Disclaimer

Systemyde International Corporation reserves the right to make changes at any time, without notice, to improve design or performance and provide the best product possible. Systemyde International Corporation makes no warrant for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make any commitment to update the information contained herein.

Systemyde International Corporation products are not authorized for use in life support devices or systems unless a specific written agreement pertaining to such use is executed between the manufacturer and the President of Systemyde International Corporation. Nothing contained herein shall be construed as a recommendation to use any product in violation of existing patents, copyrights or other rights of third parties. No license is granted by implication or otherwise under any patent, patent rights or other rights, of Systemyde International Corporation. All trademarks are trademarks of their respective companies.

Every effort has been made to ensure the accuracy of the information contain herein. If you find errors or inconsistencies please bring them to our attention. In all cases, however, the Verilog HDL source code for the YVR design defines “proper operation”.

Copyright © 2014, Systemyde International Corporation. All rights reserved.

Notice:

“Atmel” and “AVR” are registered trademarks of Atmel, Inc. All uses of these terms in this document are to be construed as adjectives, whether or not the noun “microprocessor”, “microcontroller”, “CPU” or “device” are actually present.
Table of Contents

1. Introduction .................................................................................................................. 3
2. Features ...................................................................................................................... 5
3. Pin Descriptions ......................................................................................................... 7
4. External Timing ........................................................................................................... 11
5. Instruction Set ............................................................................................................ 17
6. Special Function Registers ......................................................................................... 99
# Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Changes</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/10/2014</td>
<td>Preliminary issue</td>
<td></td>
</tr>
</tbody>
</table>
This publication documents the operation of the YVR microcontroller. This CPU design is supplied in Verilog HDL and can be implemented in any technology supported by a logic synthesis tool that accepts Verilog HDL. Included in the design package is a test bench that exercises all instructions, flag settings, and representative data patterns. The test patterns should achieve at least 95% fault coverage.

The YVR CPU was designed in a clean-room environment and is instruction-set compatible with the Atmel AVR line of microcontrollers. Only publicly available documentation was used to create this design so there may be minor differences where the public documentation is misleading or lacking. The instruction execution times are not identical between the two designs. The YVR CPU operates with a consistent two-clock-cycle machine cycle, while the AVR microcontroller uses a machine cycle with one clock cycle.

This document should always be used as the final word on the operation of the YVR CPU, but it is useful to refer to the Atmel documentation if the description given here is too cryptic. The AVR architecture is over twenty years old, so it is assumed that it is already at least somewhat familiar to the reader.

The YVR CPU is accompanied by full design documentation, in the form of a large spreadsheet, which describes nearly every facet of the internal operation of the processor. This provides knowledgeable users the opportunity to customize the design for unique application requirements.
Features

* Fully functional synthesizable Verilog HDL version of the AVR CPU
  - Peripheral functions not included in base design
* Vendor and technology independent
* Software compatible with numerous industry-standard processors
* Static, fully synchronous design uses no 3-state buses
* Uniform 2 clock-cycle machine cycle
* Memory interfaces match common synchronous FPGA and ASIC memory timing
* Program memory and External RAM memory utilize separate interfaces
* Separate Special Function Register (I/O) bus with dedicated strobes
* Internal Register File & RAM uses FPGA memory macros
  - Lattice ICE40 version uses two memory macros, for 512 bytes of RAM
  - Microsemi A3P version uses four memory macros, for 1024 bytes of RAM
  - Xilinx XC3S version uses four memory macros, for 4096 bytes of RAM
* Full design documentation included
* Verilog simulation and test suite included
The YVR CPU design makes no attempt to match the signals or timing present on the AVR microprocessor. Rather, all of the bus interfaces and signals are optimized for use in either an FPGA or an ASIC. The interface signals for the design are detailed below.

**CLOCK and RESET**

The design uses a single clock and single reset. No clock gating is employed in the design.

**clk** (input, active-High) The rising edge of the Master Clock samples all inputs except for the asynchronous reset and all outputs normally change in response to the rising edge of the Master Clock.

**resetb** (input, active-Low) The Master Reset signal is used to initialize all state flip-flops, and user registers consistent with the original AVR design. This is an asynchronous signal, but the trailing edge should be synchronized with the rising edge of the Master Clock.

**PROGRAM MEMORY**

The interface for Program Memory consists of a 22-bit address bus, a 16-bit bus for read data, a 16-bit bus for write data and dedicated read and write strobes. The cycle time for this bus is two clock cycles. Program memory is organized as words, but can be accessed as bytes via the LPM and ELPM instructions. In this case the least-significant byte of a word has an even address.

**ph** (output, active-High) The Bus Phase signal is High during the first clock of a machine cycle and Low during the second clock of a machine cycle.

**pmem_addr** (output, 22-bit bus) The Program Memory Address bus carries the address for all Program Memory read and write transactions.

**pmem_rd** (output, active-High) The Program Memory Read signal is one clock cycle wide and identifies the data transfer clock cycle for Program Memory read transactions.
pmem_rdata (input, 16-bit bus) The Program Memory Read Data bus is sampled during Program Memory read transactions.

pmem_wdata (output, 16-bit bus) The Program Memory Write Data bus carries the output data for Program Memory write transactions. To conserve power, this bus only changes state as required for a memory write.

pmem_wr (output, active-High) The Program Memory Write signal is one clock cycle wide and identifies the data transfer clock cycle for Program Memory write transactions.

DATA MEMORY

The interface for Data Memory consists of a 24-bit address bus, an 8-bit bus for read data, an 8-bit bus for write data and dedicated read and write strobes. The interface for Data Memory is used for all explicit Data Memory read and write transactions, even those to the regions of data memory reserved for CPU registers (0x000000-0x00001F) or I/O registers (0x000020-0x00005F). However, implicit accesses (that is, anything other than LD or STD) of CPU registers or I/O registers do not use this bus. The cycle time for this bus is two clock cycles.

dmem_addr (output, 24-bit bus) The Data Memory Address bus carries the address all explicit Data Memory read and write transactions.

dmem_rd (output, active-High) The Data Memory Read signal is one clock cycle wide and identifies the data transfer clock cycle for Data Memory read transactions.

dmem_rdata (input, 8-bit bus) The Data Memory Read Data bus is sampled during explicit Data Memory read transactions.

dmem_wdata (output, 8-bit bus) The Data Memory Write Data bus carries the output data for Data Memory write transactions. To conserve power, this bus only changes state as required for a memory write.

dmem_wr (output, active-High) The Data Memory Write signal is one clock cycle wide and identifies the data transfer clock cycle for Data Memory write transactions.

SPECIAL FUNCTION (I/O) REGISTERS

Access to I/O Registers is via the Special Function Register (SFR) bus. The SFR bus operates with a one-clock cycle time, and simultaneous reads and writes are not possible. The
SFR bus is used for Data addresses in the range 0x0020-0x005F, corresponding to I/O addresses 0x00-0x3F. A number of I/O registers are implemented internal to the design, and do not use this bus, but are directly connected internally.

**sfr_addr** (output, 6-bit bus) The Special Function Register Address bus carries the address for Input and Output transactions. This bus only valid during Input or Output transfers.

**sfr_rd** (output, active-High) The Special Function Register Read Enable signal identifies data transfer clock cycles for Special Function Register read transactions.

**sfr_rdata** (input, 8-bit bus) The Special Function Register Read Data bus is sampled by the clock when the **sfr_rd** signal is active and an external Special Function Register is addressed.

**sfr_wdata** (output, 8-bit bus) The Special Function Register Write Data bus carries the output data for Special Function Register write transactions. This bus is valid only during Special Function write operations.

**sfr_wr** (output, active-High) The Special Function Register Write Enable signal identifies data transfer clock cycles for Special Function Register write transactions.

**INTERRUPTS**

The YVR CPU design supports eight “external” interrupt requests with fixed interrupt vectors, and an “internal” interrupt request that uses an externally-supplied interrupt vector. The “internal” interrupt request are intended for use with on-chip peripherals beyond those native to the design.

**int_ack** (output, active-High) The External Interrupt Acknowledge signal is active for two clock cycles to indicate that an interrupt acknowledge sequence is in progress. If the “internal” interrupt is being acknowledged the **ivec_rd** signal will be active during the second clock to indicate that the CPU is sampling the **ivec_rdata** bus.

**ireq_ext** (input, 8-bit bus, active-High) The eight External Interrupt Request signals are level-sensitive, and the interrupt request must be de-asserted before the end of the interrupt service routine. Edge-triggering can be implemented externally to the core.

