YVR Microprocessor

Technical Manual

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

Revison History

Date	Changes	Page(s)
03/10/2014	Preliminary issue	

Introduction

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

Interface Description

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 acknowedge transaction for the internal interrupt. This interrupt vector is loaded into the least-significant byte pf 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.

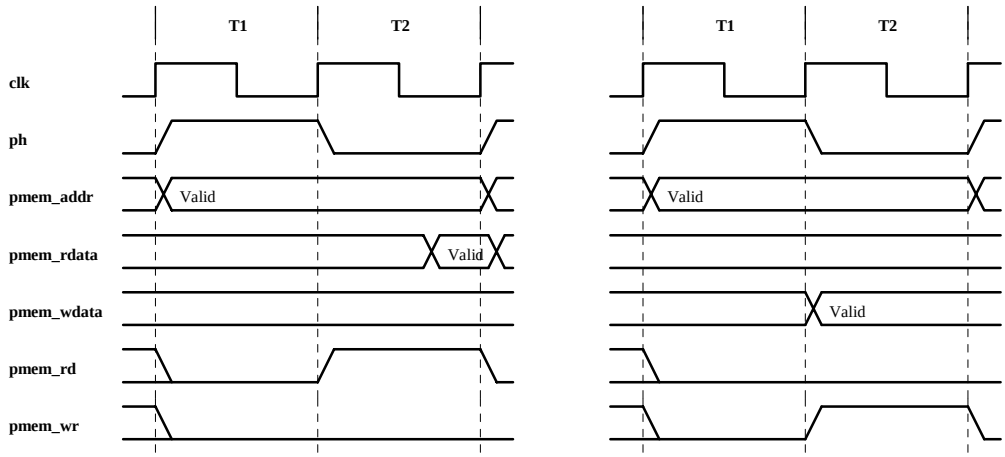
The program and data memory interface timing and signals are designed to make it easy to interface to standard ASIC and FPGA memories. These interfaces use separate read and write strobes.

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.

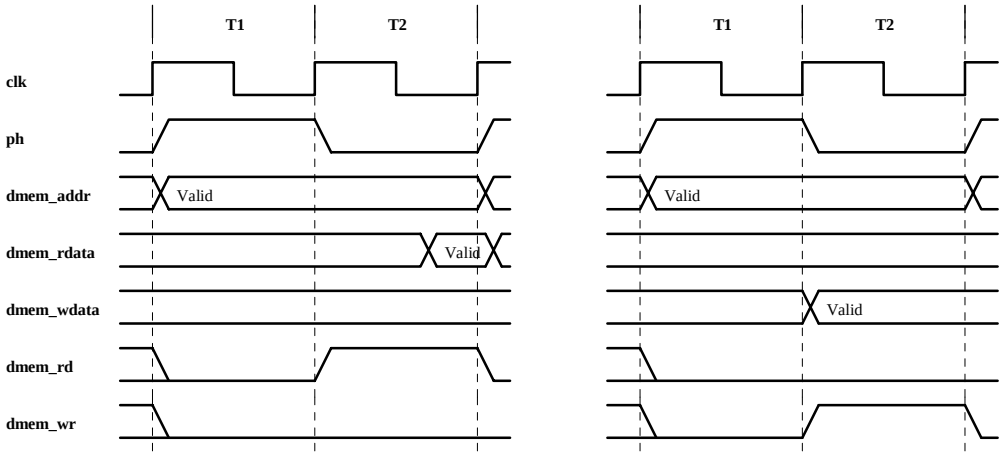
Program Memory Read and Write

The figure below shows Program Memory read and write transactions.



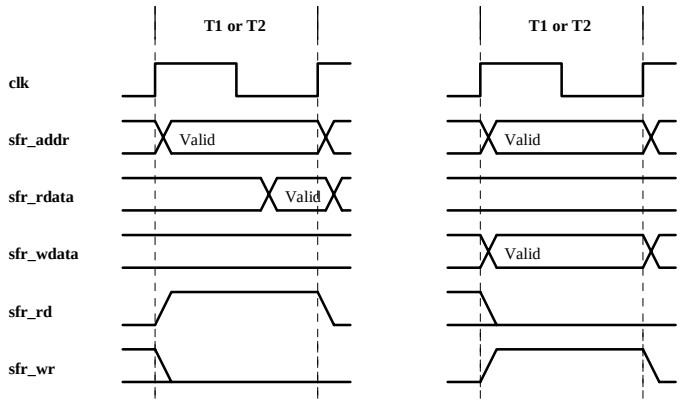
Data Memory Read and Write

The figures below show read and write transactions for Data Memory. Data Memory transactions are always aligned with Program Memory transactions, so the **ph** signal can also be used to determine the phase for Data Memory transactions.



