Introduction to AVR Assembly Language Programming II
February 2009
Introduction to AVR Assembly Language Programming II – ALU and SREG

Reading

The AVR Microcontroller and Embedded Systems using Assembly and C
by Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi

Chapter 2: AVR Architecture and Assembly Language Programming
   Section 2.4: AVR Status Register

Chapter 5: Arithmetic, Logic Instructions, and Programs
   Section 5.1: Arithmetic Instructions
   Section 5.2: Signed Number Concepts and Arithmetic Operations

Chapter 6: AVR Advanced Assembly Language Programming
   Section 6.5: Bit Addressability

Complementary Reading

The following source(s) cover the same material as Chapter 2 of your textbook. They are provided to you in case you want a different viewpoint.

ATMEL document doc8161 "8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash" Section 6.3.1: SREG - AVR Status Register
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Set Architecture (Review)</td>
<td>4</td>
</tr>
<tr>
<td>ALU – Two Operand Instructions</td>
<td>5</td>
</tr>
<tr>
<td>ALU – Single Operand Instructions</td>
<td>6</td>
</tr>
<tr>
<td>ALU Program Example</td>
<td>7</td>
</tr>
<tr>
<td>SREG – AVR Status Register</td>
<td>8</td>
</tr>
<tr>
<td>The SREG Overflow Bit</td>
<td>10</td>
</tr>
<tr>
<td>Computing ALU Status Register Bits – Addition</td>
<td>11</td>
</tr>
<tr>
<td>Computing ALU Status Register Bits – Subtraction</td>
<td>12</td>
</tr>
</tbody>
</table>
Figure 1-5    AVR Central Processing Unit ISA Registers

1 Source: ATmega16 Data Sheet  
http://www.atmel.com/dyn/resources/prod_documents/2466s.pdf page 3
ALU – TWO OPERAND INSTRUCTIONS

- All math (+, -, *, /) and logic (and, or, xor) instructions work with the Register File (register to register).
- Most math and logic instructions have two operands Rd, Rr with register Rd initially containing one of the values to be operated on and ultimately the result of the operation. The initial contents of Rd are therefore destroyed by this operation.

```
add    Rd, Rr    ; Rd = Rd + Rr, You may use any register (R0 - R31).
```

- Some math and logic operations replace the source register Rr with a constant K. Typically denoted by an “i” postfix.

```
subi   Rd, K    ; Rd = Rd - K, You may only registers (R16 - R31).
```

add, adc, adiw  Adds two registers and the contents of the C Flag (adc only) and places the result in the destination register Rd.

sub, sbc, subi, sbci, sbiw  Subtracts the source register Rs or constant K from the source/destination register Rr and subtracts with the C Flag (sbc and sbci only) and places the result in the source/destination register Rd. Think of the C Flag as the Borrow bit within this context.

mul, muls, mulsu, fmul, fmuls, fmulsu  The multiplicand Rd and the multiplier Rr are two registers containing binary or fractional (f-prefix) encoded numbers. Both numbers may be unsigned (mul, fmul), or signed (muls, fmuls). Finally, the multiplicand Rd may be signed with the multiplier Rr unsigned (mulsu, fmulsu). The 16-bit unsigned product is placed in R1 (high byte) and R0 (low byte).  

```
R1:R0 ← Rd x Rs
```

and, andi, or, ori, eor  Performs the logical AND, OR, and XOR operations between the contents of register Rd and register Rr or constant K.