**ireq_int** (input, active-High) The Internal Interrupt Request signal is level-sensitive. This input is not synchronized, because it is assumed that it will be generated by on-chip peripheral devices.
ivec_rd (output, active-High) The Interrupt Vector Read Enable signal identifies data transfer clock cycles for Interrupt Acknowledge transactions.

ivec_rdata (input, 8-bit bus) The Interrupt Vector bus is sampled during an interrupt acknowledge transaction for the internal interrupt. This interrupt vector is loaded into the least-significant byte of the Program Counter at the end of the interrupt acknowledge transaction. The other byte(s) of the Program counter are set to all zeros at the same time.

reti_pls (output, active-High) The Return-from-Interrupt signal is active for one clock cycle during the execution of the RETI instruction.

MISCELLANEOUS

break_pls (output, active-High) The Break Instruction signal is active for one clock cycle when the Break instruction is executed.

sleep_pls (output, active-High) The Sleep Instruction signal is active for one clock cycle when the Sleep instruction is executed.

wdr_pls (output, active-High) The WDR Instruction signal is active for one clock cycle when the Watchdog Reset instruction is executed.

gpior0_reg (output, 8-bit bus) This is the contents of the GPIOR0 register in the SFR address space (I/O address 0x1E).

gpior1_reg (output, 8-bit bus) This is the contents of the GPIOR1 register in the SFR address space (I/O address 0x2A).

gpior2_reg (output, 8-bit bus) This is the contents of the GPIOR2 register in the SFR address space (I/O address 0x2B).
External Timing

The YVR CPU design uses a uniform two-clock-cycle machine cycle. This consistent timing simplifies the design of logic external to the CPU makes it easier to track the state of the CPU.

The program and data memory interface timing and signals are designed to make it easy to interface to standard ASIC and FPGA memories. These interfaces use separate read and write strobes.

The SFR interface is suitable for connecting directly to hardware registers (flip-flops.)

In the diagrams below only the relevant signals are shown for each transaction. All other signals are either inactive or hold the previous value.
Program Memory Read and Write

The figure below shows Program Memory read and write transactions.
Data Memory Read and Write

The figures below show read and write transactions for Data Memory. Data Memory transactions are always aligned with Program Memory transactions, so the \( \text{ph} \) signal can also be used to determine the phase for Data Memory transactions.
SFR Read and Write

The figures below show read and write transactions on the SFR interface. Unlike the Program Memory and Data Memory interfaces, the SFR interface uses one clock-cycle timing. The `sfr_addr` and `sfr_wdata` buses are driven with all zeros when not in use, but since they are also used to carry information to the internal Special Function Registers as well as the CPU registers, these buses will carry non-zero information at other times. External Special Function Registers must use the `sfr_rd` and `sfr_wr` strobes to read and write information.
The figure below shows the interrupt acknowledge transaction, including the preceding aborted instruction fetch and subsequent instruction fetch for the service routine. The `dmem_addr` and `dmem_wdata` buses are also shown to indicate where the internal RAM is written. Timing is the same in the case of a 16-bit Program Counter, but only two bytes are pushed to the stack.
The Reset state is entered immediately when the resetb signal goes Low, independent of the current state, and this state continues until the first rising edge of clk after the resetb signal is de-asserted. At this rising edge there is a one clock cycle transient state to set up the internal pipeline controls, and on the next clock the processor begins fetching the first instruction from address 0x0000.

The minimum width of the resetb signal is set by the flip-flops used in the design. The setup time for the resetb signal to the rising edge of the clk signal is likewise determined by the flip-flops used in the design.
This chapter presents the assembly language syntax, addressing modes, flag settings, binary encoding, and execution time for the YVR instruction set. The entire instruction set is presented in alphabetical order.

The assembly language syntax is identical to that used by the original Atmel assembler. Different assembler programs may or may not use identical syntax. The syntax is presented generically at the beginning of each instruction, with the details presented for each addressing mode later in each entry.

The operation of each instruction is specified in a format similar to Verilog HDL for minimum ambiguity, but no descriptive text or examples are included.

The effect of the instruction on each flag is listed, with a brief description. The flags are organized as shown below in the Status Register:

<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>
</table>

These flags have the following meanings:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Carry</td>
</tr>
<tr>
<td>Z</td>
<td>Zero</td>
</tr>
<tr>
<td>N</td>
<td>Negative</td>
</tr>
<tr>
<td>V</td>
<td>Overflow</td>
</tr>
<tr>
<td>S</td>
<td>N ^ V</td>
</tr>
<tr>
<td>H</td>
<td>Half Carry</td>
</tr>
<tr>
<td>T</td>
<td>Transfer Bit</td>
</tr>
<tr>
<td>I</td>
<td>Global Interrupt Enable</td>
</tr>
</tbody>
</table>
Fields in the instruction are listed using shortcuts for common fields. These shortcuts should be self-explanatory in most cases, but will be detailed here for completeness.

The most common field in an instruction specifies a CPU register, employing one of the following encodings:

<table>
<thead>
<tr>
<th>Rd, Rr (5-bit field)</th>
<th>Rd (4-bit field)</th>
<th>Rd (4-bit field)</th>
<th>Rd (3-bit field)</th>
<th>Rd (2-bit field)</th>
<th>Register</th>
<th>Data Memory address</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0000</td>
<td>R0</td>
<td>0x0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001</td>
<td></td>
<td>R1</td>
<td>0x0001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010</td>
<td>0001</td>
<td>R2</td>
<td>0x0002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00011</td>
<td></td>
<td>R3</td>
<td>0x0003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100</td>
<td>0010</td>
<td>R4</td>
<td>0x0004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101</td>
<td></td>
<td>R5</td>
<td>0x0005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00110</td>
<td>0011</td>
<td>R6</td>
<td>0x0006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00111</td>
<td></td>
<td>R7</td>
<td>0x0007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01000</td>
<td>0100</td>
<td>R8</td>
<td>0x0008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01001</td>
<td></td>
<td>R9</td>
<td>0x0009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01010</td>
<td>0101</td>
<td>R10</td>
<td>0x000A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01011</td>
<td></td>
<td>R11</td>
<td>0x000B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100</td>
<td>0110</td>
<td>R12</td>
<td>0x000C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01101</td>
<td></td>
<td>R13</td>
<td>0x000D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110</td>
<td>0111</td>
<td>R14</td>
<td>0x000E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01111</td>
<td></td>
<td>R15</td>
<td>0x000F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>0000 1000</td>
<td>R16</td>
<td>0x0010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10001</td>
<td>0001 001</td>
<td>R17</td>
<td>0x0011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10010</td>
<td>0010 1001</td>
<td>R18</td>
<td>0x0012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10011</td>
<td>0011 011</td>
<td>R19</td>
<td>0x0013</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100</td>
<td>0100 1010</td>
<td>R20</td>
<td>0x0014</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10101</td>
<td>0101 101</td>
<td>R21</td>
<td>0x0015</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10110</td>
<td>0110 110</td>
<td>R22</td>
<td>0x0016</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10111</td>
<td>0111 111</td>
<td>R23</td>
<td>0x0017</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11000</td>
<td>1000 1100</td>
<td>R24</td>
<td>0x0018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11001</td>
<td>1001</td>
<td>R25</td>
<td>0x0019</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11010</td>
<td>1010 1101</td>
<td>R26</td>
<td>0x001A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11011</td>
<td>1011</td>
<td>R27</td>
<td>0x001B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11100</td>
<td>1100 1110</td>
<td>R28</td>
<td>0x001C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11101</td>
<td>1101</td>
<td>R29</td>
<td>0x001D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11110</td>
<td>1110 1111</td>
<td>R30</td>
<td>0x001E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11111</td>
<td>1111</td>
<td>R31</td>
<td>0x001F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bits within a byte are selected by the usual encoding:

<table>
<thead>
<tr>
<th>b, s</th>
<th>bit selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>bit 0</td>
</tr>
<tr>
<td>001</td>
<td>bit 1</td>
</tr>
<tr>
<td>010</td>
<td>bit 2</td>
</tr>
<tr>
<td>011</td>
<td>bit 3</td>
</tr>
<tr>
<td>100</td>
<td>bit 4</td>
</tr>
<tr>
<td>101</td>
<td>bit 5</td>
</tr>
<tr>
<td>110</td>
<td>bit 6</td>
</tr>
<tr>
<td>111</td>
<td>bit 7</td>
</tr>
</tbody>
</table>