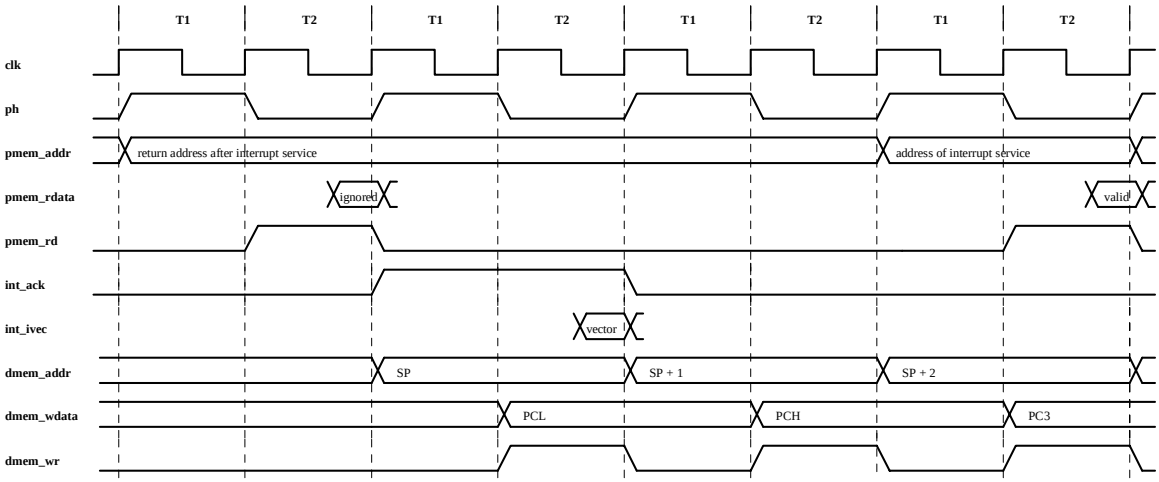
SFR Read and Write

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.



Interrupt Acknowledge

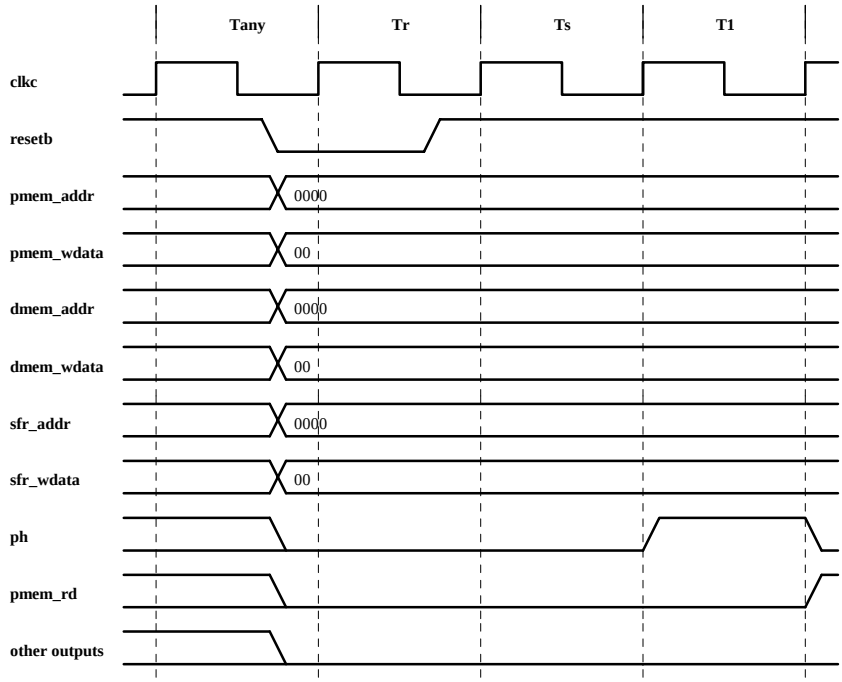
The figure below shows the interrupt acknowledge transaction, including the preceding aborted instruction fetch and subsequent instruction fetch for the service routine. The **dmem_addr** and **dmem_wdata** buses are also shown to indicate where the internal RAM is written. Timing is the same in the case of a 16-bit Program Counter, but only two bytes are pushed to the stack.



