Embedded RISC Microcontroller Core

AVR®

Features

- Utilizes the *AVR*® Enhanced RISC Architecture
  - High Performance and Low Power
  - Sleep Mode to Conserve Power
- 120 Powerful Instructions - Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- Operating Range: 1.6 to 3.6 Volts
- Fully Static Operation, 0-33 MHz (0.5 micron), 0-45 MHz (0.35 micron)
- Seven External Interrupt Sources
- AVR Scalable Test Access Interface
- Test Vectors for >99% Fault Coverage
- Verilog and VHDL Simulation Models
- Faster Version can be Created Upon Request

Description

The *AVR*® Embedded RISC Microcontroller Core is a low-power CMOS 8-bit microprocessor based on the *AVR* enhanced RISC architecture. By executing powerful instructions in a single clock cycle, it achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The *AVR* Core is based on an enhanced RISC architecture that combines a rich instruction set with the 32 general purpose working registers. Each of the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

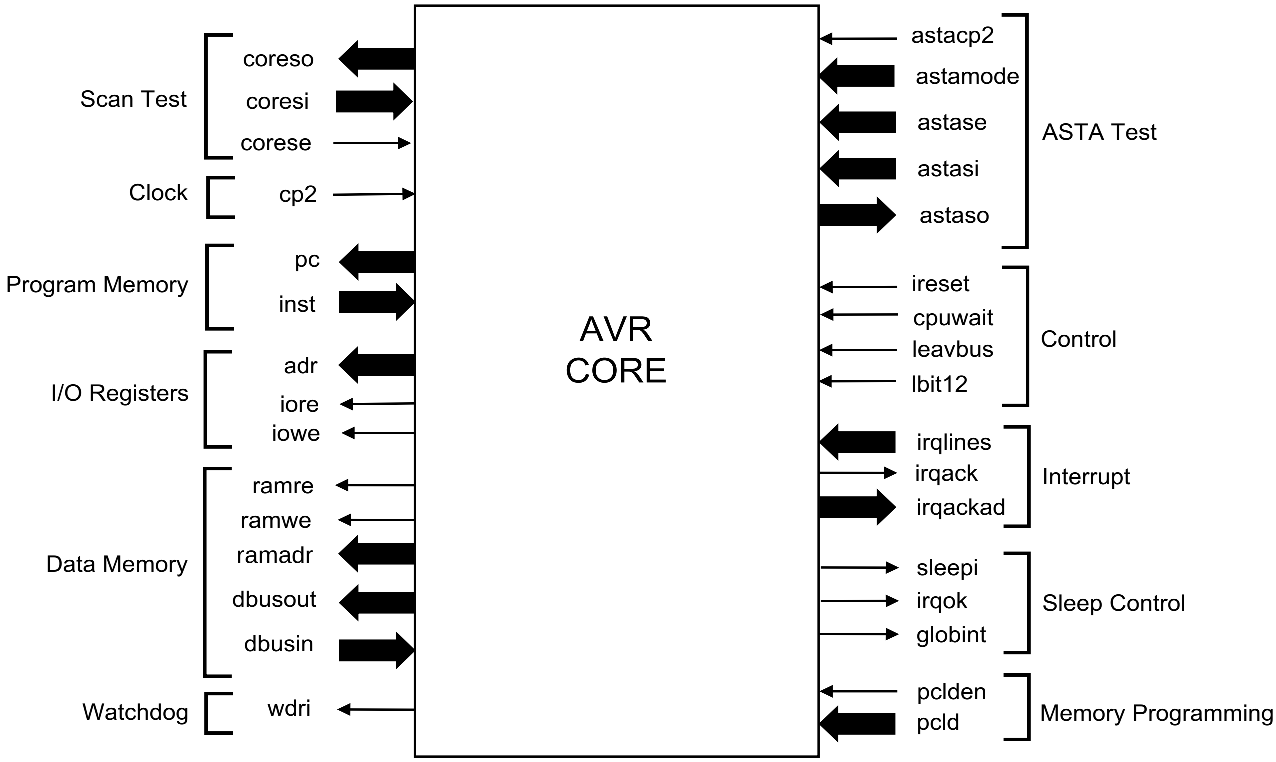
The architecture supports high level languages efficiently as well as extremely dense assembler code programs. It also provides any number of external and internal interrupts.