CPU registers as well as special function registers are mapped to the beginning of the Data Memory address space:

<table>
<thead>
<tr>
<th>Data Memory address</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 - 0x001F</td>
<td>R0 - R31</td>
</tr>
<tr>
<td>0x0020 - 0x005F</td>
<td>Special Function Registers 0 - 3F</td>
</tr>
<tr>
<td>&gt; 0x0060</td>
<td>External Data Memory</td>
</tr>
</tbody>
</table>

The execution time for instructions is always a multiple of two clocks.
**ADC**

Add With Carry

ADC Rd, Rr

Rd: R0-R31
Rr: R0-R31

Operation: Rd <= Rd + Rr + C

Flags:
- **I**: Unaffected.
- **T**: Unaffected.
- **H**: Set if arithmetic carry out of bit 3; cleared otherwise.
- **S**: $N \lor V$
- **V**: Set if arithmetic overflow; cleared otherwise.
- **N**: Set if bit 7 of the result is set; cleared otherwise.
- **Z**: Set if the result is zero; cleared otherwise.
- **C**: Set if arithmetic carry out of bit 7; cleared otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC Rd, Rr</td>
<td>000111rd_dddrrrr</td>
<td>2</td>
</tr>
</tbody>
</table>
**ADD**

**Add**

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD Rd, Rr</td>
<td>000011rd_dddrrrr</td>
<td>2</td>
</tr>
</tbody>
</table>

**ADD Rd, Rr**

Rd: R0-R31
Rr: R0-R31

**Operation:**

Rd <= Rd + Rr

**Flags:**

I: Unaffected.
T: Unaffected.
H: Set if arithmetic carry out of bit 3; cleared otherwise.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic carry out of bit 7; cleared otherwise.
ADIW
Add Word (Immediate)

ADIW Rd+1:Rd, K
Rd: 24, 26, 28, 30
K: 0-63 (6 bits)

Operation:
{Rd+, Rd} <= {Rd+, Rd} + K

Flags:
I: Unaffected.
T: Unaffected.
H: Unaffected.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 15 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic carry out of bit 15; cleared otherwise.

Assembly Syntax Encoding Clocks

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADIW Rd+1:Rd, K</td>
<td>10010110_KKddKKKK</td>
<td>2</td>
</tr>
</tbody>
</table>
**AND**

Logical AND

\[ \text{AND} \text{ Rd, Rr} \]

Rd: R0-R31
Rr: R0-R31

**Operation:**

\[ \text{Rd} <= \text{Rd} \land \text{Rr} \]

**Flags:**

I: Unaffected.
T: Unaffected.
H: Unaffected.
S: N ^ V
V: Cleared.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Unaffected.

---

**Assembly Syntax Encoding Clocks**

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND Rd, Rr</td>
<td>001000rd_dddrrrr</td>
<td>2</td>
</tr>
</tbody>
</table>
**ANDI**

Logical AND (Immediate)

**Operation:**

\[
\text{Rd} \leq \text{Rd} \& \text{K}
\]

**Flags:**

- **I:** Unaffected.
- **T:** Unaffected.
- **H:** Unaffected.
- **S:** \(N \oplus V\)
- **V:** Cleared.
- **N:** Set if bit 7 of the result is set; cleared otherwise.
- **Z:** Set if the result is zero; cleared otherwise.
- **C:** Unaffected.

**Assembly Syntax Encoding Clocks**

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDI Rd, K</td>
<td>0111KKK_kdddKKK</td>
<td>2</td>
</tr>
</tbody>
</table>
ASR
Arithmetic Shift Right

ASR Rd
Rd: R0-R31

Operation:
{Rd, C} <= {Rd[7], Rd}

Flags:
I: Unaffected.
T: Unaffected.
H: Unaffected.
S: N ^ V
V: N ^ C
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Loaded with the contents of Rd[0] before the shift.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR Rd</td>
<td>1001010d_ddd0101</td>
<td>2</td>
</tr>
</tbody>
</table>
BCLR
Clear Bit (in SREG)

BCLR s

s: 0-7

Operation: SREG[s] <= 0

Flags:

I: Cleared if s=111; unaffected otherwise.
T: Cleared if s=110; unaffected otherwise.
H: Cleared if s=101; unaffected otherwise.
S: Cleared if s=100; unaffected otherwise.
V: Cleared if s=011; unaffected otherwise.
N: Cleared if s=010; unaffected otherwise.
Z: Cleared if s=001; unaffected otherwise.
C: Cleared if s=000; unaffected otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCLR s</td>
<td>10010100_1sss1000</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes:

1. Most AVR assemblers support dedicated mnemonics for each individual SREG bit clear instruction.

2. When the I flag (bit 7) is being cleared by this instruction interrupts are not sampled, so an interrupt that occurs simultaneously with this instruction will not be accepted.
**BLD**

Load Bit from T

**BLD Rd, b**

- **Rd**: R0-R31
- **b**: 0-7

**Operation:**

\[ \text{Rd}[b] \leq T \]

**Flags:**

- No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLD Rd, b</td>
<td>1111100d_dddd0bb</td>
<td>2</td>
</tr>
</tbody>
</table>
**BRBC**

Branch If Bit Clear (in SREG)

<table>
<thead>
<tr>
<th>BRBC s, K</th>
<th>s: 0-7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>K: -63 to +64 (7 bits)</td>
</tr>
</tbody>
</table>

**Operation:**

if (SREG[s] = 0) then PC <= PC + K + 1

**Flags:**

No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRBC s, K</td>
<td>111101kk_kkkksss</td>
<td>2 (4)</td>
</tr>
</tbody>
</table>

**Notes:**

1. The execution time is two clocks if the branch is not taken, and four clocks if the branch is taken.
BRBS
Branch If Bit Set (in SREG)

BRBS s, K

s: 0-7
K: -63 to +64 (7 bits)

Operation: if \( \text{SREG}[s] = 1 \) then \( \text{PC} \leq \text{PC} + K + 1 \)

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRBS s, K</td>
<td>[111100kk_kkkkkss]</td>
<td>2 (4)</td>
</tr>
</tbody>
</table>

Notes:

1. The execution time is two clocks if the branch is not taken, and four clocks if the branch is taken.
**BREAK**

Break

**Operation:** break_pls <= 1

**Flags:** No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAK</td>
<td>10010101_10011000</td>
<td>2</td>
</tr>
</tbody>
</table>

**Notes:**

1. The `break_pls` signal is activated for one clock cycle.
BSET
Set Bit (in SREG)

BSET s

Operation: SREG[s] <= 1

Flags: I: Set if s=111; unaffected otherwise.
T: Set if s=110; unaffected otherwise.
H: Set if s=101; unaffected otherwise.
S: Set if s=100; unaffected otherwise.
V: Set if s=011; unaffected otherwise.
N: Set if s=010; unaffected otherwise.
Z: Set if s=001; unaffected otherwise.
C: Set if s=000; unaffected otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSET s</td>
<td>10010100_0sss1000</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes:

1. Most AVR assemblers support dedicated mnemonics for each individual SREG bit set instruction.

2. When the I flag (bit 7) is being set by this instruction interrupts are not sampled, so an interrupt that occurs simultaneously with this instruction (if the interrupt was already enabled) will not be accepted.
BST
Load T from Bit

BST Rd, b
Rd: R0-R31
b: 0-7

Operation: T <= Rd[b]

Flags:
I: Unaffected.
T: Set if bit b in register Rd is set; cleared otherwise.
H: Unaffected.
S: Unaffected.
V: Unaffected.
N: Unaffected.
Z: Unaffected.
C: Unaffected.

Assembly Syntax | Encoding | Clocks
---|---|---
BST Rd, b | 1111101d_ddd0bbb | 2
CALL
Call Subroutine (Direct)

CALL k
k: 16-bit or 22-bit

Operation:
@SP <= PC[7:0]
SP <= SP - 1
@SP <= PC[15:8]
SP <= SP - 1
if (PC22_EN) begin
   @SP <= PC[21:16]
   SP <= SP - 1
end
PC <= k

Flags:
No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL k</td>
<td>1001010k_kkkk111k</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>kkkkkkkk_kkkkkkkk</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. The timing is independent of the width of the PC.
2. The 22 bit PC is only implemented when the PC22_EN compile option is selected.
CBI
Clear I/O Bit

CBI A, b
A: 0-31 (5 bits)
b: 0-7

Operation: @A[b] <= 0

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBI A, b</td>
<td>10011000_AAAAAabbb</td>
<td>2</td>
</tr>
</tbody>
</table>
COM

Complement

Complement
Rd Rd: R0-R31
Operation: Rd <= -Rd

Flags: I: Unaffected.
       T: Unaffected.
       H: Unaffected.
       S: Set.
       V: Cleared.
       N: Set if bit 7 of the result is set; cleared otherwise.
       Z: Set if the result is zero; cleared otherwise.
       C: Set.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM Rd</td>
<td>1001010d_dddd0000</td>
<td>2</td>
</tr>
</tbody>
</table>
CP
Compare

CP Rd, Rr

Rd: R0-R31
Rr: R0-R31

Operation: Rd - Rr

Flags:
I: Unaffected.
T: Unaffected.
H: Set if arithmetic carry out of bit 3; cleared otherwise.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic carry out of bit 7; cleared otherwise.