Reset

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.



Instruction Set

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:

I	T	H	S	V	N	Z	C

These flags have the following meanings:

Flag	Meaning
C	Carry
Z	Zero
N	Negative
V	Overflow
S	N ^ V
H	Half Carry
T	Transfer Bit
I	Global Interrupt Enable

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:

Rd, Rr (5-bit field)	**Rd (4-bit field)**	**Rd (4-bit field)**	**Rd (3-bit field)**	**Rd (2-bit field)**	**Register**	**Data Memory address**
00000		0000			R0	0x0000
00001					R1	0x0001
00010		0001			R2	0x0002
00011					R3	0x0003
00100		0010			R4	0x0004
00101					R5	0x0005
00110		0011			R6	0x0006
00111					R7	0x0007
01000		0100			R8	0x0008
01001					R9	0x0009
01010		0101			R10	0x000A
01011					R11	0x000B
01100		0110			R12	0x000C
01101					R13	0x000D
01110		0111			R14	0x000E
01111					R15	0x000F
10000	0000	1000	000		R16	0x0010
10001	0001		001		R17	0x0011
10010	0010	1001	010		R18	0x0012
10011	0011		011		R19	0x0013
10100	0100	1010	100		R20	0x0014
10101	0101		101		R21	0x0015
10110	0110	1011	110		R22	0x0016
10111	0111		111		R23	0x0017
11000	1000	1100		00	R24	0x0018
11001	1001				R25	0x0019
11010	1010	1101		01	R26	0x001A
11011	1011				R27	0x001B
11100	1100	1110		10	R28	0x001C
11101	1101				R29	0x001D
11110	1110	1111		11	R30	0x001E
11111	1111				R31	0x001F

Bits within a byte are selected by the usual encoding:

b, s	bit selected
000	bit 0
001	bit 1
010	bit 2
011	bit 3
100	bit 4
101	bit 5
110	bit 6
111	bit 7

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

Data Memory address	Use
0x0000 - 0x001F	R0 - R31
0x0020 - 0x005F	Special Function Registers 0 - 3F
> 0x0060	External Data Memory

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 ^ 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
	ADC Rd, Rr	`000111rd_ddddrrrr`	2

ADD

Add

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.

	Assembly Syntax	Encoding	Clocks
	ADD Rd, Rr	`000011rd_ddddrrrr`	2

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
	ADIW Rd+1:Rd, K	`10010110_KKddKKKK`	2

AND

Logical AND

AND 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
	AND Rd, Rr	`001000rd_ddddrrrr`	2

ANDI

Logical AND (Immediate)

ANDI Rd, K

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

Operation: Rd <= Rd & K

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
	ANDI Rd, K	`0111KKKK_ddddKKKK`	2

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.

Assembly Syntax	Encoding	Clocks
ASR Rd	1001010d_dddd0101	2

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.

	Assembly Syntax	Encoding	Clocks
	BCLR s	10010100_1sss1000	2

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: Rd[b] <= T

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
BLD Rd, b	`1111100d_dddd0bbb`	2

BRBC

Branch If Bit Clear (in SREG)

BRBC s, K

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

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

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	BRBC s, K	`111101kk_kkkkksss`	2 (4)

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

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	BRBS s, K	111100kk_kkkkksss	2 (4)

Notes:

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

BREAK

Break

BREAK

Operation: break_pls <= 1

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	BREAK	10010101_10011000	2

Notes:

1. The **break_pls** signal is activated for one clock cycle.

BSET

Set Bit (in SREG)

BSET s s: 0-7

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.

Assembly Syntax	Encoding	Clocks
BSET s	10010100_0sss1000	2

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 neabled) 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_dddd0bbb`	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.

Assembly Syntax	Encoding	Clocks
CALL k	1001010k_kkkk111k kkkkkkkk_kkkkkkkk	6

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.

	Assembly Syntax	Encoding	Clocks
	CBI A, b	`10011000_AAAAAbbb`	2

COM

Complement

COM 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.

Assembly Syntax	Encoding	Clocks
COM Rd	`1001010d_dddd0000`	2

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_ddddrrrr`	2

CPC

Compare With Carry

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.