The *AVR* Core is provided in an encrypted netlist format with Verilog and VHDL simulation models, a fully functional test bench and ATPG vectors for >99% fault coverage. It is supported with a full suite of program and system development tools including: macro assemblers, ANSI C Compilers, program debugger/simulators, and in-circuit emulator.

Rev. 0890A–02/99

I/O Configuration

Figure 1. *AVR* Core I/O Configuration



I/O Description

Table 1. I/O Description

Name	Input/ Output	Function
Clock Port		
cp2 Clock	Input	Any register in the core will update its contents only on the positive edge of **cp2**.
Control Ports		
ireset *AVR* Core Reset	Input	When high, **ireset** causes the core to reset the program counter **pc**, the status register SREG, and the stack pointer, loading all with zeros ($0000). When **ireset** is high and **leavbus** is inactive, zero ($00) is driven on the Data Bus **dbusout**, and the I/O Write Strobe **iowe** is held high while the I/O Read Strobe and the Data Memory Strobes (**ramre**, **ramwe**) are held low. This allows I/O registers to be reset by reading zero from the Bus.
cpuwait Wait CPU	Input	This signal is used to add wait cycles to allow slow memory accesses. When **cpuwait** is high, the core repeats the current cycle (only for instructions addressing the RAM space such as 'ld' or 'st'). When **cpuwait** is released, the cycle is executed as normal. For details, refer to the timing diagrams below.
leavbus leave dbusout	Input	This signal is used to control **dbusout** externally. When high, **dbusin** is connected directly to **dbusout**, and all I/O and Data Memory Strobes are held low.
lbit12 Logical and between Lock bit 1 and 2	Input	Disables 'lpm' and 'elpm' instructions.
Program Memory Ports		
pc [15:0] Program Counter	Output	Program Memory always returns the instruction stored at the address pointed to. The size of this port determines the program memory size.
inst [15:0] Program Memory data bus	Input	Instruction from Program Memory is presented to the core, selected by the address on **pc** address bus.
I/O Registers		
adr [5:0] I/O Register address bus	Output	Valid only when accompanied by a strobe on **iore** or **iowe** lines.
iore I/O Registers read strobe	Output	Used only with the 64 I/O memory locations. These locations can be mapped into the regular Data Memory Address Space. The core will then issue an **iore** or **ramre** read strobe based on target address.
iowe I/O Registers write strobe	Output	Used only with the 64 I/O memory locations. These locations can be mapped into the regular Data Memory Address Space. The core will then issue an **iowe** or **ramwe** read strobe based on target address.
Data Memory Ports		
ramadr [15:0] Data Memory address bus	Output	Valid only when accompanied by a strobe on **ramre** or **ramwe** lines.
ramre Data Memory read strobe	Output	Used to address the SRAM memory locations. The core will issue an **iore** or **ramre** read strobe based on target address.

Table 1. I/O Description (Continued)

Name	Input/ Output	Function
ramwe Data Memory write strobe	Output	Used to address the SRAM memory locations. The core will issue an **iore** or **ramre** read strobe based on target address.
dbusin [7:0] Data Bus Input	Input	All data transfers use **dbusin** or **dbusout** to transfer data into or out of the core. Memory locations are selected by the address on **ramadr** (Data Memory Address). I/O Register locations are selected by the address on **adr**.
dbusout [7:0] Data Bus Output	Output	
		Interrupt Ports
irqlines [6:0] Interrupt Request Lines	Input	Each interrupt source drives its own dedicated IRQ line into the Core. When the global interrupt bit is enabled, a high level (one) on any interrupt line will push the current **pc** on the stack. The associated interrupt handler vector address is put in the Program Counter **pc** before execution is restarted.
irqack Interrupt Acknowledge	Output	**irqack** will go high (one) for one clock cycle to acknowledge the interrupt being executed. This is often used as input to interrupt flags designed to clear when their corresponding interrupt handler is executed. The **irqackad** lines identify which interrupt is being executed during the same cycle.
irqackad [2:0] Interrupt Acknowledge Address	Output	The address of the interrupt being executed. The address is valid only if the **irqack** signal is set (one).
		Sleep Controller Ports
sleepi Sleep instruction	Output	Set while executing the ‘sleep’ instruction. This should cause the sleep controller to stop the clock to the core if sleep mode has been enabled.
irqok Interrupt Request OK	Output	When in sleep mode (clock stopped), this signal will tell the sleep controller that an interrupt exists which should cause the clock to restart. The sleep controller should start the core clock as soon as possible.
globint Global interrupts enabled	Output	This is the current state of the I bit in the Core State Register. This signal is used to qualify wake-up from power-down by external interrupts.
		Memory Programming Ports
pclden enable **pc** load	Input	Enable **pc** load with **pcld** signals.
pcld [1:0] Load Program Counter	Input	Load Program Counter **pc** from **dbusin** if **pclden** is active. **pcld [1]** load high byte, **pcld [0]** load low byte.
		Watchdog Port
wdri Watchdog reset instruction	Output	Set while executing the ‘wdi’ (watchdog reset) instruction.
		Scan Test Ports
corese	Input	Core Test Scan Enable
coresi [2:0](1)	Input	Core Test Scan Inputs
coreso [2:0](1)	Output	Core Test Scan Outputs

