Assemblieren

Um die Funktionsweise eines Assemblers besser verstehen zu können, wollen wir ein kleines Assemblerprogramm einmal von Hand assemblieren. Als Beispiel wählen wir unser Programm hf_sender.asm aus dem letzten Kapitel:

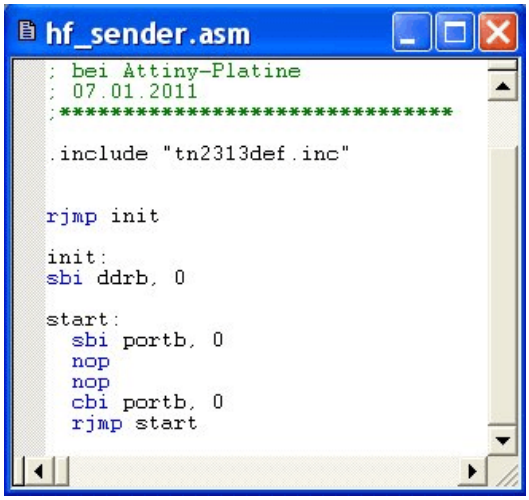

Abb. 1

Zunächst verschaffen wir uns Informationen über den Maschinencode der auftauchenden Mnemonics; dazu greifen wir auf die Hilfe-Datei von AVR-Studio zurück: Wenn wir z. B. etwas über den Befehl `cbi` in Erfahrung bringen wollen, setzen wir im Quelltext den Cursor auf diesen Befehl und betätigen die Taste F1. Es öffnet sich das folgende Hilfe-Fenster:

Abb. 2

Unter der Zwischenüberschrift 16-Bit-Opcode finden wir die gewünschte Information für den Maschinecode zu `cbi Adr, bit`:

1001	1000	AAAA	Abbb

Dabei stehen die Buchstaben A für die einzelnen Bits der Adresse `Adr` des I/O-Registers und die Buchstaben b für die einzelnen Bits der Bitnummer `bit` des Ports. In unserem Fall steht `Adr` für die Adresse des I/O-Registers PortB; diese ist &H18 = &B11000 (vgl. Manual S. 211). Die Bitnummer ist &B000. Damit lautet unser Befehl `cbi portb, 0` in binärer und hexadezimaler Schreibweise:

`cbi portb, 0`	&B 1001 1000 1100 0000
	&H 98 C0

Auf ähnliche Weise findet man:

`sbi portb, 0`	&B 1001 1010 1100 0000
	&H 9A C0

`sbi ddrb, 0`	&B 1001 1010 1011 1000
	&H 9A B8

`nop`	&B 0000 0000 0000 0000
	&H 00 00

Die Assemblierung des Sprungbefehls `rjmp` ist etwas komplizierter. Der Blick in die Hilfe-Datei zeigt zunächst, dass der Maschinenbefehl durch den 16-Bit-Code

1100	kkkk	kkkk	kkkk

gegeben ist. Die k-Bits beschreiben den relativen Sprung zu dem neuen Befehl. Wie ist das zu verstehen? Üblicherweise wird der Programmzähler (PC) nach jedem Befehl um 1 erhöht; dadurch wird gewährleistet, dass die einzelnen Befehle in der Reihenfolge abgearbeitet werden, wie sie im Speicher (Flash) des Mikrocontrollers stehen:

PC ← PC + 1

Bei einem Sprung wird diese Standardreihenfolge unterbrochen. Bei einem relativen Sprung um den Wert k gilt für den Programmzähler PC:

PC ← PC + k + 1

k gibt also an, um wie viele Einheiten der Programmzähler zusätzlich erhöht wird. Durch k wird also nicht die (absolute) Zieladresse angegeben; vielmehr gibt k an, um wie viele Einheiten der PC relativ zur aktuellen Adresse zusätzlich zur standardmäßigen Erhöhung um 1 verändert werden soll. In der Abb. 3 ist dies für den Wert k = 3 dargestellt. In diesem Beispiel werden die nächsten 3 Befehle übersprungen.

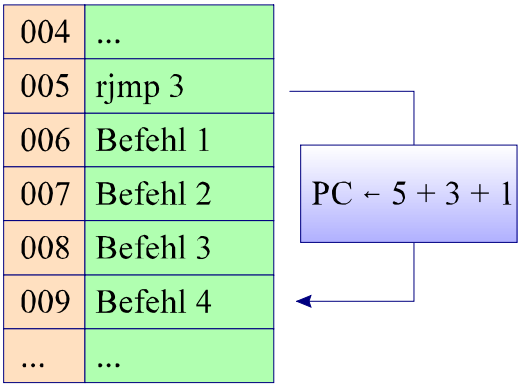

Abb. 3

Ist der Wert k = 0, dann wird einfach der nächste Befehl ausgeführt. Das ist genau bei dem ersten Sprungbefehl in unserem Programm hf_sender.asm der Fall. Der Sprung zur Marke init: führt ja nur zum nächsten Befehl im Programmspeicher. Der Assembler ermittelt somit für k den Wert 0 und setzt ihn in den Maschinencode ein; er lautet damit:

`rjmp 0`	&B 1100 0000 0000 0000
	&H C0 00

Bei dem zweiten Sprung ist der Wert von k negativ: Vom Befehl `rjmp start` zum Befehl `sbi portb, 0` muss der PC (nach seiner Erhöhung um 1) um 5 verkleinert werden; es gilt also k = -5.