Assembly Syntax	Encoding	Clocks
CPSE Rd, Rr	`000100rd_ddddrrrr`	2 (4/6)

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.

Assembly Syntax	Encoding	Clocks
DEC Rd	1001010d_dddd1010	2

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.

	Assembly Syntax	Encoding	Clocks
	EICALL	`10010101_00011001`	6

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)

EIJMP

Operation: PC <= {EIND, Z}

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	EIJMP	10010100_00011001	4

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.

ELPM

Load From Program Memory (Indirect, using RAMPZ and Z)

ELPM			case 1
ELPM Rd, Z		Rd: R0-R31	case 2
ELPM Rd, Z+		Rd: R0-R31	case 3
Operation:	R0 <= @{RAMPZ, Z}		case 1
	Rd <= @{RAMPZ, Z}		case 2
	Rd <= @{RAMPZ, Z} {RAMPZ, Z} <= {RAMPZ, Z} + 1		case 3

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_ddddrrrr`	2

FMUL

Fractional Multiply Unsigned

FMUL 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
	FMUL Rd, Rr	00000011_0ddd1rrr	4

Notes:

1. Rd and Rs 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} <= (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
FMULSU Rd, Rr	00000011_1ddd1rrr	4

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.

	Assembly Syntax	Encoding	Clocks
	ICALL	`10010101_00001001`	4 (6)

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)

IJMP

Operation: PC <= {6'h00, Z}

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
IJMP	10010100_00011001	4

IN

Load from I/O

IN Rd, A

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

Operation: Rd <= @A

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	IN Rd, A	`10110AAd_ddddAAAA`	2

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.

	Assembly Syntax	Encoding	Clocks
	INC Rd	1001010d_dddd0011	2

JMP

Jump (Long)

JMP k k: 16 bits or 22 bits

Operation: PC <= k

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	JMP k	1001010k_kkkk110k kkkkkkkk_kkkkkkkk	6

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.

Assembly Syntax	Encoding	Clocks
LAC Rd	`1001001d_dddd0110`	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.

LAS

Load and Set

LAS Rd — Rd: R0-R31

Operation:

```
tmp <= @Z
@Z <= Rd | tmp
Rd <= tmp
```

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
LAS Rd	`1001001d_dddd0101`	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.

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_dddd0111`	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)

LD Rd, X		Rd: R0-R31	case 1
LD Rd, X+		Rd: R0-R31	case 2
LD Rd, -X		Rd: R0-R31	case 3
Operation:	Rd <= @{RAMPX, X}		case 1
	Rd <= @{RAMPX, X} if (PC22_EN) {RAMPX, X} <= {RAMPX, X} + 1 else X <= X + 1		case 2
	if (PC22_EN) {RAMPX, X} <= {RAMPX, X} - 1 else X <= X - 1 Rd <= @{RAMPX, X}		case 3

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
case 1	LD Rd, X	1001000d_dddd1100	4
case 2	LD Rd, X+	1001000d_dddd1101	4
case 3	LD Rd, -X	1001000d_dddd1110	4

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)

LD Rd, Y		Rd: R0-R31	case 1
LD Rd, Y+		Rd: R0-R31	case 2
LD Rd, -Y		Rd: R0-R31	case 3
LD Rd, Y+q		Rd: R0-R31 q: 0-63 (6 bits)	case 4
Operation:	Rd <= @Y		case 1
	Rd <= @{RAMPY, Y} if (PC22_EN) {RAMPY, Y} <= {RAMPY, Y} + 1 else Y <= Y + 1		case 2
	if (PC22_EN) {RAMPY, Y} <= {RAMPY, Y} - 1 else Y <= Y - 1 Rd <= @{RAMPY, Y}		case 3
	Rd <= @({RAMPY, Y} + q)		case 4

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	LD Rd, Y	`1000000d_dddd1000`	4
case 2	LD Rd, Y+	`1001000d_dddd1001`	4
case 3	LD Rd, -Y	`1001000d_dddd1010`	4
case 4	LD Rd, Y+q	`10q0qq0d_dddd1qqq`	4

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)

LD Rd, Z	Rd: R0-R31	case 1
LD Rd, Z+	Rd: R0-R31	case 2
LD Rd, -Z	Rd: R0-R31	case 3
LD Rd, Z+q	Rd: R0-R31 q: 0-63 (6 bits)	case 4