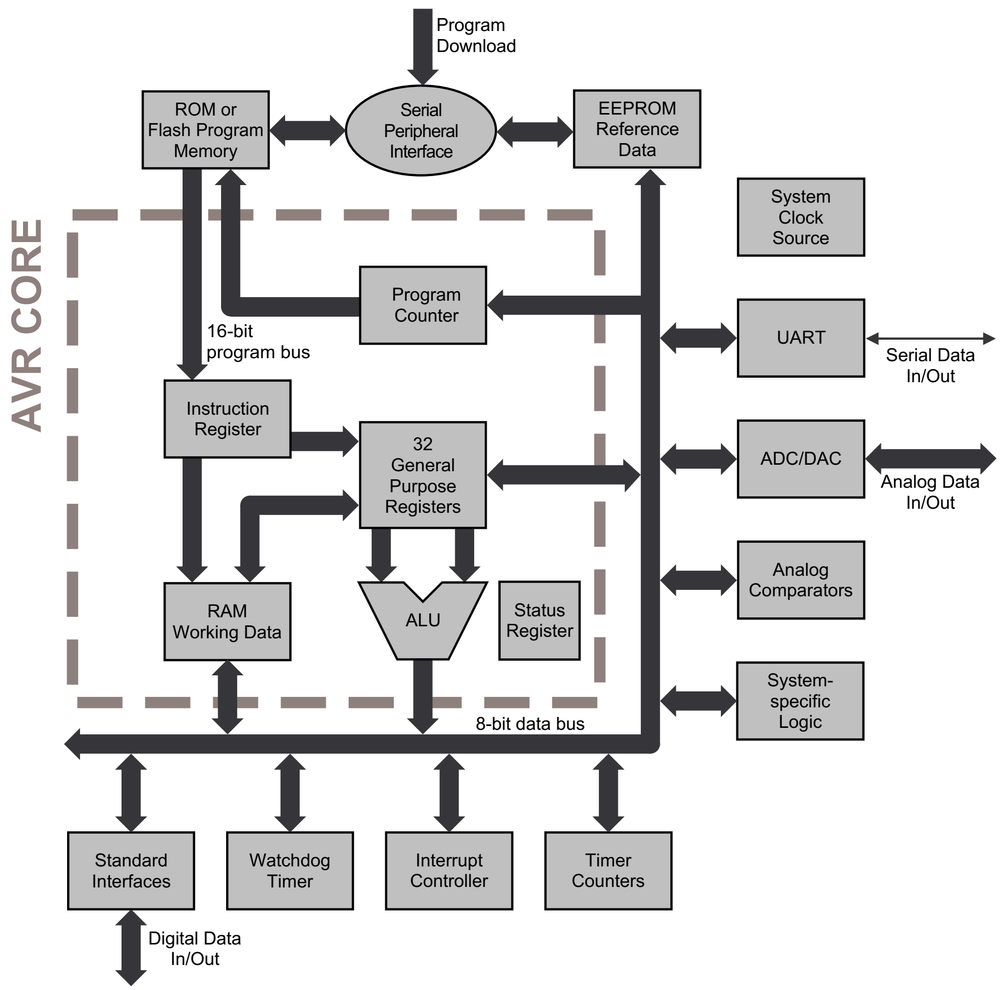
Table 1. I/O Description (Continued)

Name	Input/ Output	Function
		ASTA Test Ports
astacp2 ASTA clock	Input	Any register in the ASTA interface will update its contents only on the positive edge of **astacp2**.
astamode [1:0](1)	Input	ASTA mode inputs used to swap between ASTA mode or normal function mode.
astase [8:0](1)	Input	ASTA Test Scan Enables
astasi [8:0](1)	Input	ASTA Test Scan Inputs
astaso [8:0](1)	Output	ASTA Test Scan Outputs

Note: 1. Width is subject to change.