Jetzt müssen wir nur noch wissen, wie hier negative Zahlen dargestellt werden. Die Vorgehensweise erinnert an einen Kilometerzähler beim Auto. Fährt man vom Kilometerstand 00003 aus mehrere Kilometer rückwärts, so zeigt er die folgenden Werte an:

00003	3
00002	2
00001	1
00000	0
99999	-1
99998	-2
99997	-3
99996	-4

Die Angaben, welche das Zählwerk anzeigt, wenn 00000 unterschritten wird, können wir als negative Zahlen deuten. Bei 3-stelligen Hexadezimalzahlen sieht das dann so aus:

&H 002	2
&H 001	1
&H 000	0
&H FFF	-1
&H FFE	-2
&H FFD	-3
&H FFC	-4
&H FFB	-5

Damit lautet der Maschinencode für unseren letzten Sprungbefehl:

`rjmp -5`	&B 1100 1111 1111 1011
	&H CF FB

Das gesamte Maschinenprogramm ist somit:

&H 0000	&H C0 00
&H 0001	&H 9A B8
&H 0002	&H 9A C0
&H 0003	&H 00 00
&H 0004	&H 00 00
&H 0005	&H 98 C0
&H 0006	&H CF FB

Wir vergleichen diesen Code mit dem Intel-Hex-Code, welchen wir uns mit dem Uploader-Programm anschauen können:

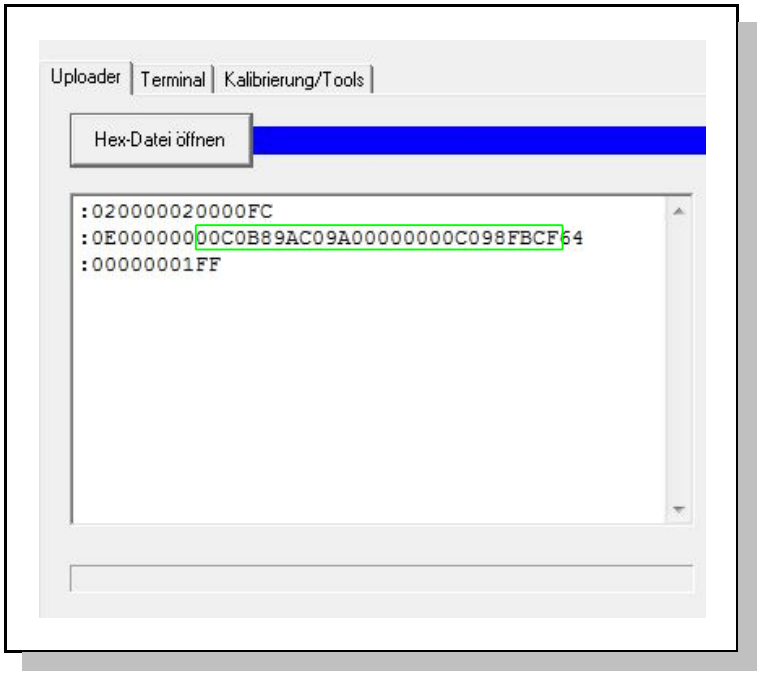

Abbildung 4

In der zweiten Zeile finden wir unser Maschinenprogramm; allerdings sind hier jeweils das High-Byte und das Low-Byte eines Befehlswortes vertauscht. Im INTEL-HEX-Code steht ja immer das Lowbyte vor dem Highbyte. Statt C0 00 finden wir hier also 00 C0.

Bis auf diesen Unterschied in der Schreibweise stimmt unser Übersetzungsversuch vollständig mit dem Ergebnis des Assemblers überein. Das zeigt, dass eine solche Übersetzung eigentlich keine Hexerei ist. Allerdings wollen wir in Zukunft diese Arbeit doch dem Assembler-Programm überlassen...

Aufgaben

1. Wie groß muss der Wert von k beim Sprungbefehl `rjmp` sein, damit eine Endlosschleife entsteht? Geben Sie den Hexadezimalcode für eine solche Endlosschleife an.

2. Wie viele verschiedene Adressen sind bei dem `cbi`-Befehl höchstens möglich? Warum sind für die Bitnummer beim `cbi`-Befehl nur 3 Stellen reserviert?

3. Bestimmen Sie den Hexadezimalcode für `sbi portd, 3.`