Operation:

```
Rd <= @Z                                                    case 1

Rd <= @{RAMPZ, Z}                                           case 2
if (PC22_EN) {RAMPZ, Z} <= {RAMPZ, Z} + 1
else Z <= Z + 1

if (PC22_EN) {RAMPZ, Z} <= {RAMPZ, Z} - 1                   case 3
else Z <= Z - 1
Rd <= @{RAMPZ, Z}

Rd <= @({RAMPZ, Z} + q)                                     case 4
```

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	LD Rd, Z	`1000000d_dddd0000`	4
case 2	LD Rd, Z+	`1001000d_dddd0001`	4
case 3	LD Rd, -Z	`1001000d_dddd0010`	4
case 4	LD Rd, Z+q	`10q0qq0d_dddd0qqq`	4

Notes:

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

LDI

Load (Immediate)

LDI Rd, K

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

Operation: Rd <= K

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
	LDI Rd, K	`1110KKKK_ddddKKKK`	2

LDS

Load From Data Space (Direct)

LDS Rd, k	Rd: R0-R31 k: 16 bits	case 1
LDS Rd, k	Rd: R16-R31 k: 0-127 (7 bits)	case 2

Operation: Rd <= @{RAMPD, k}

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	LDS Rd, k	`1001000d_dddd0000` `kkkkkkkk_kkkkkkkk`	4

LPM

Load From Program Memory (Indirect, using Z)

LPM			case 1
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} {RAMPZ, Z} <= {RAMPZ, Z} + 1		case 3

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	LPM	`10010101_11001000`	6
case 2	LPM Rd, Z	`1001000d_dddd0100`	6
case 3	LPM Rd, Z+	`1001000d_dddd0101`	6

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

LSR Rd Rd: R0-R31

Operation: {Rd, C} <= {1'b0, 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.

	Assembly Syntax	Encoding	Clocks
	LSR Rd	1001010d_dddd0110	2

MOV

Move

MOV Rd, Rr

Rd: R0-R31
Rr: R0-R31

Operation: Rd <= Rr

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
MOV Rd, Rr	`001011rd_ddddrrrr`	2

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.

	Assembly Syntax	Encoding	Clocks
	MOVW Rd+1:Rd, Rs+1:Rs	`00000001_ddddrrrr`	2

MUL

Multiply Unsigned

MUL Rd, Rr

Rd: R0-R31
Rr: R0-R31

Operation: {R1, R0} <= Rd * Rr (unsigned <= unsigned * 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
MUL Rd, Rr	`100111rd_ddddrrrr`	4

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.

Assembly Syntax	Encoding	Clocks
MULS Rd, Rr	00000010_ddddrrrr	4

MULSU

Multiply Signed with Unsigned

MULSU Rd, Rr

Rd: R16-R23
Rr: R16-R23

Operation: {R1, R0} <= Rd * Rr (signed <= signed * 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.

	Assembly Syntax	Encoding	Clocks
	NEG Rd	1001010d_dddd0001	2

NOP

No Operation

NOP

Operation: None

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
NOP	`00000000_00000000`	2

OR

Logical OR

OR 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
OR Rd, Rr	`001010rd_ddddrrrr`	2

ORI

Logical OR (Immediate)

ORI Rd, K

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

Operation: Rd <= Rd | K

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
ORI Rd, K	`0110KKKK_ddddKKKK`	2

OUT

Store to I/O

OUT A, Rr

Rr: R0-R31
A: 0-63 (6 bits)

Operation: @A <= Rr

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	OUT A, Rr	10111AAd_ddddAAAA	4

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.

Assembly Syntax	Encoding	Clocks
POP Rd	`1001000d_dddd1111`	2

PUSH

Push To Stack

PUSH Rr Rr: R0-R31

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

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	PUSH Rr	`1001001r_rrrr1111`	4

RCALL

Call Subroutine (Relative)

RCALL k: -2048 to +2047 (12 bits)

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 <= PC + k + 1
```

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	RCALL	`1101kkkk_kkkkkkkk`	4 (6)

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:

```
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: No flags affected.

