Use of the LPM (Load Program Memory)

- **Use of the LPM (Load Program Memory) Instruction with the AVR Assembler**
- **Load Constants from Program Memory**
- **Use of Lookup Tables**

The LPM instruction is included in the **AVR Instruction Set** to load a data byte from the FLASH Program memory into the Register File.

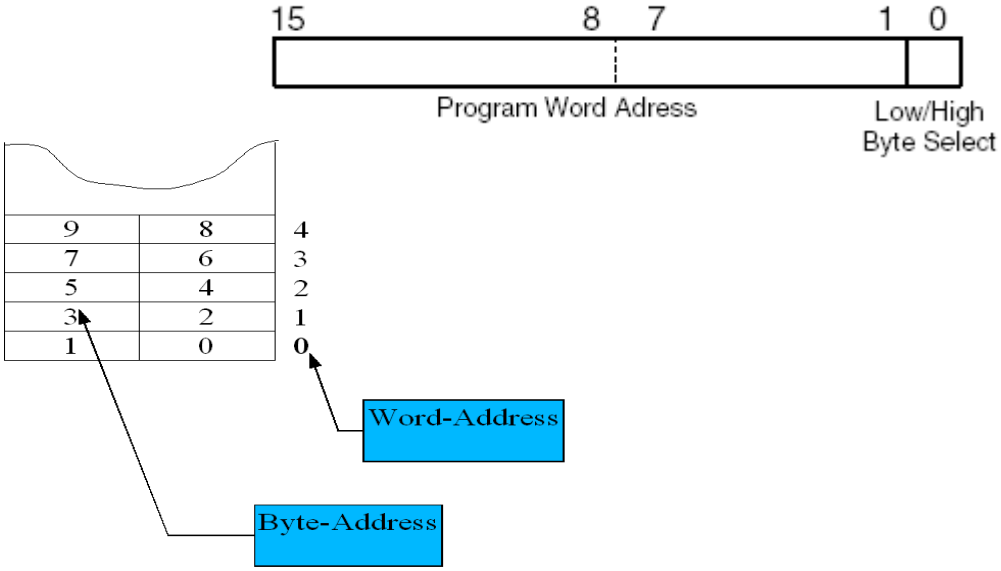
The **Flash** Program memory of the AVR microcontroller is organized as **16 bits** words.

The Register File and **SRAM** Data memory are organized as **eight bits** bytes.

Special consideration must therefore be taken when loading data from Program memory to Data meory.

The **Z-register** in the Register File is used to access the Program memory. This 16 bits register pair is used as a **16 bits pointer** to the Program memory.

Figure 1. Z Address Register




```
Byte-Address = Word-Address*2
```

The 15 most significant bits selects the word address in Program memory. Because of this, the word address is multiplied by two before it is put in the Z-register (Z-register needs the Byte-Address of low Byte)

The least significant bit of the Z Address Register selects either Low byte (0) or High

byte(1) of the Program memory word.

To load data from random places in program memory, the Z-register must be set up with the proper address each time a new address is accessed.

In Program memory the data is organized with one byte in the low part of a program word and the next byte in the high part. Because of this, the message string will appear as if every pair of characters has been swapped, when viewed in the memory view in AVR Studio

\x0	d	1F
l	r	1E
o	W	1D
\x20	o	1C
l	l	1B
e	H	1A

```
;**** Ex_LPM1 ****
;**** A P P L I C A T I O N N O T E A V R 1 0 8 ************************
;* DESCRIPTION
;* This Application note shows how to use the Load Program Memory (LPM)
;* instruction. The App. note loads the string "Hello World" from
;* program memory byte by byte, and puts it onto port B.
;*
;*******************************************************************************

.include "8515def.inc"     für PORTB, DDRB benötigt
.device AT90S8515   ; Specify device    der Assembler kann dann Meldungen liefern, ob ein bestimmter Befehl für dieses Device überhaupt vorhanden ist
.def temp=r16                    ; Define temporary variable
start:
      ldi temp,low(RAMEND)   der Stack wird am Ende des RAMS eingerichtet; er wächst nach unten; SPL = Stack pointer LOW Byte SPH = Stack pointer HIGHT Byte
      out SPL,temp                     ; stack pointer to last int. RAM location
      ldi temp,high(RAMEND)
      out SPH,temp
      ldi temp,$ff               der Befehl „out" nimmt nur ein Register als Argument! Daher zuerst den Wert in Register laden
      out DDRB,temp              ; Set port B as output
      out PORTB,temp             ; Set all pins at port B high

; Load the address of 'message' into the Z register. Multiplies

; word address with 2 to achieve the byte address, and uses the
; functions high() and low() to calculate high and low address byte.
```