Assembly Syntax | Encoding | Clocks
--- | --- | ---
CP Rd, Rr | 000101rd_dddrrrr | 2
CPC Rd, Rr

Rd: R0-R31
Rr: R0-R31

Operation: Rd - Rr - C

Flags:

I: Unaffected.
T: Unaffected.
H: Set if arithmetic carry out of bit 3; cleared otherwise.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic carry out of bit 7; cleared otherwise.

Assembly Syntax | Encoding | Clocks
--- | --- | ---
CPC Rd, Rr | 000001rd_ddddrrrr | 2
CPI

Compare (Immediate)

CPI Rd, K

Rd: R16-R31
K: 0-255 (8 bits)

Operation: Rd - K

Flags:

I: Unaffected.
T: Unaffected.
H: Set if arithmetic carry out of bit 3; cleared otherwise.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic carry out of bit 7; cleared otherwise.

Assembly Syntax | Encoding | Clocks
--- | --- | ---
CPI Rd, K | 0011KKKK_ddddKKKK | 2
CPSE
Compare, Skip if Equal

CPSI Rd, Rr
Rd: R0-R31
Rr: R0-R31

Operation: if (Rd = Rr) skip next instruction

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPSE Rd, Rr</td>
<td>000100rd_ddddrrrr</td>
<td>2 (4/6)</td>
</tr>
</tbody>
</table>

Notes:

1. This instruction executes in two clocks if the next instruction is not skipped, in four clocks if the next instruction is skipped (one word instruction), or six clocks if the next instruction is skipped (two word instruction).

2. Interrupts are not sampled by this instruction, so an interrupt that occurs simultaneously with this instruction will not be accepted until after this instruction (or the subsequent non-skipped instruction) completes.
DEC
Decrement

DEC Rd
Rd: R0-R31

Operation: Rd <= Rd - 1

Flags:  
I: Unaffected.  
T: Unaffected.  
H: Unaffected.  
S: N ^ V  
V: Set if arithmetic overflow; cleared otherwise.  
N: Set if bit 7 of the result is set; cleared otherwise.  
Z: Set if the result is zero; cleared otherwise.  
C: Unaffected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC Rd</td>
<td>1001010d_dddd1010</td>
<td>2</td>
</tr>
</tbody>
</table>
EICALL

Call Subroutine (Indirect, using EIND and Z)

EICALL

Operation:

@SP <= PC[7:0]
SP <= SP - 1
@SP <= PC[15:8]
SP <= SP - 1
if (PC22_EN) begin
   @SP <= PC[21:16]
   SP <= SP - 1
end
PC <= {EIND, Z}

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>EICALL</td>
<td>10010101_00011001</td>
<td>6</td>
</tr>
</tbody>
</table>

Notes:

1. The timing is independent of the width of the PC.
2. The 22 bit PC is only implemented when the PC22_EN compile option is selected.
EIJMP
Jump (Indirect, using EIND and Z)

Operation: PC <= (EIND, Z)

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIJMP</td>
<td>10010100_00011001</td>
<td>4</td>
</tr>
</tbody>
</table>

Notes:

1. The timing is independent of the width of the PC.

2. This instruction is identical to IJMP when the PC22_EN compile option is not selected.
Load From Program Memory (Indirect, using RAMPZ and Z)

ELPM

Case 1
ELPM
Rd, Z
Rd: R0-R31
R0 <= @{RAMPZ, Z}

Case 2
ELPM Rd, Z
Rd: R0-R31
Rd <= @{RAMPZ, Z}

Case 3
ELPM Rd, Z+
Rd <= @{RAMPZ, Z}
(RAMPZ, Z) <= (RAMPZ, Z) + 1

Flags:
No flags affected.

Assembly Syntax | Encoding | Clocks |
--- | --- | --- |
Case 1 | ELPM | 10010101_11011000 | 6 |
Case 2 | ELPM Rd, Z | 1001000d_dddd0110 | 6 |
Case 3 | ELPM Rd, Z+ | 1001000d_dddd0111 | 6 |
EOR

Exclusive-OR

EOR Rd, Rr
Rd: R0-R31
Rr: R0-R31

Operation: Rd <= Rd ^ Rr

Flags: I: Unaffected.
       T: Unaffected.
       H: Unaffected.
       S: N ^ V
       V: Cleared.
       N: Set if bit 7 of the result is set; cleared otherwise.
       Z: Set if the result is zero; cleared otherwise.
       C: Unaffected.

Assembly Syntax | Encoding | Clocks
-----------------|----------|-------
EOR Rd, Rr       | 001001rd_dddrerrr | 2
FMUL Rd, Rr  
Rd: R16-R23  
Rr: R16-R23  

Operation: \[ \{R1, R0\} <= (\text{Rd} \times \text{Rr}) \ll 1 \]

Flags:  
- **I**: Unaffected.  
- **T**: Unaffected.  
- **H**: Unaffected.  
- **S**: Unaffected.  
- **V**: Unaffected.  
- **N**: Unaffected.  
- **Z**: Set if the result is zero; cleared otherwise.  
- **C**: Set if bit 15 of the result is set before the shift; cleared otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMUL Rd, Rr</td>
<td>00000011_0dd1rrr</td>
<td>4</td>
</tr>
</tbody>
</table>

Notes:  
1. Rd and Rr are in unsigned 1.7 format. The result is in 1.15 unsigned format.
FMULS
Fractional Multiply Signed

FMULS Rd, Rr
Rd: R16-R23
Rr: R16-R23

Operation:
\{R1, R0\} <= (Rd * Rr) << 1

Flags:
I: Unaffected.
T: Unaffected.
H: Unaffected.
S: Unaffected.
V: Unaffected.
N: Unaffected.
Z: Set if the result is zero; cleared otherwise.
C: Set if bit 15 of the result is set before the shift; cleared otherwise.

Assembly Syntax Encoding Clocks
FMULS Rd, Rr 00000011_1ddd0rrr 4

Notes:
1. Rd and Rs are in signed 1.7 format. The result is in 1.15 signed format.
FMULSU
Fractional Multiply Signed with Unsigned

FMULSU Rd, Rr
Rd: R16-R23
Rr: R16-R23

Operation: \[ \{R1, R0\} \leq (Rd \times Rr) \ll 1 \]

Flags:
- I: Unaffected.
- T: Unaffected.
- H: Unaffected.
- S: Unaffected.
- V: Unaffected.
- N: Unaffected.
- Z: Set if the result is zero; cleared otherwise.
- C: Set if bit 15 of the result is set before the shift; cleared otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMULSU Rd, Rr</td>
<td>00000011_1dddlrrr</td>
<td>4</td>
</tr>
</tbody>
</table>

Notes:
1. Rd is signed 1.7 format. Rr is unsigned 1.7 format. The result is in 1.15 signed format.
ICALL

Call Subroutine (Indirect, using Z)

ICALL

Operation: @SP <= PC[7:0]
SP <= SP - 1
@SP <= PC[15:8]
SP <= SP - 1
if (PC22_EN) begin
    @SP <= PC[21:16]
    SP <= SP - 1
end
PC <= {6'h00, Z}

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICALL</td>
<td>10010101_00001001</td>
<td>4 (6)</td>
</tr>
</tbody>
</table>

Notes:

1. The execution time is six clocks when the PC22_EN compile option is selected, and four clocks otherwise.
IJMP

Jump (Indirect, using Z)

**Operation:** \( \text{PC} \leftarrow \{6'\text{h}00, Z\} \)

**Flags:** No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IJMP</td>
<td>10010100_00011001</td>
<td>4</td>
</tr>
</tbody>
</table>
IN
Load from I/O