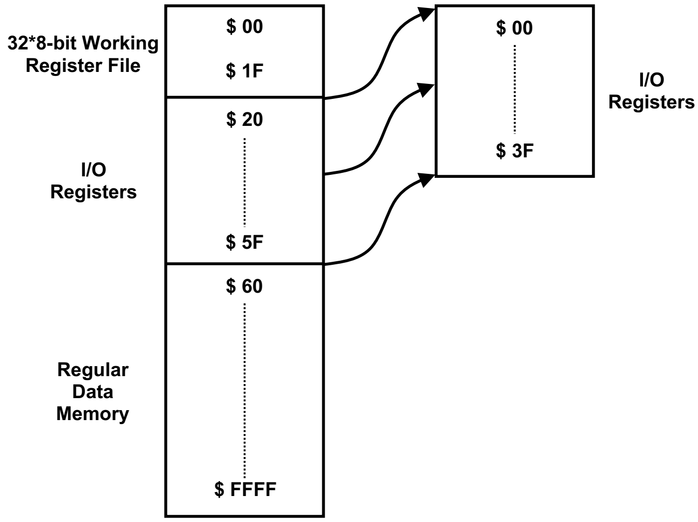
AVR Core Architecture

Figure 2. Block Diagram of the AVR Core and a Typical Set of Peripherals



The AVR core is based on a Harvard architecture with separate memories and buses for program and data (Figure 2). The memory spaces in the AVR architecture are all linear and regular memory maps.

Figure 3. AVR Data Memory Map



The central AVR architectural element is a fast-access register file containing 32 x 8-bit general purpose registers with a single clock cycle access time. This means that during one clock cycle, one ALU operation is executed. Two operands are accessed from the register file, the operation is executed, and the result is stored back in the register file - in one clock cycle. The ALU supports arithmetic and logic functions between registers or between a constant and a register, as well as single register operations.

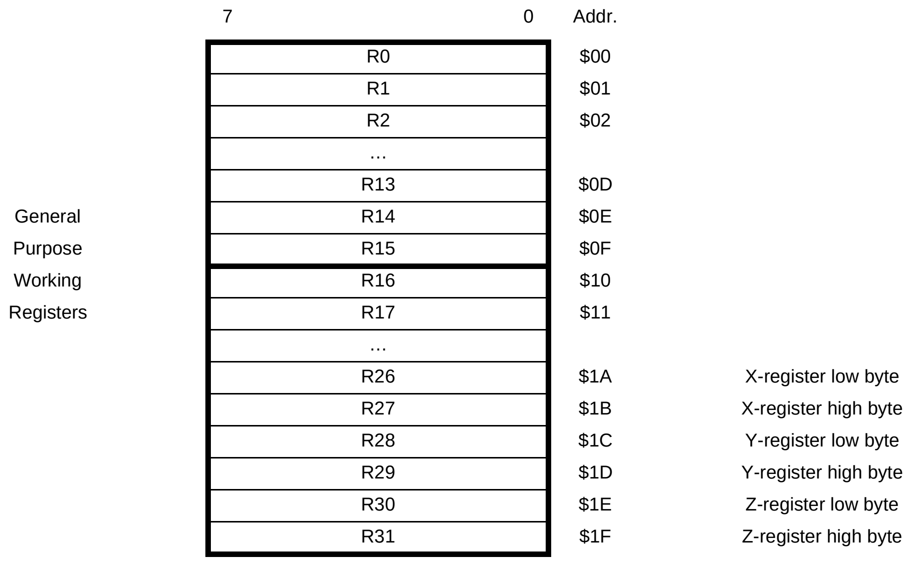
The program memory can be implemented in ROM or Flash memory. It is accessed with a single level of pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This enables instructions to be executed in every clock cycle. All AVR instructions have a single 16-bit word format, meaning that every program memory address contains a single instruction. During interrupts and subroutine calls, the return address is stored on a software stack.

The 8-bit data memory (Figure 3) has 16-bit direct addressing. This gives a potential memory space of 64K bytes. The data memory address space includes the register file, and a 64-address I/O memory space for peripheral functions such as control registers, timer-counters and A/D converters. As shown in Figure 3, the I/O memory space is automatically re-mapped for access by the register file.