	Assembly Syntax	Encoding	Clocks
	RET	`10010101_01001000`	8 or 10

Notes:

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

RETI

Return From Interrupt

RETI

Operation:

```
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.

Assembly Syntax	Encoding	Clocks
RETI	10010101_00001000	8 or 10

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.

RJMP

Jump (Relative)

RJMP k k: -2048 to +2047 (12 bits)

Operation: PC <= PC + k + 1

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	RJMP k	`1100kkkk_kkkkkkkk`	4

ROR

Rotate Right

ROR Rd Rd: R0-R31

Operation: {Rd, C} <= {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.

	Assembly Syntax	Encoding	Clocks
	ROR Rd	1001010d_dddd0110	2

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
SBC Rd, Rr	`000010rd_ddddrrrr`	2

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	`0100KKKK_ddddKKKK`	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.

	Assembly Syntax	Encoding	Clocks
	SBI A, b	`10011010_AAAAAbbb`	4

SBIC

Skip If I/O Bit Clear

SBIC A, b

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

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

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
SBIC A, b	10011001_AAAAAbbb	2 (4/6)

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.

	Assembly Syntax	Encoding	Clocks
	SBIS A, b	`10011011_AAAAAbbb`	2 (4/6)

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.

	Assembly Syntax	Encoding	Clocks
	SBRC A, b	`1111110d_dddd0bbb`	2 (4/6)

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.

	Assembly Syntax	Encoding	Clocks
	SBRS A, b	`1111111d_dddd0bbb`	2 (4/6)

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

SLEEP

Operation: sleep_pls <= 1

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	SLEEP	10010101_10001000	2

Notes:

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

SPM

Store To Program Memory (Indirect, using Z)

SPM		case 1
SPM Z+		case 2
Operation:	@{RAMPZ, Z} <= {R1, R0}	case 1
	@{RAMPZ, Z} <= {R1, R0} Z <= Z + 2	case 2

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	SPM	10010101_11101000	6
case 2	SPM Z+	10010101_11111000	6

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)

ST X, Rr	Rr: R0-R31	case 1
ST X+, Rr	Rr: R0-R31	case 2
ST -X, Rr	Rr: R0-R31	case 3

Operation:

case 1:
```
@{RAMPX, X} <= Rr
```

case 2:
```
@{RAMPX, X} <= Rr
if (PC22_EN) {RAMPX, X} <= {RAMPX, X} + 1
else X <= X + 1
```

case 3:
```
if (PC22_EN) {RAMPX, X} <= {RAMPX, X} - 1
else X <= X - 1
@{RAMPX, X} <= Rr
```

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
case 1	ST X, Rr	`1001001d_dddd1100`	4
case 2	ST X+, Rr	`1001001d_dddd1101`	4
case 3	ST -X, Rr	`1001001d_dddd1110`	4

Notes:

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

ST

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 q: 0-63 (6 bits)	case 4
Operation:	@{RAMPY, Y} <= Rr		case 1
	@{RAMPY, Y} <= Rr if (PC22_EN) {RAMPY, Y} <= {RAMPY, Y} + 1 else Y <= Y + 1		case 2
	if (PC22_EN) {RAMPY, Y} <= {RAMPY, Y} - 1 else Y <= Y - 1 @{RAMPY, Y} <= Rr		case 3
	@({RAMPY, Y} + q) <= Rr		case 4

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	ST Y, Rr	`1000001d_dddd1000`	4
case 2	ST Y+, Rr	`1001001d_dddd1001`	4
case 3	ST -Y, Rr	`1001001d_dddd1010`	4
case 4	ST Y+q, Rr	`10q0qq1d_dddd1qqq`	4

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)

ST Z, Rr	Rr: R0-R31	case 1
ST Z+, Rr	Rr: R0-R31	case 2
ST -Z, Rr	Rr: R0-R31	case 3
ST Z+q, Rr	Rr: R0-R31 q: 0-63 (6 bits)	case 4

Operation:	@{RAMPZ, Z} <= Rr	case 1
	@{RAMPZ, Z} <= Rr if (PC22_EN) {RAMPZ, Z} <= {RAMPZ, Z} + 1 else Z <= Z + 1	case 2
	if (PC22_EN) {RAMPZ, Z} <= {RAMPZ, Z} - 1 else Z <= Z - 1 @{RAMPZ, Z} <= Rr	case 3
	@({RAMPZ, Z} + q) <= Rr	case 4

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	ST Z, Rr	`1000001d_dddd0000`	4
case 2	ST Z+, Rr	`1001001d_dddd0001`	4