IN Rd, A
Rd: R0-R31
A: 0-63 (6 bits)

Operation: Rd <= @A

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN Rd, A</td>
<td>10110AAd_ddddAAAA</td>
<td>2</td>
</tr>
</tbody>
</table>
INC
Increment

INC Rd  
Rd: R0-R31

Operation:  
Rd <= Rd + 1

Flags:  
I: Unaffected.
T: Unaffected.
H: Unaffected.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Unaffected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC Rd</td>
<td>1001010d_ddd0011</td>
<td>2</td>
</tr>
</tbody>
</table>
JMP
Jump (Long)

JMP k

k: 16 bits or 22 bits

Operation: PC <= k

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP k</td>
<td>1001010k_kkkk110k</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>kkkkkkk_kkkkkkkk</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

1. The timing is independent of the width of the PC.
LAC
Load and Clear

LAC Rd
Rd: R0-R31

Operation:
- tmp <= @Z
- @Z <= Rd & ~tmp
- Rd <= tmp

Flags:
No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAC Rd</td>
<td>1001001d_ddd0110</td>
<td>6</td>
</tr>
</tbody>
</table>

Notes:
1. This instruction is useful when dealing with certain memory-mapped I/O devices, and is only implemented when the RMW_EN option is selected.
**LAS**

**Load and Set**

**LAS Rd**

Rd: R0-R31

**Operation:**

\[

tmp \leftarrow @Z \\
@Z \leftarrow Rd \mid tmp \\
Rd \leftarrow tmp
\]

**Flags:**

No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAS Rd</td>
<td>1001001d_dddd0101</td>
<td>6</td>
</tr>
</tbody>
</table>

**Notes:**

1. This instruction is useful when dealing with certain memory-mapped I/O devices, and is only implemented when the RMW_EN option is selected.
LAT
Load and Toggle

LAT Rd

Rd: R0-R31

Operation:

tmp <= @Z

@Z <= Rd ^ tmp

Rd <= tmp

Flags:

No flags affected.

Assembly Syntax | Encoding | Clocks
---|---|---
LAT Rd | 1001001d_ddd0111 | 6

Notes:

1. This instruction is useful when dealing with certain memory-mapped I/O devices, and is only implemented when the RMW_EN option is selected.
**LD**

Load From Data Memory (Indirect, using X)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD Rd, X</td>
<td>1001000d(Employee)ddd1100</td>
<td>4</td>
</tr>
<tr>
<td>LD Rd, X+</td>
<td>1001000d(Employee)ddd1101</td>
<td>4</td>
</tr>
<tr>
<td>LD Rd, -X</td>
<td>1001000d(Employee)ddd1110</td>
<td>4</td>
</tr>
</tbody>
</table>

Notes:

1. X is R27:R26. Using either of these registers as the destination is not allowed.
**LD**

Load From Data Memory (Indirect, using Y)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD Rd, Y</td>
<td>LD Rd, Y</td>
<td>1000000d_dddd1000</td>
<td>4</td>
</tr>
<tr>
<td>LD Rd, Y+</td>
<td>LD Rd, Y+</td>
<td>1001000d_dddd1001</td>
<td>4</td>
</tr>
<tr>
<td>LD Rd, -Y</td>
<td>LD Rd, -Y</td>
<td>1001000d_dddd1010</td>
<td>4</td>
</tr>
<tr>
<td>LD Rd, Y+q</td>
<td>LD Rd, Y+q</td>
<td>10q0qq0d_dddd1qqq</td>
<td>4</td>
</tr>
</tbody>
</table>

Flags: No flags affected

Notes:
1. Y is R29:R28. Using either of these registers as the destination is not allowed.
# LD

**Load From Data Memory (Indirect, using Z)**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1</td>
<td>LD Rd, Z</td>
<td>1000000d_dddd0000</td>
<td>4</td>
</tr>
<tr>
<td>case 2</td>
<td>LD Rd, Z+</td>
<td>1001000d_dddd0001</td>
<td>4</td>
</tr>
<tr>
<td>case 3</td>
<td>LD Rd, -Z</td>
<td>1001000d_dddd0010</td>
<td>4</td>
</tr>
<tr>
<td>case 4</td>
<td>LD Rd, Z+q</td>
<td>10q0qq0d_dddd0qqq</td>
<td>4</td>
</tr>
</tbody>
</table>

**Flags:** No flags affected

**Notes:**

1. Z is R31:R30. Using either of these registers as the destination is not allowed.
# LDI

Load (Immediate)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDI Rd, K</td>
<td>1110KKK_kddKKK</td>
<td>2</td>
</tr>
</tbody>
</table>

**LDI Rd, K**

- **Rd**: R16-R31
- **K**: 0-255 (8 bits)

**Operation:**

\( \text{Rd} \leftarrow \text{K} \)

**Flags:**

No flags affected
LDS
Load From Data Space (Direct)

LDS Rd, k
Rd: R0-R31
k: 16 bits

LDS Rd, k
Rd: R16-R31
k: 0-127 (7 bits)

Operation: Rd <= @{RAMPD, k}

Flags: No flags affected

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1 LDS Rd, k</td>
<td>1001000d_dddd0000</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>kkkkkkkkk_kkkkkkkk</td>
<td></td>
</tr>
</tbody>
</table>
LPM
Load From Program Memory (Indirect, using Z)

LPM
LPM Rd, Z   Rd: R0-R31  case 2
LPM Rd, Z+  Rd: R0-R31  case 3

Operation:
R0 <= @{RAMPZ, Z}  case 1
Rd <= @{RAMPZ, Z}  case 2
Rd <= @{RAMPZ, Z}  case 3
{RAMPZ, Z} <= {RAMPZ, Z} + 1

Flags: No flags affected

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1</td>
<td>LPM</td>
<td>10010101_11001000</td>
</tr>
<tr>
<td>case 2</td>
<td>LPM Rd, Z</td>
<td>1001000d_dddd0100</td>
</tr>
<tr>
<td>case 3</td>
<td>LPM Rd, Z+</td>
<td>1001000d_dddd0101</td>
</tr>
</tbody>
</table>

Notes:

1. Z is R31:R30. Using either of these registers as the destination is not allowed.

2. Z holds a byte address. An even byte address accessed the lower byte in a word of program memory, while an odd byte address accesses the upper byte in a program memory word.
LSR

Logical Shift Right

**Operation:** \( \{\text{Rd, C}\} \leq \{1'b0, \text{Rd}\}\)

**Flags:**
- **I:** Unaffected.
- **T:** Unaffected.
- **H:** Unaffected.
- **S:** Identical to C flag.
- **V:** Identical to C flag.
- **N:** Cleared.
- **Z:** Set if the result is zero; cleared otherwise.
- **C:** Loaded with the contents of Rd[0] before the shift.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSR Rd</td>
<td>1001010d_dddd0110</td>
<td>2</td>
</tr>
</tbody>
</table>
MOV Rd, Rr
Rd: R0-R31
Rr: R0-R31

Operation: Rd <= Rr

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV Rd, Rr</td>
<td>001011rd_dddrrrr</td>
<td>2</td>
</tr>
</tbody>
</table>
MOVW
Move Word

MOVW Rd+1:Rd, Rr+1:Rr
Rd: even
Rr: even

Operation:  \{Rd+1, Rd\} <= \{Rr+1, Rr\}

Flags:  No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVW Rd+1:Rd, Rs+1:Rs</td>
<td>00000001_ddddrrrr</td>
<td>2</td>
</tr>
</tbody>
</table>
MUL Rd, Rr

Rd: R0-R31
Rr: R0-R31

Operation:

\[ \{R1, R0\} \leftarrow Rd \times Rr \quad \text{(unsigned} \leq \text{unsigned} \times \text{unsigned)} \]

Flags:
- I: Unaffected.
- T: Unaffected.
- H: Unaffected.
- S: Unaffected.
- V: Unaffected.
- N: Unaffected.
- Z: Set if the result is zero; cleared otherwise.
- C: Set if bit 15 of the result is set; cleared otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL Rd, Rr</td>
<td>100111rd_ddddrrrr</td>
<td>4</td>
</tr>
</tbody>
</table>
MULS
Multiply Signed

MULS Rd, Rr    Rd: R16-R31
Rr: R16-R31

Operation:    [R1, R0] <= Rd * Rr   (signed <= signed * signed)

Flags:    I: Unaffected.
T: Unaffected.
H: Unaffected.
S: Unaffected.
V: Unaffected.
N: Unaffected.
Z: Set if the result is zero; cleared otherwise.
C: Set if bit 15 of the result is set; cleared otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULS Rd, Rr</td>
<td>00000010_ddдрррр</td>
<td>4</td>
</tr>
</tbody>
</table>
MULSU
Multiply Signed with Unsigned

MULSU Rd, Rr
Rd: R16-R23
Rr: R16-R23

Operation:  \[ \{R1, R0\} \leftarrow Rd \times Rr \quad \text{(signed} \leq \text{signed} \times \text{unsigned)} \]

Flags:
- I: Unaffected.
- T: Unaffected.
- H: Unaffected.
- S: Unaffected.
- V: Unaffected.
- N: Unaffected.
- Z: Set if the result is zero; cleared otherwise.
- C: Set if bit 15 of the result is set; cleared otherwise.

