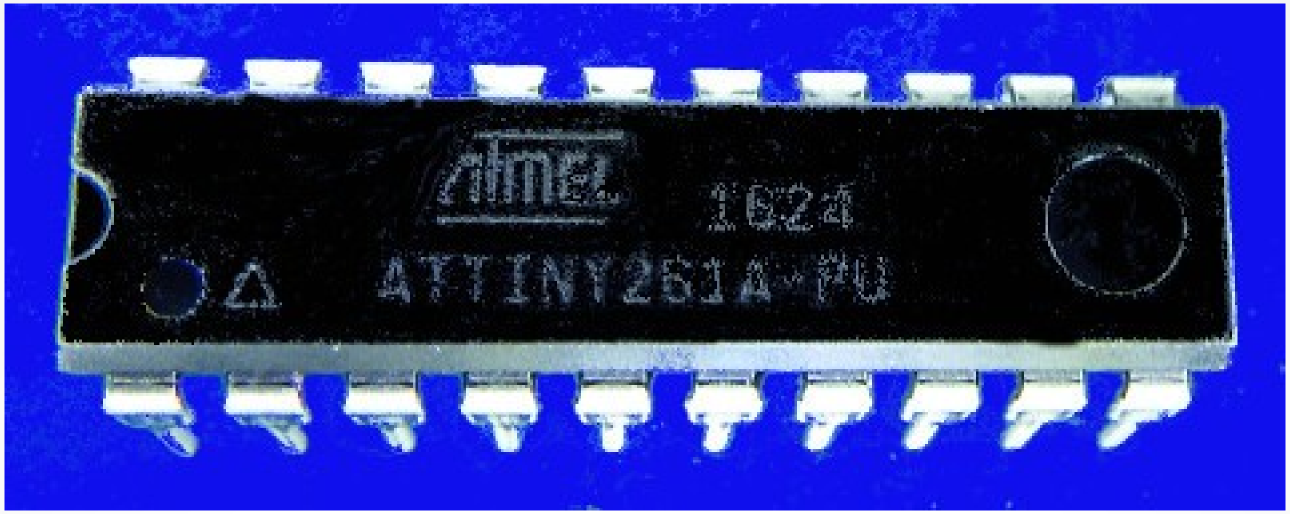


Lecture 11: Tables and accesses

Hardware, Internals and Programming of AVR Microcontrollers in Assembler

by
Gerhard Schmidt
Kastanienallee 20
D-64289 Darmstadt

Table access

- Tables can be used to store fixed values.
- Those can be located in the flash, in the SRAM or in the EEPROM memory.
- To store a table in the flash memory, use the following source code:

```
; A table with bytes and one with words
MyByteTable: ; The table needs a label to be accessible
.DB 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ; A byte-wise table, bytes separated with comma
MyWordTable: ; A label for the word table, words separated by comma
.DW 1000, 2000, 3000, 4, 5, 6
```

- As the flash memory is organized as 16-bit words, the bytes in MyByteTable: are byte-wise packed as words. The first byte goes to the LSB, the second to the MSB, therefore the first word is 0x0201.
- The MyWordTable: is already organized in 16-bit word format, so each MSB:LSB word creates one flash memory location. In case that the number has MSB=0 (4, 5 and 6 in the example), the MSB is nevertheless stored (0x0004, 0x0005 and 0x0006).

Table access

- When reading from this table,
  - two read operations have to take place, and
  - the least significant bit of the read address decides whether the LSB or the MSB at this address shall be accessed, and
  - reading requires the double register ZH:ZL, which is R31:R30, for the address, and
  - this address has to be shifted one bit position left to gain space for the MSB/LSB access bit.
- Reading one byte from the table MyByteTable: in the flash memory:

```
; Reading the fifth byte from MyByteTable
	LDI ZH, High(2*MyByteTable + 4) ; Point to the fifth byte entry, MSB
	LDI ZL, Low(2*MyByteTable + 4) ; dto., LSB
	LPM R16, Z ; Read byte at that address to R16
```

Table access auto-inc/dec

- If LPM does not specify the register and Z (just pure LPM) it reads the byte at Z to R0.
- As accessing memory with LPM is very often sequentially:
    - LPM R16, Z+ auto-increments Z (adds 1 to Z wordwise) after reading,
    - LPM R16, -Z first subtracts one from Z and reads the byte there-after.
- Access to calculated table addresses can be done with adding the displacement first:

```
; Reading the fifth byte from MyByteTable
        LDI ZH, High(2*MyByteTable + 4) ; Point to the fifth byte entry, MSB
        LDI ZL, Low(2*MyByteTable + 4) ; dto., LSB
        ADD ZL, R16 ; Displacement in R16
        LDI R16, 0 ; MSB displacement (LDI does not affect flags, CLR would also clear C)
        ADC ZH, R16 ; Add carry flag to MSB
        LPM R16, Z ; Read byte at that address displaced by R16
```

Table access word-wise

- If the table is organized wordwise: LPM has to be done twice to read LSB and MSB.