The General Purpose Register File

The figure below shows the structure of the 32 general purpose working registers in the CPU.

Figure 4. *AVR* CPU General Purpose Working Registers



All the register operating instructions in the instruction set have direct and single cycle access to all registers. The only exception is the five constant arithmetic and logic instructions SBCI, SUBI, CPI, ANDI and ORI between a constant and a register and the LDI instruction for load immediate constant data. These instructions apply to the second half of the registers in the register file - R16 to R31. The general SBC, SUB, CP, AND and OR and all other operations between two registers or on a single register apply to the entire register file.

As shown in Figure 4, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X,Y and Z registers can be set to index any register in the file.

THE X-REGISTER, Y-REGISTER AND Z-REGISTER

The registers R26 to R31 have some added functions to their general purpose usage. These registers are address pointers for indirect addressing of the Data Space. The three indirect address registers X, Y and Z are defined as:

Figure 5. The X, Y and Z Registers

In the different addressing modes these address registers have functions as fixed displacement, automatic increment and decrement (see the descriptions for the different instructions).

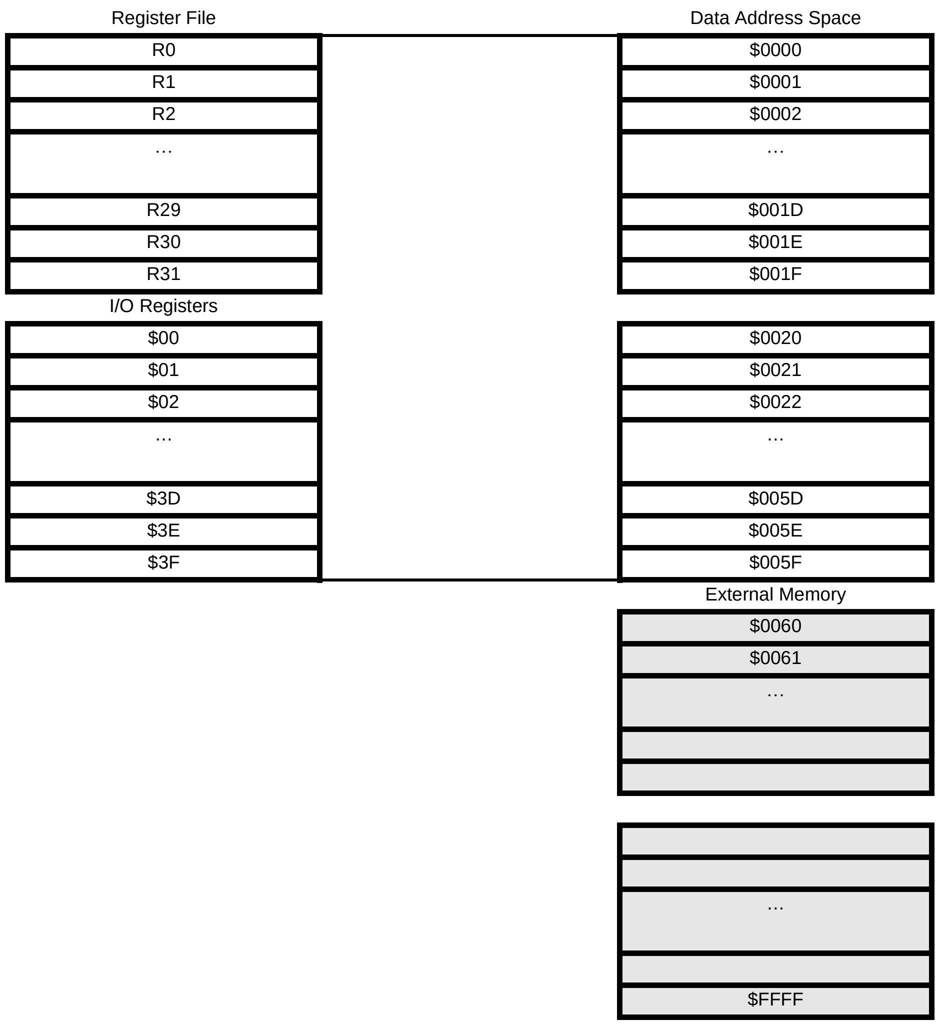
The ALU - Arithmetic Logic Unit

The high-performance *AVR* ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, ALU operations between registers in the register file are executed. The ALU operations are divided into three main categories - arithmetic, logical and bit-functions.