Assembly Syntax  Encoding  Clocks

MULSU Rd, Rr  00000011_0ddd0rrr  4
NEG

Negate

NEG Rd

Rd: R0-R31

Operation:

Rd <= 8'h00 - Rd

Flags:

I: Unaffected.
T: Unaffected.
H: Set if arithmetic borrow out of bit 3; cleared otherwise.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic borrow out of bit 7; cleared otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG Rd</td>
<td>1001010d_dddd0001</td>
<td>2</td>
</tr>
</tbody>
</table>
NOP

No Operation

Operation: None

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>00000000_00000000</td>
<td>2</td>
</tr>
</tbody>
</table>
OR
Logical OR

**OR Rd, Rr**

**Rd:** R0-R31

**Rr:** R0-R31

**Operation:**  \( \text{Rd} \leftarrow \text{Rd} \lor \text{Rr} \)

**Flags:**
- **I:** Unaffected.
- **T:** Unaffected.
- **H:** Unaffected.
- **S:** \( \text{N} \land \text{V} \)
- **V:** Cleared.
- **N:** Set if bit 7 of the result is set; cleared otherwise.
- **Z:** Set if the result is zero; cleared otherwise.
- **C:** Unaffected.

**Assembly Syntax Encoding Clocks**

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR Rd, Rr</td>
<td>001010rd_dddrrrr</td>
<td>2</td>
</tr>
</tbody>
</table>
**ORI**

Logical OR (Immediate)

**ORI Rd, K**

Rd: R16-R31  
K: 0-255 (8 bits)

**Operation:**  
Rd \( \Leftarrow \) Rd | K

**Flags:**
- **I:** Unaffected.
- **T:** Unaffected.
- **H:** Unaffected.
- **S:** N \(^\wedge\) V
- **V:** Cleared.
- **N:** Set if bit 7 of the result is set; cleared otherwise.
- **Z:** Set if the result is zero; cleared otherwise.
- **C:** Unaffected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORI Rd, K</td>
<td>0110KKKK_ddddKKK</td>
<td>2</td>
</tr>
</tbody>
</table>
**OUT**

*Store to I/O*

**OUT A, Rr**

Rr: R0-R31

A: 0-63 (6 bits)

**Operation:**  
@A <= Rr

**Flags:**  
No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT A, Rr</td>
<td>10111AAAd_ddddAAAA</td>
<td>4</td>
</tr>
</tbody>
</table>

**Notes:**

1. Interrupts are not sampled by this instruction, so an interrupt that occurs simultaneously with this instruction will not be accepted until after the next instruction completes.

2. Writing to the SPL register using the OUT instruction disables the sampling of interrupts for four subsequent instructions, or until the next OUT instruction, whichever occurs first.
POP

Pop From Stack

**POP Rd**

**Rd:** R0-R31

**Operation:**

- SP <= SP + 1
- Rd <= @SP

**Flags:** No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>POP Rd</td>
<td>1001000d_dddd1111</td>
<td>2</td>
</tr>
</tbody>
</table>
PUSH
Push To Stack

PUSH Rr

Rr: R0-R31

Operation:
@SP <= Rr
SP <= SP - 1

Flags:
No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH Rr</td>
<td>1001001r_rrrr1111</td>
<td>4</td>
</tr>
</tbody>
</table>
RCALL

Call Subroutine (Relative)

**RCALL**

k: -2048 to 2047 (12 bits)

**Operation:**

\[
\begin{align*}
@SP & \leftarrow PC[7:0] \\
SP & \leftarrow SP - 1 \\
@SP & \leftarrow PC[15:8] \\
SP & \leftarrow SP - 1 \\
\text{if (PC22_EN)} & \text{ begin} \\
@SP & \leftarrow PC[21:16] \\
SP & \leftarrow SP - 1 \\
\text{end} \\
PC & \leftarrow PC + k + 1
\end{align*}
\]

**Flags:**

No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCALL</td>
<td>1101kkkk_kkkkkkkkk</td>
<td>4 (6)</td>
</tr>
</tbody>
</table>

**Notes:**

1. The execution time is six clocks when the PC22_EN compile option is selected, and four clocks otherwise.
RET

Return From Subroutine

RET

Operation:

\[
\begin{align*}
&\text{SP} \leftarrow \text{SP} + 1 \\
&\text{if (PC22\_EN)} \begin{align*}
&\text{PC}[21:16] \leftarrow @\text{SP} \\
&\text{SP} \leftarrow \text{SP} + 1 \\
&\end{align*} \\
&\text{PC}[15:8] \leftarrow @\text{SP} \\
&\text{SP} \leftarrow \text{SP} + 1 \\
&\text{PC}[7:0] \leftarrow @\text{SP}
\end{align*}
\]

Flags:

No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RET</td>
<td>10010101_01001000</td>
<td>8 or 10</td>
</tr>
</tbody>
</table>

Notes:

1. The execution time is 10 clocks when the PC22\_EN compile option is selected, and 8 clocks otherwise.
### RETI

**Return From Interrupt**

**Operation:**

```plaintext
SP <= SP + 1
if (PC22_EN) begin
    PC[21:16] <= @SP
    SP <= SP + 1
end
PC[15:8] <= @SP
SP <= SP + 1
PC[7:0] <= @SP
```

**Flags:**

- **I:** Set.
- **T:** Unaffected.
- **H:** Unaffected.
- **S:** Unaffected.
- **V:** Unaffected.
- **N:** Unaffected.
- **Z:** Unaffected.
- **C:** Unaffected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETI</td>
<td>10010101_00001000</td>
<td>8 or 10</td>
</tr>
</tbody>
</table>

**Notes:**

1. The execution time is 10 clocks when the PC22_EN compile option is selected, and 8 clocks otherwise.

2. Interrupts are not sampled by this instruction, so an interrupt that occurs simultaneously with this instruction will not be accepted until after the subsequent instruction completes.
RJPMP
Jump (Relative)

RJPMP k

k: -2048 to +2047 (12 bits)

Operation:

PC <= PC + k + 1

Flags:

No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJMP k</td>
<td>1100kkkk_kkkkkkkk</td>
<td>4</td>
</tr>
</tbody>
</table>
ROR
Rotate Right

ROR Rd
Rd: R0-R31

Operation:  \[ \{ \text{Rd, C} \} \leftarrow \{ \text{C, Rd} \} \]

Flags:
- I: Unaffected.
- T: Unaffected.
- H: Unaffected.
- S: Identical to C flag.
- V: Identical to C flag.
- N: Cleared.
- Z: Set if the result is zero; cleared otherwise.
- C: Loaded with the contents of Rd[0] before the shift.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROR Rd</td>
<td>1001010d_ddd0110</td>
<td>2</td>
</tr>
</tbody>
</table>
SBC
Subtract With Carry

SBC Rd, Rr
Rd: R0-R31
Rr: R0-R31

Operation:
Rd <= Rd - Rr - C

Flags:
I: Unaffected.
T: Unaffected.
H: Set if arithmetic borrow out of bit 3; cleared otherwise.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic borrow out of bit 7; cleared otherwise.

Assembly Syntax Encoding Clocks

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBC Rd, Rr</td>
<td>000010rd_ddrr</td>
<td>2</td>
</tr>
</tbody>
</table>
SBCI
Subtract With Carry (Immediate)

SBCI Rd, K
Rd: R16-R31
K: 0-255 (8 bits)

Operation: Rd <= Rd - K - C

Flags:
I: Unaffected.
T: Unaffected.
H: Set if arithmetic borrow out of bit 3; cleared otherwise.
S: N ^ V
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 7 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic borrow out of bit 7; cleared otherwise.

