COMP3221: Microprocessors and Embedded Systems

Lecture 7: Arithmetic and logic Instructions

Overview

- Arithmetic and Logic Instructions in AVR
- Sample AVR Assembly Programs Using AL instructions

AVR Instruction Overview

- Load/store architecture
- At most two operands in each instruction
- Most instructions are two bytes long
- Some instructions are 4 bytes long
- Four Categories:
  - Arithmetic and logic instructions
  - Program control instruction
  - Data transfer instruction
  - Bit and bit test instructions

General-Purpose Registers in AVR

- 32 general-purpose registers
  - named r0, r1, ..., r31 in AVR assembly language
  - Broken into two parts: with 16 registers each, r0 to r15 and r16 to r31.
  - Each register is also assigned a memory address in SRAM space.
  - Register r0 and r26 through r31 have additional functions.
    - r0 is used in the instruction LPM (load program memory)
    - Registers x (r27 : r26), y (r29 : r28) and z (r31 : r30) are used as pointer registers
  - Most instructions that operate on the registers have direct, single cycle access to all general registers. Some instructions such as sbci, subi, cpi, andi, or and ldi operates only on a subset of registers.
General-Purpose Registers in AVR (Cont.)

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>r0</td>
</tr>
<tr>
<td>0x01</td>
<td>r1</td>
</tr>
<tr>
<td>0x1A</td>
<td>r26</td>
</tr>
<tr>
<td>0x1B</td>
<td>r27</td>
</tr>
<tr>
<td>0x1C</td>
<td>r28</td>
</tr>
<tr>
<td>0x1D</td>
<td>r29</td>
</tr>
<tr>
<td>0x1E</td>
<td>r30</td>
</tr>
<tr>
<td>0x1F</td>
<td>r31</td>
</tr>
</tbody>
</table>

x register low byte
x register high byte
y register low byte
y register high byte
z register low byte
z register high byte

The Status Register in AVR

• The Status Register (SREG) contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations.

• SREG is updated after all ALU operations.

• SREG is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The Status Register in AVR (Cont.)

• Bit 7 – I: Global Interrupt Enable
  - The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

• Bit 6 – T: Bit Copy Storage
  - Used to enable and disable interrupts.
  - 1: enabled. 0: disabled.
  - The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts.
The Status Register in AVR (Cont.)

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

- **Bit 5 – H: Half Carry Flag**
  - The Half Carry Flag H indicates a Half Carry (carry from bit 4) in some arithmetic operations.
  - Half Carry is useful in BCD arithmetic.

- **Bit 4 – S: Sign Bit**
  - Exclusive OR between the Negative Flag N and the Two’s Complement Overflow Flag V (S = N ⊕ V).

- **Bit 3 – V: Two’s Complement Overflow Flag**
  - The Two’s Complement Overflow Flag V supports two’s complement arithmetic.

- **Bit 2 – N: Negative Flag**
  - N is the most significant bit of the result.

- **Bit 1 – Z: Zero Flag**
  - Z indicates a zero result in an arithmetic or logic operation. 1: zero. 0: Non-zero.

- **Bit 0 – C: Carry Flag**
  - Its meaning depends on the operation.

  For addition X+Y, it is the carry from the most significant bit. In other words, C = Rd7 • Rr7 + Rr7 • NOT(R7) + NOT(R7) • Rd7, where Rd7 is bit 7 of x, Rr7 is bit 7 of y, R7 is bit 7 of x+y, • is the logical AND and + is the logical OR.

  For subtraction x-y, where x and y are unsigned integer, it indicates if x<y. If x<y, the C=1; otherwise, C=0. In other words, C = NOT(Rd7) • Rr7 + Rr7 • R7 + R7 • NOT(Rd7).
Selected Arithmetic and Logic Instructions

- add, adc, inc
- sub, sbc, dec
- mul, muls, mulsu
- and, or, eor
clr, cbr, cp, cpc, cpi, tst
- com, neg
- Refer to the main textbook (Pages 63~67) and AVR Instruction Set for the complete list of AL instructions.

