AVR108: Setup and Use of the LPM Instruction

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

Introduction
This application note describes how to access constants saved in Flash Program memory of the AVR microcontrollers. The AVR is based on a Harvard architecture, this means that Address and Data memory use separate busses. This is necessary to achieve single cycle instructions execution speed. To be able to save constants in Flash memory the Load Program Memory (LPM) instruction is included in the instruction set.

Use of the LPM
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 memory.

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. 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.

Figure 1. Z Address Register

The least significant bit of the Z Address Register selects either Low byte (0) or High byte(1) of the Program memory word. To calculate the Low (ZL) and High (ZH) part of the address, use the LOW() and HIGH() functions.

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®.
The program in this application note loads a string of bytes from the Program memory, and writes it to Port B. It first initializes Port B so that all the pins are output. It loads the starting address of the string “Hello World” into Z-register, as described above. Then a byte is loaded from program memory using LPM. The program checks whether or not the end of the string is reached (byte was zero). If the end is not reached yet the last read byte is put on Port B, a short delay is made, and the Z-register is increased. The program then jumps back to load another byte.

```
.include "8515def.inc"
.device  AT90S8515 ; Specify device
def temp=r16 ; Define temporary variable

start:
ldi temp,low(RAMEND) ; Set stack pointer to last internal RAM location
out SPL,temp ; Set stack pointer to last internal RAM location
ldi temp,high(RAMEND)
out SPH,temp
ldi temp,$ff
out PORTB,temp ; Set all pins at port B high
out DDRB,temp ; Set port B as output

; 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.
ldi ZH,high(2*message) ; Load high part of byte address into ZH
ldi ZL,low(2*message) ; Load low part of byte address into ZL

loadbyte:
  lpm ; Load byte from program memory into r0
```
tst    r0    ; Check if we've reached the end of the message
breq   quit    ; If so, quit

out    PORTB,r0    ; Put the character onto Port B
rcall   one_sec_delay    ; A short delay

adiw    ZL,1    ; Increase Z registers
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   0