Assembly Syntax | Encoding | Clocks
--- | --- | ---
SBCI Rd, K | 0100KKK_dddKKK | 2
SBI
Set I/O Bit

SBI A, b
A: 0-31 (5 bits)
b: 0-7

Operation: @A[b] <= 1

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBI A, b</td>
<td>10011010_AAAAAbbb</td>
<td>4</td>
</tr>
</tbody>
</table>
SBIC A, b
A: 0-31 (5 bits)
b: 0-7

Operation: if (@A[b] == 0) then skip next instruction

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIC A, b</td>
<td>10011001_AAAAAbbb</td>
<td>2 (4/6)</td>
</tr>
</tbody>
</table>

Notes:

1. This instruction executes in two clocks if the next instruction is not skipped, in four clocks if the next instruction is skipped (one word instruction), or six clocks if the next instruction is skipped (two word instruction).

2. Interrupts are not sampled by this instruction, so an interrupt that occurs simultaneously with this instruction will not be accepted until after this instruction (or the subsequent non-skipped instruction) completes.
SBIS
Skip If I/O Bit Set

SBIS A, b
A: 0-31 (5 bits)
b: 0-7

Operation: if (@A[b] == 1) then skip next instruction

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBIS A, b</td>
<td>10011011_AAAAAbbb</td>
<td>2 (4/6)</td>
</tr>
</tbody>
</table>

Notes:

1. This instruction executes in two clocks if the next instruction is not skipped, in four clocks if the next instruction is skipped (one word instruction), or six clocks if the next instruction is skipped (two word instruction).

2. Interrupts are not sampled by this instruction, so an interrupt that occurs simultaneously with this instruction will not be accepted until after this instruction (or the subsequent non-skipped instruction) completes.
SBIW
Subtract Word (Immediate)

SBIW Rd+1:Rd, K
Rd: 24, 26, 28, 30
K: 0-63 (6 bits)

Operation:
\( \{Rd+1, Rd\} <= \{Rd+1, Rd\} - K \)

Flags:
- I: Unaffected.
- T: Unaffected.
- H: Unaffected.
- S: N ^ V
- V: Set if arithmetic overflow; cleared otherwise.
- N: Set if bit 15 of the result is set; cleared otherwise.
- Z: Set if the result is zero; cleared otherwise.
- C: Set if arithmetic borrow out of bit 15; cleared otherwise.

Assembly Syntax | Encoding | Clocks
--- | --- | ---
SBIW Rd+1:Rd, K | 10010111_KKddKKKK | 2
SBRC
Skip If Register Bit Clear

SBRC Rr, b  
Rr: R0-R31
b: 0-7

Operation:  if (Rr[b] == 0) then skip next instruction

Flags:  No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBRC A, b</td>
<td>1111110d_dddd0bb</td>
<td>2 (4/6)</td>
</tr>
</tbody>
</table>

Notes:

1. This instruction executes in two clocks if the next instruction is not skipped, in four clocks if the next instruction is skipped (one word instruction), or six clocks if the next instruction is skipped (two word instruction).

2. Interrupts are not sampled by this instruction, so an interrupt that occurs simultaneously with this instruction will not be accepted until after this instruction (or the subsequent non-skipped instruction) completes.
**SBRS**

Skip If Register Bit Set

**SBRS** Rr, b

- **Rr**: R0-R31
- **b**: 0-7

**Operation:**

if (Rr[b] == 1) then skip next instruction

**Flags:**

No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBRS A, b</td>
<td>1111111d_ddd0bbb</td>
<td>2 (4/6)</td>
</tr>
</tbody>
</table>

**Notes:**

1. This instruction executes in two clocks if the next instruction is not skipped, in four clocks if the next instruction is skipped (one word instruction), or six clocks if the next instruction is skipped (two word instruction).

2. Interrupts are not sampled by this instruction, so an interrupt that occurs simultaneously with this instruction will not be accepted until after this instruction (or the subsequent non-skipped instruction) completes.
SLEEP

Sleep

Operation:     sleep_pls <= 1

Flags:        No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLEEP</td>
<td>10010101_10001000</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes:

1. The sleep_pls signal is activated for one clock cycle.
**SPM**

*Store To Program Memory (Indirect, using Z)*

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SPM</td>
<td>case 1</td>
<td></td>
</tr>
<tr>
<td>SPM Z+</td>
<td>case 2</td>
<td></td>
</tr>
</tbody>
</table>

**Operation:**

@({RAMPZ, Z}) <= [R1, R0] 

@({RAMPZ, Z}) <= [R1, R0]  

Z <= Z + 2

**Flags:** No flags affected

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1 SPM</td>
<td>![10010101_11101000]</td>
<td>6</td>
</tr>
<tr>
<td>case 2 SPM Z+</td>
<td>![10010101_11111000]</td>
<td>6</td>
</tr>
</tbody>
</table>

**Notes:**

1. Z holds a byte address, so it must be even to properly access the word in program memory.
**ST**

*Store To Data Memory (Indirect, using X)*

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Rr: R0-R31</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST X, Rr</td>
<td>Rr: R0-R31</td>
<td>case 1</td>
</tr>
<tr>
<td>ST X+, Rr</td>
<td>Rr: R0-R31</td>
<td>case 2</td>
</tr>
<tr>
<td>ST -X, Rr</td>
<td>Rr: R0-R31</td>
<td>case 3</td>
</tr>
</tbody>
</table>

**Operation:**

- case 1: @\{RAMPX, X\} \(\leq\) Rr
- case 2: @\{RAMPX, X\} \(\leq\) Rr
  
  if (PC22_EN) {RAMPX, X} \(\leq\) {RAMPX, X} + 1

  else X \(\leq\) X + 1

- case 3: @\{RAMPX, X\} \(\leq\) Rr
  
  if (PC22_EN) {RAMPX, X} \(\leq\) {RAMPX, X} - 1

  else X \(\leq\) X - 1

**Flags:**

*No flags affected.*

<table>
<thead>
<tr>
<th>Case</th>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ST X, Rr</td>
<td>1001001d_dddd1100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>ST X+, Rr</td>
<td>1001001d_dddd1101</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>ST -X, Rr</td>
<td>1001001d_dddd1110</td>
<td>4</td>
</tr>
</tbody>
</table>

**Notes:**

1. X is R27:R26. Using either of these registers as the destination is not allowed.
# Store To Data Memory (Indirect, using Y)

| ST Y, Rr | Rr: R0-R31 | case 1 |
| ST Y+, Rr | Rr: R0-R31 | case 2 |
| ST -Y, Rr | Rr: R0-R31 | case 3 |
| ST Y+q, Rr | Rr: R0-R31 | case 4 |

**Operation:**

1. ST Y, Rr
   \[
   @\{\text{RAMPY}, Y\} \leftarrow Rr
   \]

2. ST Y+, Rr
   \[
   @\{\text{RAMPY}, Y\} \leftarrow Rr
   \]
   
   if (PC22\_EN) \{\text{RAMPY}, Y\} \leftarrow \{\text{RAMPY}, Y\} + 1
   
   else \ Y \leftarrow Y + 1

3. ST -Y, Rr
   \[
   @\{\text{RAMPY}, Y\} \leftarrow Rr
   \]
   
   if (PC22\_EN) \{\text{RAMPY}, Y\} \leftarrow \{\text{RAMPY}, Y\} - 1
   
   else \ Y \leftarrow Y - 1
   
   @\{\text{RAMPY}, Y\} \leftarrow Rr

4. ST Y+q, Rr
   \[
   @\{\{\text{RAMPY}, Y\} + q\} \leftarrow Rr
   \]

**Flags:** No flags affected

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1 ST Y, Rr</td>
<td>1000001d_dddd1000</td>
<td>4</td>
</tr>
<tr>
<td>case 2 ST Y+, Rr</td>
<td>1001001d_dddd1001</td>
<td>4</td>
</tr>
<tr>
<td>case 3 ST -Y, Rr</td>
<td>1001001d_dddd1010</td>
<td>4</td>
</tr>
<tr>
<td>case 4 ST Y+q, Rr</td>
<td>10q0qq1d_dddd1qqq</td>
<td>4</td>
</tr>
</tbody>
</table>