Data Memory Configuration

The following figure shows how the *AVR* Core Memory is organized:

Figure 6. SRAM Organization



The first 96 locations address the Register File + I/O Memory, and the next locations address the external data memory.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-Decrement and Indirect with Post-Increment. In the register file, registers R26 to R31 feature the indirect addressing pointer registers.

The direct addressing reaches the entire data space.

The Indirect with Displacement mode features a 63 address location reach from the base address given by the Y or Z-register.

When using register indirect addressing modes with automatic pre-decrement or post-increment, the address registers X, Y and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O registers and the 64K bytes of external data SRAM in the *AVR* Core are all accessible through all these addressing modes.

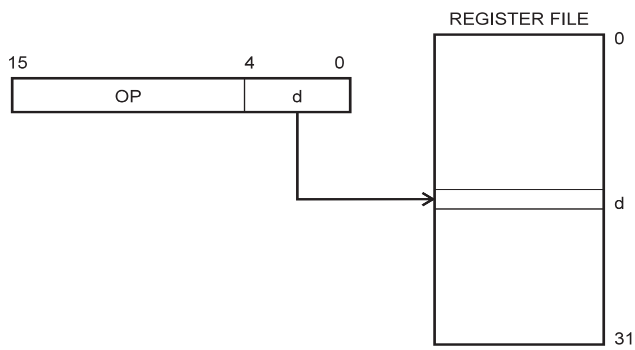
See the next section for a detailed description of the different addressing modes.

Program and Data Addressing Modes

The *AVR* Enhanced RISC microcontroller core supports powerful and efficient addressing modes for access to the program memory and data memory (SRAM, Register File and I/O Memory). This section describes the different addressing modes supported by the *AVR* architecture. In the figures, OP means the operation code part of the instruction word. To simplify, not all figures show the exact location of the addressing bits.

REGISTER DIRECT, SINGLE REGISTER RD

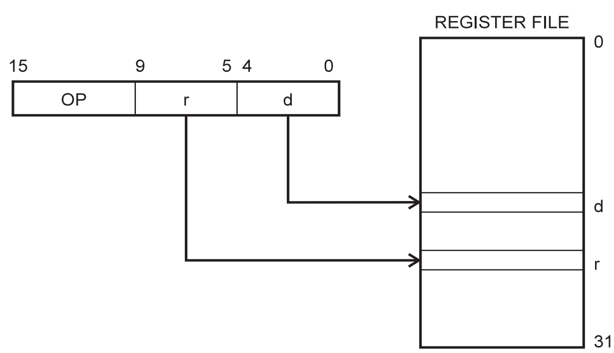
Figure 7. Direct Single Register Addressing



The operand is contained in register d (Rd).

REGISTER DIRECT, TWO REGISTERS RD AND RR

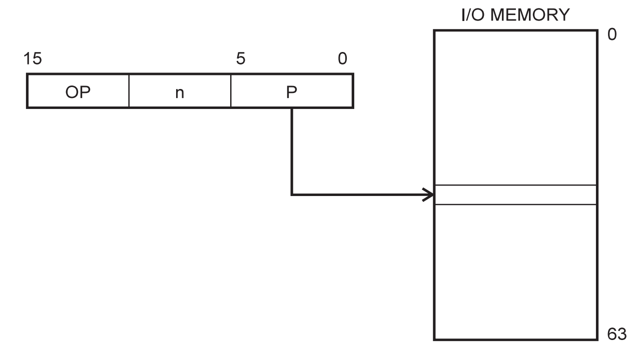
Figure 8. Direct Register Addressing, Two Registers



The operands are contained in registers r (Rr) and d (Rd). The result is stored in register d (Rd).

I/O DIRECT

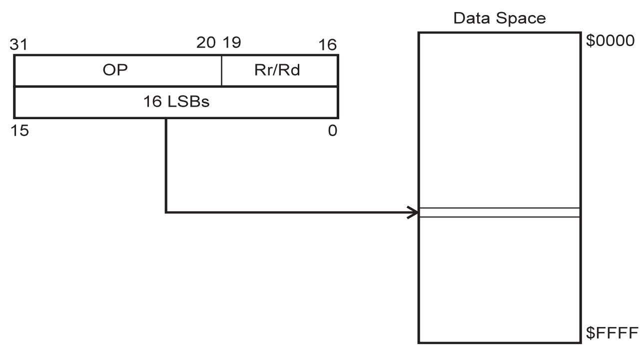
Figure 9. I/O Direct Addressing



The operand address is contained in 6 bits of the instruction word. n is the destination or source register address.

