2.5 Atmega-Peripherie/SRAM

2.5.1 Wozu das SRAM benutzen?

Es sollen die Bytes gesammelt werden, die von der seriellen Schnittstelle hereingekommen sind. Auf Tastendruck sollen sie alle wieder über die serielle Schnittstelle ausgegeben werden.

In welchem Speicherbereich kann man die Bytes sammeln? Die Universalregister scheiden für diesen Zweck aus, denn ihre Anzahl ist begrenzt (nur 32). Das Flash-EPROM scheidet ebenfalls aus, denn es kann in der Regel nur gelesen werden (die Programmierung erfolgt bei Flash-EPROMs in großen Blöcken). Daher muss hier das SRAM angesprochen werden.

2.5.2 Speicherbereich

Das SRAM in ATmega-Controllern ist acht Bit breit; jedes Byte kann einzeln adressiert werden. Der Adressbereich liegt zwischen null und einer Zahl, auf die man durch die Konstante `RAMEND` zugreifen kann. Tabelle 1 zeigt den Inhalt von `RAMEND` bei verschiedenen ATmega-Typen[1]. Man

Typ	`RAMEND`	`SRAM_START`	`SRAM_SIZE`
ATmega8	1119	96	1024
ATmega16	1119	96	1024
ATmega32	2143	96	2048

Tabelle 1: SRAM bei verschiedenen ATmega-Typen

sieht dort auch, dass die SRAM-Adressen höher reichen, als es die SRAM-Größe vermuten lässt. Das liegt daran, dass die unteren (hier: 96) SRAM-Adressen gleichzeitig als Register angesprochen werden können:

- An den Adressen 0 bis 31 liegen die 8-Bit-Universal-Register `r0` bis `r31`.
- An den Adressen 32 bis 95 liegen die 8-Bit-SFR-Register mit den Nummern 0 bis 64.

Die niedrigste für Variablen nutzbare Adresse ist dann die 96.

2.5.3 Festlegen von Variablen im SRAM

Zum Ansprechen des SRAM gibt es die Speicher-Direktiven `dseg` und `byte`. Mit `dseg` legt man fest, dass die folgenden Elemente fortlaufend im SRAM abgelegt werden, beginnend mit der niedrigsten freien Adresse. Mit `byte` kann man nun eine bestimmte Menge an Speicherplatz reservieren, angegeben in Byte. Zum einfachen Zugriff auf den SRAM-Speicherplatz empfiehlt es sich, mit Hilfe einer Marke einen symbolischen Namen für die Adresse festzulegen. Er kann dann als Variablenname für diesen SRAM-Platz dienen:

```
1           .dseg
2 beispiel: .byte 20      ; 20 Bytes ab einer freien Adresse>95
```

Eine Initialisierung wie beim Flash-EPROM ist beim SRAM nicht möglich, da man im SRAM jan nichts dauerhaft speichern kann.

2.5.4 Direkte SRAM-Adressierung mit `sts` und `lds`

Mit dem Befehl `sts zahl, reg` speichert man den Inhalt eines Universalregisters `reg` im SRAM an der Adresse Zahl. Mit dem Befehl `lds reg, zahl` lädt man den Inhalt des SRAM-Bytes an der Adresse `zahl` in ein Universalregister. Man spricht von *direkter Adressierung*. Tabelle 2 zeigt den Unterschied zwischen den Befehlen `ldi` und `lds`. Meistens benutzt man für die Adresse (oben `zahl` genannt) einen symbolischen Namen. Ein kleines Programmbeispiel sieht man hier:

[1] Wenn man AVRA installiert, kann man die Werte in den Dateien `/usr/share/avra/m*def.inc` nachsehen.

Befehl	Aktion	Adressierungsart	Schreibw. PC-ASM
`ldi r16,100`	Zahl 100 nach r16	unmittelbar	`mov r16,100`
`lds r16,100`	Inh.v.SRAM-Byte Nr.100 nach r16	direkt	`mov r16,[100]`

Tabelle 2: Vergleich `ldi` und `lds`

```
1          sts     meins, r16      ; speichert r16 nach meins
2          inc     r16
3          lds     r16, meins      ; laedt r16 aus meins
4          ....
5 ;=============================================================
6          .dseg                   ; hier beginnt das SRAM
7 meins:   .byte   1               ; Adresse 96, 1 Byte
```

2.5.5 Indirekte SRAM-Adressierung mit `sts` und `lds`

Die Befehle `st`, `ld` benutzen den Inhalt von einem der 16-Bit-Register `x`, `y` oder `z` als Adresse:

```
1          st      z, r16      ; entspricht *z = r16
2          ld      r16, z      ; entspricht r16 = *z
```

Mit dem oberen Befehl wird der Inhalt von r16 in die Speicherzelle, deren Inhalt in `z` steht, kopiert. Bei dem unteren Befehl ist es umgekehrt. Man spricht von *indirekter* Register-Adressierung. Tabelle 3 zeigt den Unterschied zwischen den Befehlen `mov` und `ld`. Die Befehle `st` und `ld` bieten

Befehl	Aktion	Adressierungsart	Schreibw. PC-ASM
`mov r16,r9`	r9 nach r16	Register-direkt	`mov r16,r9`
`ld r16,z`	Inh.v.SRAM-B.mit Adr.in z nach r16	Register-indirekt	`mov r16,[z]`

Tabelle 3: Vergleich `mov` und `ld`

zudem noch die Möglichkeit des Auto-Inkrement und Auto-Dekrement:

```
1          st      z+, r16 ; entspricht st z, r16; inc z
2                          ; oder *(z++)=r16; /* postfix */
3          st      z-, r16 ; entspricht dec z; st z, r16
4                          ; oder *(--z)=r16; /* prefix */
5                          ; ebenso ld mit z+ und z-
```

Damit kann man in einer Schleife eine ganze Tabelle ohne viel Aufwand lesen oder speichern.