Add without Carry

- Syntax: add Rd, Rr
- Operands: Rd, Rr ∈ {r0, r1, ..., r31}
- Operation: Rd←Rd + Rr
- Flags affected: H, S, V, N, Z, C
- Encoding: 0000 11rd dddd rrrr
- Words: 1
- Cycles: 1
- Example: add r1, r2 ; Add r2 to r1
  add r28, r28 ; Add r28 to itself

Add with Carry

- Syntax: adc Rd, Rr
- Operands: Rd, Rr ∈ {r0, r1, ..., r31}
- Operation: Rd ← Rd + Rr + C
- Flags affected: H, S, V, N, Z, C
- Encoding: 0001 11rd dddd rrrr
- Words: 1
- Cycles: 1
- Example: add r1, r2 ; Add r2 to r1
  add r28, r28 ; Add r28 to itself
  cpi r22, $4F ; compare r22 to $4F
  brne loop ; Branch to loop if not equal
  nop ; Continue (do nothing

Increment

- Syntax: inc Rd
- Operands: Rd ∈ {r0, r1, ..., r31}
- Operation: Rd←Rd+1
- Flags affected: S, V, N, C
- Encoding: 1001 010d dddd 1011
- Words: 1
- Cycles: 1
- Example: clr r22 ; clear r22
  inc r22 ; Increment r22
  loop: inc r22 ; Branch to loop if not equal
Subtract without Carry

- **Syntax:** `sub Rd, Rr`
- **Operands:** `Rd, Rr ∈ {r0, r1, ..., r31}`
- **Operation:** `Rd ← Rd - Rr`
- **Flags affected:** `H, S, V, N, Z, C`
- **Encoding:** `0011 01rd dddd rrrr`
- **Words:** 1
- **Cycles:** 1
- **Example:** `sub r13, r12 ; Subtract r12 from r13`

Subtract with Carry

- **Syntax:** `sbc Rd, Rr`
- **Operands:** `Rd, Rr ∈ {r0, r1, ..., r31}`
- **Operation:** `Rd ← Rd - Rr - C`
- **Flags affected:** `H, S, V, N, Z, C`
- **Encoding:** `0000 10rd dddd rrrr`
- **Words:** 1
- **Cycles:** 1
- **Example:** `sbc r3, r1 ; Subtract with carry high byte`
  `sbc r2, r0 ; Subtract low byte`
- **Comments:** `sbc` is used in multi-byte subtraction

Decrement

- **Syntax:** `dec Rd`
- **Operands:** `Rd ∈ {r0, r1, ..., r31}`
- **Operation:** `Rd ← Rd - 1`
- **Flags affected:** `S, V, N, Z`
- **Encoding:** `1001 010d dddd 1010`
- **Words:** 1
- **Cycles:** 1
- **Example:** `ldi r17, $10 ; Load constant in r17`
  `loop: add r1, r2 ; Add r2 to r1`
  `dec r17 ; Decrement r17`
  `bne loop; ; Branch to loop if r17≠0`
  `nop ; Continue (do nothing)`

Multiply Unsigned

- **Syntax:** `mul Rd, Rr`
- **Operands:** `Rd, Rr ∈ {r0, r1, ..., r31}`
- **Operation:** `r1, r0 ← Rr*Rd (unsigned * unsigned)`
- **Flags affected:** `Z, C`
- **Encoding:** `1001 11rd dddd rrrr`
- **Words:** 1
- **Cycles:** 2
- **Example:** `mul r6, r5 ; Multiply r6 and r5`
  `mov r6, r1`
  `mov r5, r0 ; Copy result back in r6 : r5`
### Multiply Signed

- **Syntax:** `muls Rd, Rr`
- **Operands:** `Rd, Rr ∈ {r16, r17, ..., r31}`
- **Operation:** `r1, r0 ← Rr*Rd (signed−signed * signed )`
- **Flags affected:** `Z, C`
- **Encoding:** `0000 0010 dddd rrrr`
- **Words:** `1`
- **Cycles:** `2`
- **Example:** `mul r17, r16 ; Multiply r6 and r5
  movw r17:r16, r1:r0 ; Copy result back in r17 : r16`