- As an example: with a table that holds addresses, access to the third address can be done as follows:

```
; Reading the third address from MyWordTable
	LDI R16, 3 ; The displacement of the address in the address table to be copied
	LDI ZH, High(2*MyWordTable) ; Point to the beginning of the table, MSB
	LDI ZL, Low(2*MyWordTable) ; dto., LSB
	LSL R16 ; Double the 3 in R16 because each address is 16 bits long
	ADD ZL, R16 ; Adding the displacement in R16 to the base address, LSB
	LDI R16, 0 ; MSB of the displacement
	ADC ZH, R16 ; Add carry flag to MSB
	LPM XL, Z+ ; Read byte at that address displaced to double register X, LSB
	LPM XH, Z ; Read the MSB
```

- Another example: To convert a 16-bit binary to decimal one needs the decimals of 10,000, 1,000, 100 and 10 in binary form:

```
; My decimal table
DecTab:
.DW 10000, 1000, 100, 10, 0 ; The decimal table as binary words
```

Decimal conversion

- **With that table the decimal conversion of a 16-bit binary in R1:R0 to a decimal in the SRAM goes as follows:**

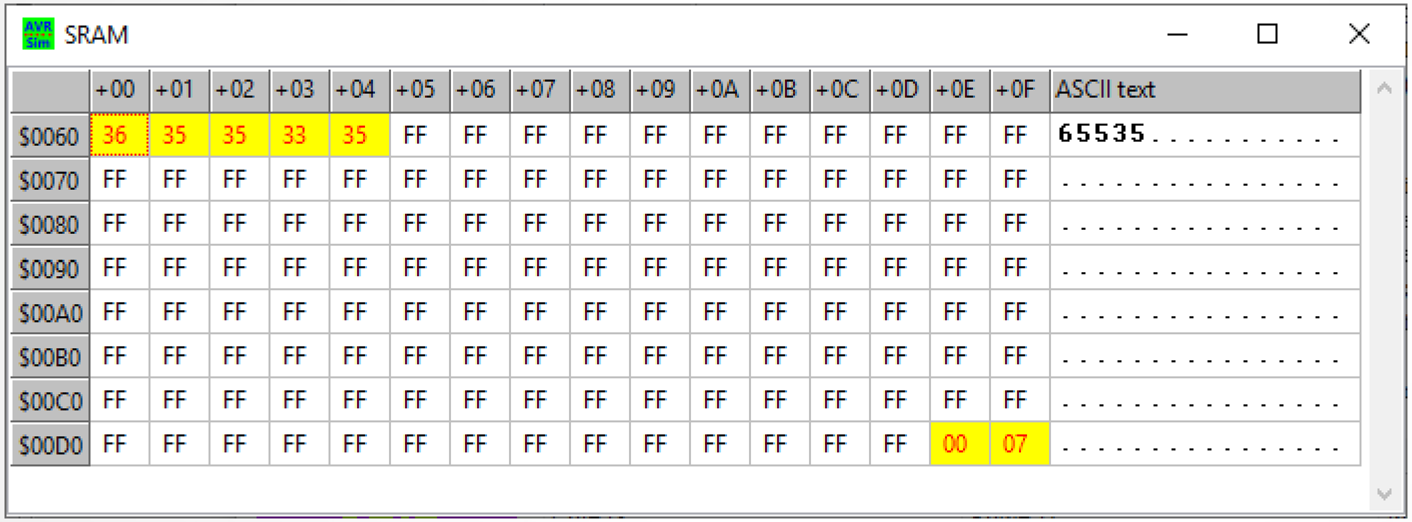
```
; Converting the binary in R1:R0 to decimal in SRAM
DecConv:
	LDI XH, High(SRAM_START) ; Point X to SRAM start, MSB
	LDI XL, Low(SRAM_START) ; dto., LSB
	LDI ZH, High(2*DecTab) ; Point Z to decimal tab, MSB
	LDI ZL, Low(2*DecTab) ; dto., LSB
DecConvDec:
	LPM R2, Z+ ; Read decimal value to R2, LSB, and auto-increment Z
	LPM R3, Z+ ; dto., Read MSB to R3
	TST R2 ; Is the LSB zero?
	BREQ DecConvFinished ; Yes, decimal conversion is complete
	SER R16 ; Count how often the decimal can be subtracted, start with -1
DecConvSubtr:
	INC R16 ; Count number of subtractions up
	SUB R0, R2 ; Subtract the decimal binary from the LSB
	SBC R1, R3 ; Subtract the MSB and the carry flag
	BRCC DecConvSubtr ; No carry has occurred, continue subtraction
	ADD R0, R2 ; Undo the last subtraction, LSB
	ADC R1,R3 ; dto., MSB
	SUBI R16, -‘0‘ ; Add an ASCII Null (decimal 48)
	ST X+, R16 ; Copy result byte to SRAM and auto-increment
	RJMP DecConvDec ; Continue with the next decimal binary
```