Der Befehl lpm holt von der Adresse z (16bit pointer) ein Byte in das Register r0

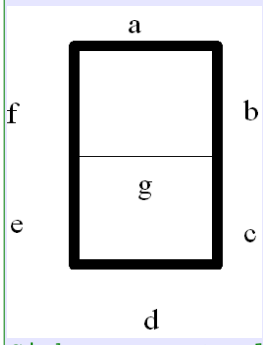
message ist ein label (Adresse); der Programmspeicher ist in Worten (2Byte) organisiert d.h. Das Label hat z.B. den Wert 5 (Wortadresse 5), die Adresse im Speicher, also die Adresse wo der Pointer z hinzeigt ist aber 10

```
      ldi ZH,high(2*message)     ; Load high part of byte address
      ldi ZL,low(2*message)      ; Load low part of byte address into ZL
loadbyte:
      lpm                        ; Load byte from program memory into r0
      tst r0                     ; end of the message reached?
      breq quit     branch if equal          ; If so, quit
      out PORTB,r0                     ; Put the character onto Port B
      rcall one_sec_delay        ; A short delay
      adiw ZL,1     16 bit Pointer increment (add immediate word); Increase Z
      rjmp loadbyte
   quit: rjmp quit

one_sec_delay:
      ldi r20, 20
      ldi r21, 255
      ldi r22, 255
   delay:
      dec r22
      brne delay
      dec r21
      brne delay
      dec r20
```

```
        brne delay
        ret
message:
        .db “Hello World”   db = define byte; legt die Zeichen hintereinander im Programmspeicher ab!
        .db 0   abschließende Null (vgl. Strings in c)
```

```
;**** Ex_LPM2 ****
;binär --> siebensegment Coder
```




```
In diesem Beispiel wird für eine Ziffer r1 der
Siebensegmentcode ermittelt. Dazu wird in einer Lookuptable
nachgeschlagen, die mit db-Direktiven als Konstante in den
Programmspeicher gebracht wurden
Siebensegmentcode: die Elemente a..g werden in aufeineanderfolgender
Reihe angegeben:
z.B: '0'    0111 1110 = 0x7E
          '1'    0011 0000 = 0x30
das 8. Element ist der Punkt;




.include "8515def.inc"

.CSEG

      ldi r31,HIGH(lookuptable*2)      ; Z high byte
      ldi r30,LOW(lookuptable*2)       ; Set Z low byte to
      lpm                                          ; load from flash (0x33)

      adiw ZH:ZL,1        ; z++ 16bit increment of Pointer z
add immediate word

      lpm                        ; load into r0 from z Pointer (0x77)

;     lpm r0,z+                  ; does not work at at90s8515
microprocessor

      lpm                        ; load into r0 0xff (255)

      ldi r30, 1           ; ldi needs register r16-r31
      mov r1, r30
      rcall bin_to_7seg
      ldi r30, 0xFF              ; write back the result
      out DDRB, r30
      out PORTB, r0

end:rjmp end

bin_to_7seg:
      ; get the 7segment code
      ; parameter:  r1 ... bin value
      ; return value: r0 .. 7seg code
      ; changed registers: z
```

```
      ldi r31,HIGH(code7seg<<1) shift left 1 position = Multipliziere* 2 ; Z high
byte
      ldi r30,LOW(code7seg<<1)   ; Set Z low byte to
```

z zeigt jetzt auf den Anfang der Lookuptable; er muß jetzt um r1 Stellen weitergestellt werden z.B: r1 =2 ==> z zeigt auf den Siebensegmentcode von „2“

```
      clr r2                                    ; set the pointer to z+r1
      add r30, r1
      adc r31, r2

      lpm   ;7seg Cod
      ret

      .ORG 0x122    ; organize the following code starting at location 0x122
lookuptable:
      .DW 0x7733,255, 0b01010101, -128, 0xaa ,1,2,3,4,0
code7seg:

      .DW 0xa0,0xa1,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9 diese Werte dienen
in der Entwicklungsphase als Ersatzwerte für die eigentlichen 7-Segment-Codes
```