### Multiply Signed with Unsigned

- **Syntax:** `mulu Rd, Rr`
- **Operands:** `Rd, Rr ∈ {r16, r17, ..., r23}`
- **Operation:** `r1, r0 ← Rr*Rd (signed−signed * unsigned )`
- **Flags affected:** `Z, C`  
  `C` is set if bit 15 of the result is set; cleared otherwise.
- **Encoding:** `0000 0011 0ddd 0rrr`
- **Words:** `1`
- **Cycles:** `2`

### Multiply Signed with Unsigned (Cont.)

Example: Signed multiply of two 16-bit numbers stored in r23:r22 and r21:r20 with 32-bit result stored in r19:r18:r17:r16

How to do?

Let `ah` and `al` be the high byte and low byte, respectively, of the multiplicand and `bh` and `bl` the high byte and low byte, respectively, of the multiplier.

\[
\begin{align*}
  ah : al & \times bh : bl \\
  = (ah \times 2^8 + al) & \times (bh \times 2^8 + bl) \\
  = ah \times bh \times 2^{16} & + al \times bh \times 2^8 + ah \times bl \times 2^8 + al \times bl
\end{align*}
\]

### Multiply Signed with Unsigned (Cont.)

Example: Signed multiply of two 16-bit numbers stored in r23:r22 and r21:r20 with 32-bit result stored in r19:r18:r17:r16

```
muls16x16_32:
  clr r2
  muls r23, r21
  movw r19 : r18, r1 : r0
  mul r22, r20
  movw r17 : r16, r1 : r0
  mulu r23, r20
  sbs r19, r2
  add r17, r0
  adc r18, r1
  adc r19, r2
  mulsu r21, r22
  sbs r19, r2
  add r17, r0
  adc r18, r1
  adc r19, r2
  ret
```
Lower-Case to Upper-Case

.include "m64.def.inc"
.equ size =5
.def counter =r17
.dseg
.org 0x100 ; Set the starting address of data segment to 0x100
Cap_string: .byte 5
.cseg
Low_string: .db "hello"
    ldi zl, low(Low_string<<1) ; Get the low byte of the address of "h"
    ldi zh, high(Low_string<<1) ; Get the high byte of the address of "h"
    ldi yh, high(Cap_string)
    ldi yl, low(Cap_string)
    clr counter ; counter=0

Bitwise AND

- Syntax: and Rd, Rr
- Operands: Rd, Rr ∈ {r0, r1, ..., r31}
- Operation: Rd ← Rd · Rr (Bitwise AND Rd and Rr)
- Flags affected: S, V, N, Z
- Encoding: 0010 00rd dddd rrrr
- Words: 1
- Cycles: 1
- Example:
  ldi r2, 0b00110101
  ldi r16, 1
  and r2, r16 ; r2=0b00000001

Bitwise OR

- Syntax: or Rd, Rr
- Operands: Rd, Rr ∈ {r0, r1, ..., r31}
- Operation: Rd ← Rd v Rr (Bitwise OR Rd and Rr)
- Flags affected: S, V, N, Z
- Encoding: 0010 10rd dddd rrrr
- Words: 1
- Cycles: 1
- Example:
  ldi r15, 0b11110000
  ldi r16, 0b00001111
  or r15, r16 ; Do bitwise or between registers
  ; r15=0b11111111
Bitwise Exclusive-OR

- Syntax: `eor Rd, Rr`
- Operands: `Rd, Rr ∈ {r0, r1, …, r31}`
- Operation: `Rd ← Rr ⊕ Rd` (Bitwise exclusive OR `Rr` and `Rd`)
- Flags affected: `S, V, N, Z`
- Encoding: `0010 01rd dddd rrrr`
- Words: 1
- Cycles: 1
- Example:
  ```
  eor r4, r4 ; Clear r4
  eor r0, r22 ; Bitwise exclusive or between r0 and r22
  ; If r0=0b101011 and r22=0b01001000
  ; then r0=0b11100011
  ```