Decimal conversion - continued

- **Now the last digit has to be prepared:**

```
DecConvFinished:
        LDI R16, ‘0‘ ; Load ASCII 0 to R16
        ADD R16, R0 ; Add the last digit
        ST X, R16 ; Write the last digit to SRAM
```

- **This is what the SRAM looks like if 0xFFFF is converted to decimal.**



SRAM

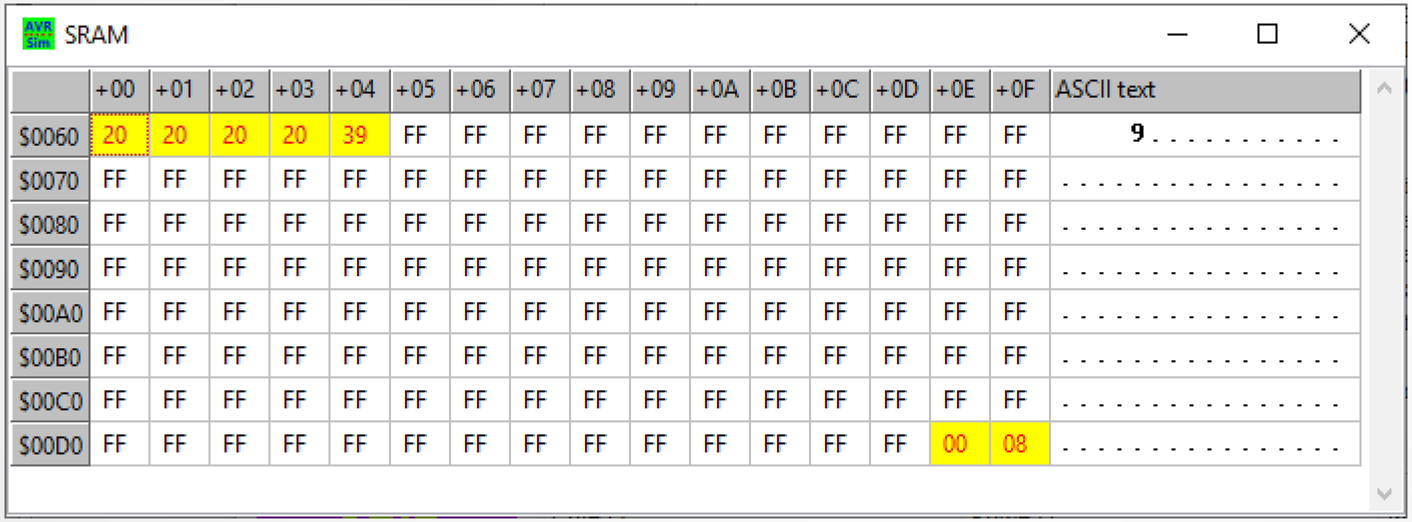
	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	ASCII text
$0060	36	35	35	33	35	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	65535...........
$0070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
$0080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
$0090	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
$00A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
$00B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
$00C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
$00D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	07	....

- **As there are leading zeroes, if the number is smaller than 10,000, those have to converted to blanks.**
- **The following code does this.**

Decimal conversion - continued

```
LeadingZeros:
	LDI XH, High(SRAM_START) ; Point X to the ASCII number, MSB
	LDI XL, Low(SRAM_START) ; Dto., LSB
LeadingZerosChck:
	LD R16, X ; Read character at position
	CPI R16, ‘0‘ ; Is that a leading zero?
	BRNE LeadingZerosFinished ; No, not a zero, finished
	LDI R16, ‘ ‘ ; Load a blank
	ST X+, R16 ; Overwrite the ASCII zero
	CPI XL, SRAM_START+4 ; Is that the last digit?
	BRNE LeadingZerosChck
LeadingZerosFinished:
```

- **This is what comes out if a binary 9 is converted:**


SRAM

	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F	ASCII text
$0060	20	20	20	20	39	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	9 . . . .
$0070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . .
$0080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . .
$0090	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . .
$00A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . .
$00B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . .
$00C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	. . . .
$00D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	08	. . . .