case 3	ST -Z, R	`1001001d_dddd0010`	4
case 4	ST Z+q, Rr	`10q0qq1d_dddd0qqq`	4

Notes:

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

STS

Store To Data Space (Direct)

STS k, Rr		Rr: R0-R31 k: 16 bits	case 1
STS k, Rr		Rr: R16-R31 k: 0-127 (7 bits)	case 2
Operation:	@{RAMPD, k} <= Rr		

Flags: **N**o flags affected

	Assembly Syntax	Encoding	Clocks
case 1	STS k, Rr	1001001d_dddd0000 kkkkkkkk_kkkkkkkk	4

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_ddddrrrr`	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 ^ 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
	SUBI Rd, K	`0101KKKK_ddddKKKK`	2

SWAP

Swap Nibbles

SWAP Rd Rd: R0-R31

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

Flags: No flags affected.

	Assembly Syntax	Encoding	Clocks
	SWAP Rd	`1001010d_dddd0010`	2

WDR

WDT Reset

WDR

Operation:

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
WDR	10010101_10101000	2

Notes:

1. The **wdr_pls** signal is activated for one clock cycle.

XCH

Exchange With Data Space

XCH Rr — Rr: R0-R31

Operation:

tmp <= @Z
@Z <= Rr
Rr <= tmp

Flags: No flags affected.

Assembly Syntax	Encoding	Clocks
XCH Rd	1001001d_dddd0100	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.

Special Function Registers

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.

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

Register Descriptions

External Interrupt Mask Register (EIMSK (Address = 0x1D)		
Bit(s)	**Value**	**Description**
7	0	Disable External Interrupt 7.
	1	Enable External Interrupt 7.
6	0	Disable External Interrupt 6.
	1	Enable External Interrupt 6.
5	0	Disable External Interrupt 6.
	1	Enable External Interrupt 5.
4	0	Disable External Interrupt 4.
	1	Enable External Interrupt 4.
3	0	Disable External Interrupt 3.
	1	Enable External Interrupt 3.
2	0	Disable External Interrupt 2.
	1	Enable External Interrupt 2.
1	0	Disable External Interrupt 1.
	1	Enable External Interrupt 1.
0	0	Disable External Interrupt 0.
	1	Enable External Interrupt 0.

General-Purpose I/O 0 (GPIOR0) (Address = 0x1E)		
Bit(s)	**Value**	**Description**
7:0		General-purpose read/write I/O register.

General-Purpose I/O 1 (GPIOR1) (Address = 0x2A)		
Bit(s)	**Value**	**Description**
7:0		General-purpose read/write I/O register.

General-Purpose I/O 2 (GPIOR2) (Address = 0x2B)		
Bit(s)	**Value**	**Description**
7:0		General-purpose read/write I/O register.

Data Page		(RAMPD)	(Address = 0x38)
Bit(s)	**Value**	**Description**	
7:0		This byte holds the page address when using direct addressing.	

X Register Page		(RAMPX)	(Address = 0x39)
Bit(s)	**Value**	**Description**	
7:0		This byte holds the page address when using the X register for indirect addressing.	

Y Register Page		(RAMPY)	(Address = 0x3A)
Bit(s)	**Value**	**Description**	
7:0		This byte holds the page address when using the Y register for indirect addressing.	

Z Register Page		(RAMPZ)	(Address = 0x3B)
Bit(s)	**Value**	**Description**	
7:0		This byte holds the page address when using the Z register for indirect addressing.	

Extended Indirect Page		(EIND)	(Address = 0x3C)
Bit(s)	**Value**	**Description**	
7:0		This byte holds the destination page address for Program Memory when using the EICALL or EIJMP instructions.	

Stack Pointer LSB		(SPL)	(Address = 0x3D)
Bit(s)	**Value**	**Description**	
7:0		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.	

Stack Pointer MSB		(SPH) (Address = 0x3E)
Bit(s)	**Value**	**Description**
7:0		This byte holds the most-significant byte of the Stack Pointer.

Status Register		(SREG (Address = 0x3F)
Bit(s)	**Value**	**Description**
7		Global Interrupt Enable (I) bit.
6		Link (T) status bit.
5		Half-carry (H) status bit.
4		Sign (S) status bit. Equal to N ^ V.
3		Overflow (V) status bit.
2		Negative (N) status bit.
1		Zero (Z) status bit.
0		Carry (C) status bit.