---

2 Source: Atmel 8-bit AVR Instruction Set Document 0856
### ALU – Single Operand Instructions

- All single operand math and logic instructions only need a single register and usually the mnemonic alone is enough to tell you what it does.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com</td>
<td>Rd ← 0xFF - Rd</td>
<td>One’s complement.</td>
</tr>
<tr>
<td>neg</td>
<td>Rd ← 0x00 - Rd</td>
<td>Two’s complement.</td>
</tr>
<tr>
<td>inc</td>
<td>Rd ← Rd + 1</td>
<td>Increment⁴</td>
</tr>
<tr>
<td>dec</td>
<td>Rd ← Rd - 1</td>
<td>Decrement⁴</td>
</tr>
<tr>
<td>clr</td>
<td>Rd ← Rd ⊕ Rd</td>
<td>Clear</td>
</tr>
<tr>
<td>ser</td>
<td>Rd ← 0xFF</td>
<td>Set Register, Limited to r16 – r31</td>
</tr>
<tr>
<td>tst</td>
<td>Rd ← Rd • Rd</td>
<td>Test for Zero or Minus</td>
</tr>
</tbody>
</table>

---

³ Source: Atmel 8-bit AVR Instruction Set Document 0856

⁴ The C Flag in SREG is not affected by the operation.
Write an Assembly program to implement the polynomial expression

\[ B = A^2 + A + 41 \]

```
INCLUDE <m328pdef.inc>

DSEG
A:   .BYTE 1 // 8 bit input
B:   .BYTE 2 // 16 bit output

CSEG

    ; load
lds  r16, A   ; r16 with the value of A
clr  r17      ; r17 with 0
ldi  r18, 41  ; r18 with 41
            ; do something
mul  r16, r16 ; r1:r0 = A^2
add  r0, r16  
adc  r1, r17  ; r1:r0 = A^2 + A
add  r0, r18  
adc  r1, r17  ; r1:r0 = A^2 + A + 41
            ; store
sts B, r0    ; answer byte ordering
sts B+1, r1   ; is little endian
```
SREG – AVR STATUS REGISTER
### SREG – AVR Status Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>T</td>
<td>H</td>
<td>S</td>
<td>V</td>
<td>N</td>
<td>Z</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

**Non ALU**

- **Bit 7 – I: Global Interrupt Enable**
  The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the `reti` instruction. The I-bit can also be set and cleared by the application with the `sei` and `cli` instructions.

- **Bit 6 – T: Bit Copy Storage**
  The Bit Copy instructions `bld` (Bit LoaD) and `bst` (Bit STore) use the T-bit as source or destination. A bit from a register can be copied into T (Rb → T) by the `bst` instruction, and a bit in T can be copied into a bit in a register (T → Rb) by the `bld` instruction.

**ALU**

**Signed two’s complement arithmetic**

- **Bit 4 – S: Sign Bit, S = N ⊕ V**
  Bit set if answer is negative with no errors or if both numbers were negative and error occurred, zero otherwise.

- **Bit 3 – V: Two’s Complement Overflow Flag**
  Bit set if error occurred as the result of an arithmetic operation, zero otherwise.

- **Bit 2 – N: Negative Flag**
  Bit set if result is negative, zero otherwise.

**Unsigned arithmetic**

- **Bit 5 – H: Half Carry Flag**
  Carry from least significant nibble to most significant nibble. Half Carry is useful in BCD arithmetic.

- **Bit 0 – C: Carry Flag**
  The Carry Flag C indicates a carry in an arithmetic operation. Bit set if error occurred as the result of an unsigned arithmetic operation, zero otherwise.

**Arithmetic and Logical**

- **Bit 1 – Z: Zero Flag**
  The Zero Flag Z indicates a zero result in an arithmetic or logic operation.

---

5 Source: ATmega328P Data Sheet Document 8161 Section 6.3 Status Register
THE SREG OVERFLOW BIT

The overflow bit indicates if there was an error caused by the addition or two n-bit 2’s complement numbers, where the n-1 “sign bit” is 1 if the number is negative and 0 if the number is positive. In other words, the sum is outside the range $-2^{n-1}$ to $2^{n-1}-1$.

Another way to recognize an error in addition is to observe that if you add two numbers of the same sign (positive + positive = negative or negative + negative = positive) then an error has occurred.