Double registers - specialties

- There are 3 double registers in AVRs: X (R27:R26), Y (R29:R28) Z (R31:R30). With those, ST writes the content of a register to the SRAM location that X/Y/Z point to (pointer registers). LD loads the content in the SRAM to a register.
- X, Y and Z can auto-post-increment and auto-pre-decrement the pointer address.
- With ADIW XL/YL/ZL, constant a constant between 0 and 63 can be added word-wise, with SBIW XL/YL/ZL, constant subtracted.
- The register pair R25:R24 also knows ADIW and SBIW, but cannot point to locations in SRAM.
- The addresses below SRAM-START can be accessed with X/Y/Z too, there are the registers and the port registers located.

Double registers - specialties

- **A special feature can be used with Y and Z: a displacement constant can be added temporarily, the instructions are STD and LDD.**
- **This feature is useful if you have a record-type structure in SRAM, that have identical inner structures. By changing the base address in Y/Z you can access the single bytes of the identical structure inside the record.**
- **Accesses to SRAM locations without pointers are STS address, register and LDS address, register. Those are two-word instructions that have the 16-bit address as second word.**
- **As in larger AVRs the port registers exceed 64 locations due to the many hardware, these extra port registers can be accessed by this method. So, if IN/OUT reports an error (port address above 0x3F), try if STS/LDS with the given address in the def.inc does not fail.**

Conclusions

- **The AVRs offer many different methods to access flash memory, registers, port registers and SRAM.**
- **Choosing the appropriate addressing method can optimize program flow, increase the elegancy of the source code and save execution time.**

Questions and tasks in Lecture 11

Task 11-1: Write a program that reads a null-terminated text from a byte table in the flash memory (inserted with .DB “This is a text.“, 0) to the beginning of the SRAM.

Bonus Task 11-1: Write the text backwards to the SRAM. Try different methods and find out which is the most elegant.

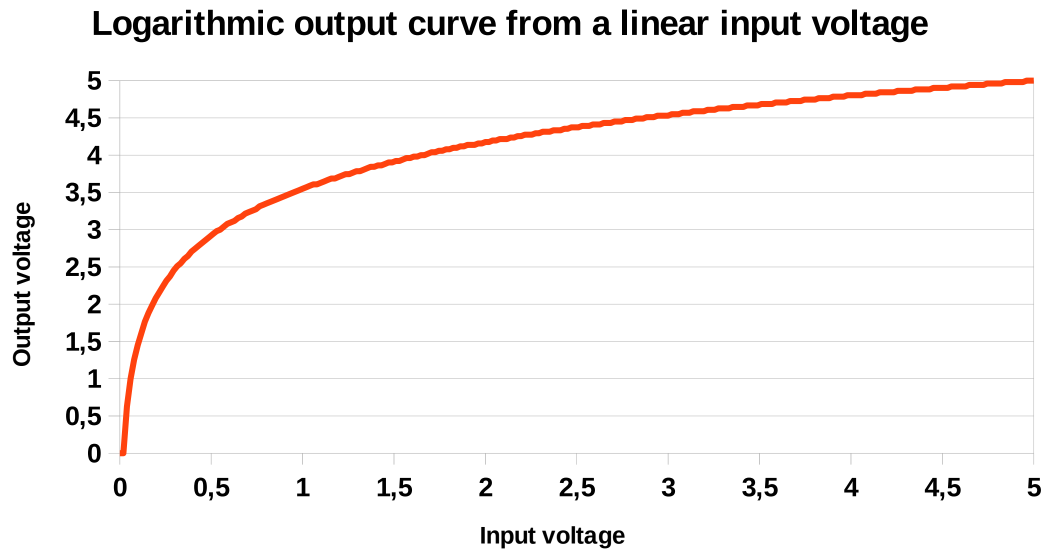
Bonus question: How long can the text be in an ATtiny24, in an ATtiny44 and in an ATtiny84 (how many characters)?

Questions and tasks in Lecture 11 - Continued

Task 11-2: Write a program that continously

➔reads an analog voltage byte-wise (with ADLAR!) every 10 ms (use DATE and a timer to start conversion, use the ADCC interrupt and the T flag),

➔converts this to its logarithm with a byte-wise table, and



➔writes that value to an 8-bit PWM at 100 Hz to convert it back into a voltage.

Questions and tasks in Lecture 11 - Continued

Task 11-3: Write a program that converts 32-bit binaries to decimal in the SRAM and blanks leading zeros.