	Bsp
cp <wert1>, <wert2>	Compare	Setzt Zero-Flag	Cp ACC1, ACC2
breq <sprungziel>	Branch if equal	Prüft, ob Zero-Flag gesetzt ist	breq start
ret	Return		ret
reti	Return Interrupt		reti
sei	Set Global Interrupt Flag	Aktiviert Interrupts generell	sei
ldi <register>, <wert>	Load Immediate	Speichert einen Wert in ein Register	
sts	Store Direct to data space		
call <sprungziel>	Call	Ruft eine Sprungmarke auf und schreibt den jetzigen Programmcnt. in den Stack	call readPin
jmp <sprungziel>	Jump	Wechselt zu einer Sprungmarke	jmp readPin
rjmp <sprungziel>	Relative Jump	Jump mit relativen "Koordinaten" (--> Compiler)	rjmp readPin
move <reg1>, <reg2>	Move	Kopiert den Wert eines "normalen" Registers in ein anderes	
out <reg1>, <reg2>	Out	Schreibt von "normalen" in Special-Function-Register	out DDRD, ACC1
in <reg1>, <reg2>	In	Liest von Special-Function- in "normalen" Register	in ACC2, PINA
lsl <reg> / lsr	Logical Shift Left / Right	Schiebt alle Bit um eine Stelle nach links / rechts	lsl ACC1

Direktiven:

Q: http://www.avr-asm-tutorial.net/avr_de/beginner/diraus.html

Syntax	Übersetzung	Funktion	Bsp
.org		Legt Startadresse zum Schreiben im Flash fest	.ORG 0x40
.cseg		Alles nach .cseg wird als Assembler-Programm erkannt und compiliert	
.db	Define Byte	Definiert konstante Bytes im Flash	LED_Table: .DB {0x5E, 0x67}
.def	define		.DEF ACC = R16

Begriffsdefinition:

Flash-Speicher / EEPROM:

Hier wird das komplette Programm abgespeichert.
Von hier aus kann es "Zeile für Zeile" ausgeführt werden.
Wichtig: Der Flash kann, nachdem das Programm geflasht wurde, nicht mehr beschrieben / geändert werden

Programmcounter

Dieses Register speichert die derzeitige Position ("Zeile") des Programms im Flash-Speicher während der Laufzeit

RAM

Arbeitsspeicher

Register (Akkumulatoren)

Diese Speicher werden oft aus Hardware-Beschränkungen als Zwischenspeicher zum Verschieben von Werten genutzt
Bsp.: Das Register R16 ist mit den meisten wichtigen Komponenten im AVR verbunden

Zero-Flag

Wenn der Compare Befehl "cp" zwei Werte vergleicht, setzt er bei gleichen Werten
(wert1 - wert2 = 0) ein gewisses Bit, das Zero-Flag, auf "1", sonst auf "0".
Von dort aus können Befehle wie "breq" das Ergbnis des Vergleiches abfragen.

Special-Function-Register

"Besondere" Register mit einer fixen Funktion. Sie sind nicht wie normale Register universell verwendbar, sondern werden für Zwecke wie I/O control, Timer, den Stackpointer, Programmcounter, return Adressen etc. genutzt

High- & Lowbytes

Mikrocontroller haben früher auf einer 8-Bit Speicherarchitektur basiert.
Um mehr speichern zu können, verwenden µC mittlerweile 16-Bit.
Dieses System wurde, bildlich gesprochen, einfach auf die alten Architekturen "drübergelegt".

Daher findet man heute immer noch eine Vielzahl an 8-Bit Registern (zB R16, R17), jedoch beinhalten die Register zur Adressierung des Flashs und des RAMs aus nun 16-Bit.

Bsp.: Um die Startadresse für den Stackpointer festzulegen, muss man beide Hälften der 16-Bit Adresse (also jeweils 8 Bit) separat setzen. Die Register hierfür heißen in diesem Fall SPL und SPH, zusammen ergeben sie das 16-Bit Register SP (Siehe "Die Sache mit dem Stackpointer")

Compiler

Den Compiler kann man als Baumeister sehen.
Seine Aufgabe ist es, den Code eines Programms in Bytecode umzuwandeln.

Es ist möglich, dem Compiler eigene Befehle, die sogenannten Präprozesse / Direktiven mitzugeben.
Bsp.: .def ACC = R16 (im Assembler, #define ACC R16 in C)
Diese Präprozesse unterscheiden sich je nach Compiler im Syntax und Funktion.

Wie läuft Call ab?

Grundsätzlich ist call stark mit jmp/rjmp verwandt.
Auch hier wird zu einer Sprungmarke gewechselt, mit dem Unterschied, dass bei call (ähnlich wie bei einer Methode in Java) am Ende (sobald ret ausgeführt wird) wieder zurück zur ursprünglichen Routine gewechselt wird.

