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.
    o r0 is used in the instruction LPM (load program memory)
    o 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, ori 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

### Selected Arithmetic and Logic Instructions

- add, adc, inc
- sub, sbc, dec
- mul, muls, mulsu
- and, or, eor
- clr, cbr, cp, 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`
- **Operand:** `Rd, Rr ∈ {r0, r1, … , r31}`
- **Operation:** `Rd ← Rd + Rr`
- **Flag 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`
- **Operand:** `Rd, Rr ∈ {r0, r1, … , r31}`
- **Operation:** `Rd ← Rd + Rr + C`
- **Flag affected:** H, S, V, N, Z, C
- **Encoding:** 0001 11rd dddd rrrr
- **Words:** 1
- **Cycles:** 1
- **Example:**
  - `add r1 : r0 to r3 : r2`
  - `adc r2, r0` ; Add low byte
  - `adc r3, r1` ; Add high byte
- **Comments:** adc is used in multi-byte addition.
### Increment

- **Syntax:** `inc Rd`
- **Operands:** `Rd ∈ {r0, r1, ..., r31}`
- **Operation:** `Rd ← Rd + 1`
- **Flag affected:** `S, V, N, C`
- **Encoding:** 1001 010d dddd 1011
- **Words:** 1
- **Cycles:** 1
- **Example:**
  ```
  clr r22 ; clear r22
  loop: inc r22 ; Increment r22
         cpi r22, $4F ; compare r22 to $4F
          brne loop ; Branch to loop if not equal
         nop ; Continue (do nothing)
  ```

### Subtract without Carry

- **Syntax:** `sub Rd, Rr`
- **Operands:** `Rd, Rr ∈ {r0, r1, ..., r31}`
- **Operation:** `Rd ← Rd − Rr`
- **Flag affected:** `H, S, V, N, Z, C`
- **Encoding:** 0001 10rd 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`
- **Flag affected:** `H, S, V, N, Z, C`
- **Encoding:** 0000 10rd dddd rrrr
- **Words:** 1
- **Cycles:** 1
- **Example:**
  ```
  Subtract r1:r0 from r3:r2
  sub r2, r0 ; Subtract low byte
  sbc r3, r1 ; Subtract with carry high byte
  ```
  **Comments:** `sbc` is used in multi-byte subtraction

### Decrement

- **Syntax:** `dec Rd`
- **Operands:** `Rd ∈ {r0, r1, ..., r31}`
- **Operation:** `Rd ← Rd − 1`
- **Flag 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
          brne loop ; Branch to loop if r17≠0
         nop ; Continue (do nothing)
  ```
Multiply Unsigned

- Syntax: \texttt{mul Rd, Rr}
- Operands: \(Rd, Rr \in \{r0, r1, \ldots, r31\}\)
- Operation: \(r1, r0\leftarrow Rr*Rd\) (unsigned\(\rightarrow\)unsigned \(*\) unsigned)
- Flag affected: \(Z, C\)
- Encoding: \texttt{1001 11rd dddd rrrr}
- Words: 1
- Cycles: 2
- Example: \texttt{mul r6, r5} ; Multiply r6 and r5
  - \texttt{mov r6, r1}
  - \texttt{mov r5, r0} ; Copy result back in r6 : r5

Multiply Signed

- Syntax: \texttt{muls Rd, Rr}
- Operands: \(Rd, Rr \in \{r16, r17, \ldots, r31\}\)
- Operation: \(r1, r0\leftarrow Rr*Rd\) (signed\(\rightarrow\)signed \(*\) signed)
- Flag affected: \(Z, C\)
- Encoding: \texttt{0000 0010 dddd rrrr}
- Words: 1
- Cycles: 2
- Example: \texttt{mul r17, r16} ; Multiply r6 and r5
  - \texttt{movw r17:r16, r1:r0} ; Copy result back in r17 : r16

Multiply Signed with Unsigned

- Syntax: \texttt{mulsu Rd, Rr}
- Operands: \(Rd, Rr \in \{r16, r17, \ldots, r23\}\)
- Operation: \(r1, r0\leftarrow Rr*Rd\) (signed\(\rightarrow\)unsigned \(*\) unsigned)
- Flag affected: \(Z, C\)
  - \(C\) is set if bit 15 of the result is set; cleared otherwise.
- Encoding: \texttt{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 \(bb\) the high byte and low byte, respectively, of the multiplier.
\[
\begin{align*}
  ah : al & \times bh : bb \\
  &= (ah \times 2^8 + al) \times (bh \times 2^8 + bb) \\
  &= ah*bh*2^{16} + al*bh*2^8 + ah*bb*2^8 + al*bb
\end{align*}
\]
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 ; (signed) ah * (signed) bh
  movw r19 : r18, r1 : r0
  mul r22, r20 ; (unsigned) al * (unsigned) bl
  movw r17 : r16, r1 : r0
  mulsu r23, r20 ; (signed) ah * (unsigned) bl
  sbc r19, r2 ; Trick here (Hint: what does the carry mean here?)
  add r17, r0
  adc r18, r1
  adc r19, r2
  mulsu r21, r22 ; (signed) bh * (unsigned) al
  sbc r19, r2 ; Trick here
  add r17, r0
  adc r18, r1
  adc r19, r2
  ret
```

Bitwise AND

- Syntax: and Rd, Rr
- Operands: Rd, Rr ∈ {r0, r1, ..., r31}
- Operation: Rd ← Rd · Rr (Bitwise AND Rd and Rr)
- Flag affected: S, V, N, Z
- Encoding: 0010 00rd dddd rrrr
- Words: 1
- Cycles: 1
- Example:
  ```
  ldi r2, 0b00110101
  ldi r16, 1
  and r15, r16 ; r2=0b00000001
  ```

Bitwise OR

- Syntax: or Rd, Rr
- Operands: Rd, Rr ∈ {r0, r1, ..., r31}
- Operation: Rd ← Rd ∨ Rr (Bitwise OR Rd and Rr)
- Flag 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 ← Rd ⊕ Rr (Bitwise exclusive OR Rd and Rr)
- Flag 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=0b1010111 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)
- Flag affected: S, V, N, Z
- Encoding: 0111 wwww dddd wwww ($wwwwww = \$FF-k$)
- 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 ← Rd – Rr (Rd is not changed)
- Flag affected: H, S, V, N, Z, C
- Encoding: 0001 01rd dddd rrrr
- Words: 1
- Cycles: 1
- Example:
  cp r4, r5 ; Compare r4 with r19
  brne noteq ; Branch if r4 ≠ r19
  ...
  noteq: nop ; Branch destination (do nothing)

Compare with Immediate

- Syntax: cpi Rd, k
- Operands: Rd ∈ \{r16, r17, ..., r31\} and 0 ≤ k ≤ 255
- Operation: Rd ← Rd – k (Rd is not changed)
- Flag affected: H, S, V, N, Z, C
- Encoding: 0011 kkkk dddd kkkk
- Words: 1
- Cycles: 1
- Example:
  cpi 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
- Flag 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 ← SFF – Rd`
- Flag 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)
- Flag affected: `H, S, V, N, Z, C`
  
  **H**: `R3 + Rd3`
  Set if there was a borrow from bit 3; cleared otherwise
- 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)
  ```