Clear Bits in Register

- Syntax: `cbr Rd, k`
- Operands: `Rd ∈ {r16, r17, …, r31}` and `0 ≤ k ≤ 255`
- Operation: `Rd ← Rd · ($FF - k)` (Clear the bits specified by `k`)
- Flags affected: `S, V, N, Z`
- Encoding: `0111 wwww dddd wwww (wwwwwwwww=$FF-k)`
- Words: 1
- Cycles: 1
- Example:
  ```
  cbr r4, 11 ; Clear bits 0 and 1 of r4.
  ```

Compare

- Syntax: `cp Rd, Rr`
- Operands: `Rd ∈ {r0, r1, …, r31}`
- Operation: `Rd ← Rr` (Rd is not changed)
- Flags affected: `H, S, V, N, Z, C`
- Encoding: `0001 01rd dddd rrrr`
- Words: 1
- Cycles: 1
- Example:
  ```
  cp r4, r5 ; Compare r4 with r5
  brne noteq ; Branch if r4 ≠ r5
  ...
  noteq: nop ; Branch destination (do nothing)
  ```

Compare with Carry

- Syntax: `cpc Rd, Rr`
- Operands: `Rd ∈ {r0, r1, …, r31}`
- Operation: `Rd ← Rr – C` (Rd is not changed)
- Flags affected: `H, S, V, N, Z, C`
- Encoding: `0001 01rd dddd rrrr`
- Words: 1
- Cycles: 1
- Example:
  ```
  ; Compare r3:r2 with r1:r0
  cp r2, r0 ; Compare low byte
  cpc r3, r1 ; Compare high byte
  brne noteq ; Branch if not equal
  ...
  noteq: nop ; Branch destination (do nothing)
  ```
Compare with Immediate

- Syntax: `cpi Rd, k`
- Operands: `Rd ∈ {r16, r17, ..., r31}` and `0 ≤ k ≤ 255`
- Operation: `Rd – k` (Rd is not changed)
- Flags affected: `H, S, V, N, Z, C`
- Encoding: `0011 kkkk dddd kkkk`
- Words: 1
- Cycles: 1
- Example:
  ```
  cp r19, 30 ; Compare r19 with 30
  brne noteq ; Branch if r19 ≠ 30
  ...
  noteq: nop ; Branch destination (do nothing)
  ```

Test for Zero or Minus

- Syntax: `tst Rd`
- Operands: `Rd ∈ {r0, r1, ..., r31}`
- Operation: `Rd ← Rd · Rd`
- Flags affected: `S, V, N, Z`
- Encoding: `0010 00dd dddd dddd`
- Words: 1
- Cycles: 1
- Example:
  ```
  tst r0 ; Test r0
  breq zero ; Branch if r0=0
  ...
  zero: nop ; Branch destination (do nothing)
  ```

One's Complement

- Syntax: `com Rd`
- Operands: `Rd ∈ {r0, r1, ..., r31}`
- Operation: `Rd ← $FF – Rd`
- Flags affected: `S, V, N, Z`
- Encoding: `1001 010d dddd 0000`
- Words: 1
- Cycles: 1
- Example:
  ```
  com r4 ; Take one's complement of r4
  breq zero ; Branch if zero
  ...
  zero: nop ; Branch destination (do nothing)
  ```

Two's Complement

- Syntax: `neg Rd`
- Operands: `Rd ∈ {r0, r1, ..., r31}`
- Operation: `Rd ← $00 – Rd` (The value of $80 is left unchanged)
- Flags affected: `H, S, V, N, Z, C`
- Encoding: `1001 010d dddd 0001`
- Words: 1
- Cycles: 1
- Example:
  ```
  sub r11, r0 ; Subtract r0 from r11
  brpl positive ; Branch if result positive
  neg r11 ; Take two's complement of r11
  positive: nop ; Branch destination (do nothing)
  ```
Reading Material

1. AVR Instruction Set.