Der Ablauf lautet wie folgt:

1. call wird aufgerufen
2. Adresse vom darauffolgenden Befehl wird auf den Stack gelegt
3. Es wird zum angegebenen Label (Sprungmarke) gesprungen
4. Coderoutine wird abgearbeitet
5. Return wird ausgeführt => Programmcounter wird auf die gespeicherte Adresse gesetzt.

Achtung: Da die Rücksprungadresse temporär auf den Stack gelegt wird, ist es wichtig, alle Werte, die innerhalb der Routine auf den Stack gelegt werden, auch wieder zu entfernen.
Ansonsten findet ret die Rücksprungadresse nicht

Unterschied ret & reti:

Interrupt wird mit reti wieder aktiviert, nachdem er, um einen Stackoverflow zu vermeiden, beim Auslösen des Interrupts deaktiviert wurde.
Bei ret müsste man diese Funktion händisch einprogrammieren.

Shiften (Bitschieben)

Beim sogenannten Shiften verschiebt man ein Bit um eine gewisse Anzahl an Stellen nach Links / Rechts.

Left [<<]

Bsp.: Shiften eines 1ers um 7 Stellen:

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

1<<7

7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0

Right [>>]

Bsp.: Shiften eines 1ers um 7 Stellen:

7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0

-128>>7

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

Gibt man als zu verschiebendes Bit zB ein Register an, werden alle Bits in diesem Register um die angegebene Anzahl von Stellen verschoben
Bsp.: Syntax: R16<<4

io-Komponenten.pdf

Wer is "Stack"?!

Den Stack kann man sich vorstellen wie einen Stapel Bücher.
Man kann immer nur ein Buch von oben nehmen, keines von unten und keines aus der Mitte.
Man kann ein Buch nur lesen, wenn man es bereits auf den Stapel gelegt hat.
Wichtig ist, jederzeit zu wissen, was man wann bzw in welcher Reihenfolge auf den Stapel gelegt hat.

Etwas fachspezifischer ausgedrückt ist der Stapel ein Speichersystem, mit dem man mit dem RAM interagieren kann.
Im Gegensatz zu Registern, mit denen man zB Daten vom Controller zur Peripherie verschickt, wird er als schneller Zwischenspe icher innerhalb von Routinen (bzw Operationen) genutzt.
Der Stapelzeiger rutscht beim lesen immer eine Stelle im Speicher "hinunter", beim schreiben eine Stelle "hinauf". (Das bedeu tet, dass der Stapel, wenn man nicht aufpasst, den gesamten RAM anfüllen kann. => Stackoverflow)

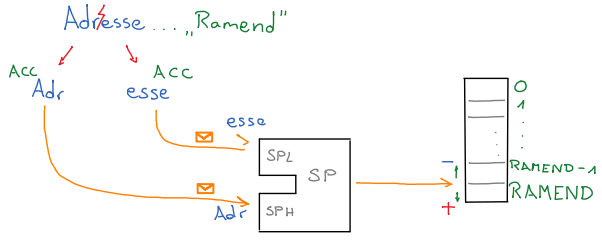
Die Sache mit dem Stackpointer...

Hier wird der Anfang des Speicherbereichs des Stapels gesetzt
(= "Startpunkt" des Stapelpointers (Adresse))

Hier ist wichtig zu wissen, dass RAMEND eine fixe Compiler-Konstante mit einer Größe von 2 Byte ist, das Register ACC aber nur ein Byte speichern kann (8 FlipFlops)

ldi ACC1, LOW(RAMEND)	Unteres Byte der Adresse des Endes des RAM-Speichers wird in ACC1 gespeichert
out SPL, ACC1	Das "untere" Byte des Stackpointers wird mit der Adresse aus dem Register konfiguriert
ldi ACC1, HIGH(RAMEND)	Oberes Byte der Adresse des Endes des RAM-Speichers wird in ACC1 gespeichert
out SPH, ACC1	Das "obere" Byte des Stackpointers wird mit der Adresse aus dem Register konfiguriert

SPH und SPL ergeben somit zusammengesetzt die Startadresse SP für den Stackpointer



Pins und Ports

Q: http://www.ruhr-uni-bochum.de/nds/lehre/vorlesungen/eingebetteteprozessoren/ss05/io-Komponenten.pdf

Ein AVR hat 4 Ports A, B, C und D, die jeweils aus 8 Pins bestehen.
Außerdem haben sie jeweils ein Data Direction Register (DDRA, DDRB, ...)

Wichtig: Man kann nur PINweise einlesen und nur PORTweise ausgeben