An overflow condition can never result from the addition of two n-bit numbers of opposite sign (positive + negative or negative + positive).

Here are examples of all four cases for two 8 bit signed numbers.

<table>
<thead>
<tr>
<th>Case</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0b6b5b4b3b2b1b0</td>
<td>0b6b5b4b3b2b1b0</td>
<td>1b6b5b4b3b2b1b0</td>
<td>1b6b5b4b3b2b1b0</td>
</tr>
<tr>
<td></td>
<td>0b6b5b4b3b2b1b0</td>
<td>1b6b5b4b3b2b1b0</td>
<td>0b6b5b4b3b2b1b0</td>
<td>1b6b5b4b3b2b1b0</td>
</tr>
</tbody>
</table>

The variable “$b_n$” simply indicates some binary value and may be 1 or 0. The index of the carry bit ($C_n$) is equal to the carry into bit $b_n$. For example, the carry into $b_0$ is $C_0$ and the carry out of an 8-bit register $b_7$ is $C_8$.

1. Looking first at Case A, a carry cannot be generated out of the sign bit ($C_{n+1}$=0); therefore, if a carry enters the sign bit ($C_n$=1), the sum will be negative and the answer is wrong.

2. For Case B and Case C no error can occur. Observe that in both case B and C because the numbers are contained in an n-bit ($n = 8$) register, we know they are in the range $-2^{n-1}$ to $2^{n-1}$-1 (-128 to 127 for our two 8-bit numbers). Because one number is positive and the other negative, we further know, the answer must be correct.

3. For Case D, a carry will always be generated out of the sign bit $C_{n+1}$=1 (ex. $C_8$ = 1) with the sign bit itself set to 0; therefore, if a carry does not enter the sign bit $C_n$=0 ($C_7$=1) the sum will be positive and the answer will be wrong.

Here is what we have discovered translated into a truth-table.

<table>
<thead>
<tr>
<th>$C_{n+1}$</th>
<th>$C_n$</th>
<th>V</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>may occur for cases A, B, C without error</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>may occur for cases B, C, D without error</td>
</tr>
</tbody>
</table>

Solving for the overflow bit (V) we have, $V = C_{n+1} \oplus C_n$
Computing ALU Status Register Bits – Addition –

\[
\begin{array}{ccccccccc}
& & & & & & 1 & & \\
& & & & & & c_8 & c_7 & c_4 & c_0 \\
& & & & & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
\hline
& & & & & & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

Unsigned

\[
\begin{align*}
76 + 186 &= 06 \\
H &= c_4 = 1 \quad \text{C = c}_8 = 1
\end{align*}
\]

Signed

\[
\begin{align*}
76 - 70 &= 06 \\
N &= b_7 = 0 \quad V = c_8 \oplus c_7 = 0 \\
S &= N \oplus V = 0
\end{align*}
\]

Arithmetic and Logical

\[
Z = 0
\]
### Computing ALU Status Register Bits – Subtraction –

For subtract instructions (sub, subi, sbc, sbci, sbiw), including compare instructions (cp, cpc, cpi, cpse), the carry bit is equal to $C = \overline{C_8}$ and $H = \overline{C_4}$.

Assume the subtract instruction `sub r16, r17` has just been run by the ATmega328P microcontroller. Complete the table provided. The “difference” column should reflect the contents of register r16 after the subtraction operation (leave the answer in 2’s complement form) and not the actual difference (i.e., if done using your calculator).

<table>
<thead>
<tr>
<th>r16</th>
<th>r17</th>
<th>difference</th>
<th>signed relationship</th>
<th>unsigned relationship</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>3B</td>
<td>3B</td>
<td>00</td>
<td>+ = +</td>
<td>=</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3B</td>
<td>15</td>
<td>26</td>
<td>+ &gt; +</td>
<td>&gt;</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>3B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>F6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>F9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>F6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>A5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use AVR Studio simulation software to check your answers.