**Notes:**

1. Y is R29:R28. Using either of these registers as the destination is not allowed.
ST
Store To Data Memory (Indirect, using Z)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Z, Rr</td>
<td>ST Z, Rr</td>
<td>1000001d_dddd0000</td>
<td>4</td>
</tr>
<tr>
<td>ST Z+, Rr</td>
<td>ST Z+, Rr</td>
<td>1001001d_dddd0001</td>
<td>4</td>
</tr>
<tr>
<td>ST -Z, Rr</td>
<td>ST -Z, R</td>
<td>1001001d_dddd0010</td>
<td>4</td>
</tr>
<tr>
<td>ST Z+q, Rr</td>
<td>ST Z+q, Rr</td>
<td>10q0qq1d_dddd0qqq</td>
<td>4</td>
</tr>
</tbody>
</table>

Flags: No flags affected

Notes:
1. Z is R31:R30. Using either of these registers as the destination is not allowed.
## Store To Data Space (Direct)

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>STS k, Rr</td>
<td><img src="image" alt="Encoded Binary" /></td>
<td>4</td>
</tr>
</tbody>
</table>

**Operation:**`@{RAMPD, k} <= Rr`

**Flags:** No flags affected
**SUB**

**Subtract**

SUB Rd, Rr

Rd: R0-R31
Rr: R0-R31

**Operation:** Rd <= Rd - Rr

**Flags:**
- I: Unaffected.
- T: Unaffected.
- H: Set if arithmetic borrow out of bit 3; cleared otherwise.
- S: N ^ V
- V: Set if arithmetic overflow; cleared otherwise.
- N: Set if bit 7 of the result is set; cleared otherwise.
- Z: Set if the result is zero; cleared otherwise.
- C: Set if arithmetic borrow out of bit 7; cleared otherwise.

**Assembly Syntax | Encoding | Clocks**
--- | --- | ---
SUB Rd, Rr | 000110rd_dddrrrr | 2
SUBI
Subtract (Immediate)

SUBI Rd, K
Rd: R16-R31
K: 0-255 (8 bits)

Operation:
Rd <= Rd + K

Flags:
I: Unaffected.
T: Unaffected.
H: Unaffected.
S: \(N \oplus V\)
V: Set if arithmetic overflow; cleared otherwise.
N: Set if bit 15 of the result is set; cleared otherwise.
Z: Set if the result is zero; cleared otherwise.
C: Set if arithmetic carry out of bit 15; cleared otherwise.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBI Rd, K</td>
<td>0101KKKK_ddddKKKK</td>
<td>2</td>
</tr>
</tbody>
</table>
SWAP
Swap Nibbles

SWAP Rd
Rd: R0-R31

Operation:
Rd <= {Rd[3:0], Rd[7:4]}

Flags:
No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWAP Rd</td>
<td>1001010d_dddd0010</td>
<td>2</td>
</tr>
</tbody>
</table>
WDR

Operation:

Flags: No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDR</td>
<td>10010101_10101000</td>
<td>2</td>
</tr>
</tbody>
</table>

Notes:

1. The `wdr_pls` signal is activated for one clock cycle.
XCH
Exchange With Data Space

XCH Rr
Rr: R0-R31

Operation:

\[
\begin{align*}
\text{tmp} & \leftarrow @Z \\
@Z & \leftarrow Rr \\
Rr & \leftarrow \text{tmp}
\end{align*}
\]

Flags:
No flags affected.

<table>
<thead>
<tr>
<th>Assembly Syntax</th>
<th>Encoding</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCH Rd</td>
<td>1001001d_dddd0100</td>
<td>6</td>
</tr>
</tbody>
</table>

Notes:

1. This instruction is useful when dealing with certain memory-mapped I/O devices, and is only implemented when the RMW_EN option is selected.
The table below lists all of the internal Special Function Registers used in the design, along with their mnemonics, address and reset state. SFR addresses not listed here are considered external, and will be accessed via the External SFR bus.

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Mnemonic</th>
<th>SFR Address</th>
<th>Data Memory Address</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Interrupt Mask</td>
<td>EIMSK</td>
<td>0x1D</td>
<td>0x003D</td>
<td>00000000</td>
</tr>
<tr>
<td>General-Purpose I/O 0</td>
<td>GPIOR0</td>
<td>0x1E</td>
<td>0x003E</td>
<td>00000000</td>
</tr>
<tr>
<td>General-Purpose I/O 1</td>
<td>GPIOR1</td>
<td>0x2A</td>
<td>0x004A</td>
<td>00000000</td>
</tr>
<tr>
<td>General-Purpose I/O 2</td>
<td>GPIOR2</td>
<td>0x2B</td>
<td>0x004B</td>
<td>00000000</td>
</tr>
<tr>
<td>Data Page</td>
<td>RAMPD</td>
<td>0x38</td>
<td>0x0058</td>
<td>00000000</td>
</tr>
<tr>
<td>X Register Page</td>
<td>RAMPX</td>
<td>0x39</td>
<td>0x0059</td>
<td>00000000</td>
</tr>
<tr>
<td>Y Register Page</td>
<td>RAMPY</td>
<td>0x3A</td>
<td>0x005A</td>
<td>00000000</td>
</tr>
<tr>
<td>Z Register Page</td>
<td>RAMPZ</td>
<td>0x3B</td>
<td>0x005B</td>
<td>00000000</td>
</tr>
<tr>
<td>Extended Indirect Page</td>
<td>EIND</td>
<td>0x3C</td>
<td>0x005C</td>
<td>00000000</td>
</tr>
<tr>
<td>Stack Pointer LSB</td>
<td>SPL</td>
<td>0x3D</td>
<td>0x005D</td>
<td>00000000</td>
</tr>
<tr>
<td>Stack Pointer MSB</td>
<td>SPH</td>
<td>0x3E</td>
<td>0x005E</td>
<td>00000000</td>
</tr>
<tr>
<td>Status Register</td>
<td>SREG</td>
<td>0x3F</td>
<td>0x005F</td>
<td>00000000</td>
</tr>
</tbody>
</table>
## Register Descriptions

### External Interrupt Mask Register (EIMSK) (Address = 0x1D)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>Disable External Interrupt 7.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 7.</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>Disable External Interrupt 6.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 6.</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>Disable External Interrupt 5.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 5.</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>Disable External Interrupt 4.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 4.</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Disable External Interrupt 3.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 3.</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Disable External Interrupt 2.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 2.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Disable External Interrupt 1.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 1.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Disable External Interrupt 0.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Enable External Interrupt 0.</td>
</tr>
</tbody>
</table>

### General-Purpose I/O 0 (GPIOR0) (Address = 0x1E)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>General-purpose read/write I/O register.</td>
</tr>
</tbody>
</table>

### General-Purpose I/O 1 (GPIOR1) (Address = 0x2A)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>General-purpose read/write I/O register.</td>
</tr>
</tbody>
</table>

### General-Purpose I/O 2 (GPIOR2) (Address = 0x2B)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>General-purpose read/write I/O register.</td>
</tr>
</tbody>
</table>
### Data Page (RAMPD)  (Address = 0x38)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>This byte holds the page address when using direct addressing.</td>
</tr>
</tbody>
</table>

### X Register Page (RAMPX)  (Address = 0x39)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>This byte holds the page address when using the X register for indirect addressing.</td>
</tr>
</tbody>
</table>

### Y Register Page (RAMPY)  (Address = 0x3A)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>This byte holds the page address when using the Y register for indirect addressing.</td>
</tr>
</tbody>
</table>

### Z Register Page (RAMPZ)  (Address = 0x3B)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>This byte holds the page address when using the Z register for indirect addressing.</td>
</tr>
</tbody>
</table>

### Extended Indirect Page (EIND)  (Address = 0x3C)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>This byte holds the destination page address for Program Memory when using the EICALL or EIJMP instructions.</td>
</tr>
</tbody>
</table>

### Stack Pointer LSB (SPL)  (Address = 0x3D)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>This byte holds the least-significant byte of the Stack Pointer. When updating the Stack Pointer, this byte should always be written first. A write to this byte disables interrupts for the next four instructions or until a write to SPH, whichever occurs first. This disabling operation only works when using the OUT instruction.</td>
</tr>
</tbody>
</table>
### Stack Pointer MSB (SPH) (Address = 0x3E)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td></td>
<td>This byte holds the most-significant byte of the Stack Pointer.</td>
</tr>
</tbody>
</table>

### Status Register (SREG) (Address = 0x3F)

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td>Global Interrupt Enable (I) bit.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Link (T) status bit.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Half-carry (H) status bit.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Sign (S) status bit. Equal to N ^ V.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Overflow (V) status bit.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Negative (N) status bit.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Zero (Z) status bit.</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>Carry (C) status bit.</td>
</tr>
</tbody>
</table>