DATA DIRECT

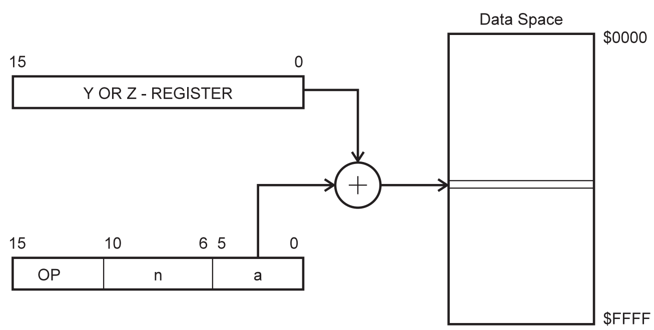
Figure 10. Direct Data Addressing



A 16-bit Data Address is contained in the 16 LSBs of a two-word instruction. Rd/Rr specify the destination or source register.

DATA INDIRECT WITH DISPLACEMENT

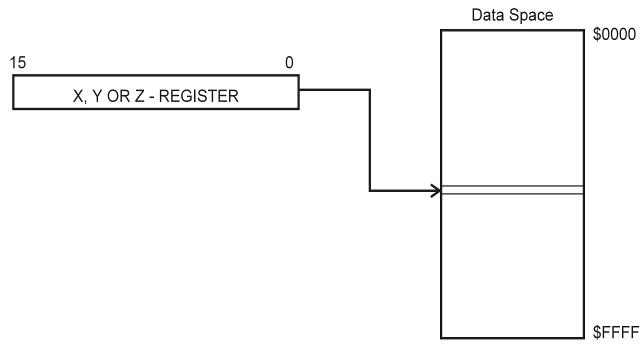
Figure 11. Data Indirect with Displacement



The operand address is the result of the Y or Z-register contents added to the displacement contained in 6 bits of the instruction word.

DATA INDIRECT

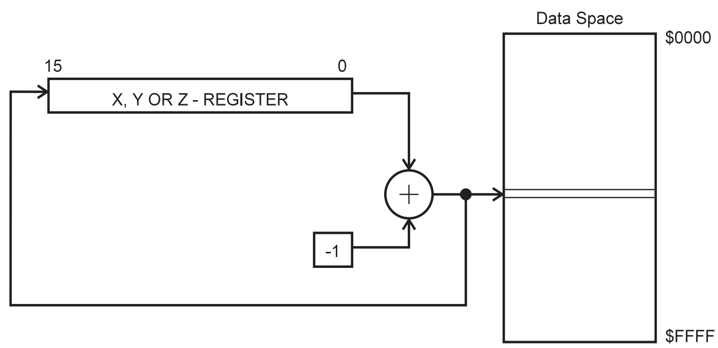
Figure 12. Data Indirect Addressing



The operand address is the contents of the X, Y or the Z-register.

DATA INDIRECT WITH PRE-DECREMENT

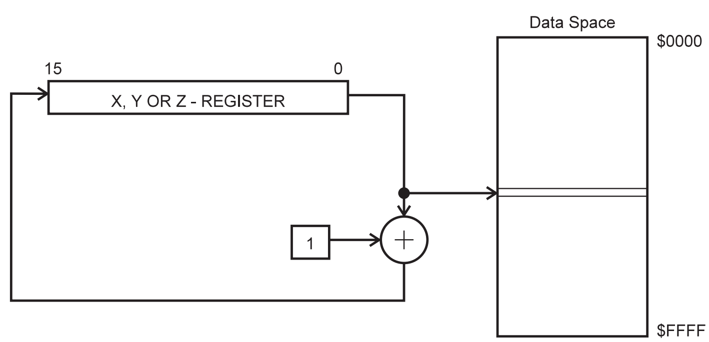
Figure 13. Data Indirect Addressing With Pre-Decrement



The X, Y or the Z-register is decremented before the operation. The operand address is the decremented contents of the X, Y or the Z-register.

DATA INDIRECT WITH POST-INCREMENT

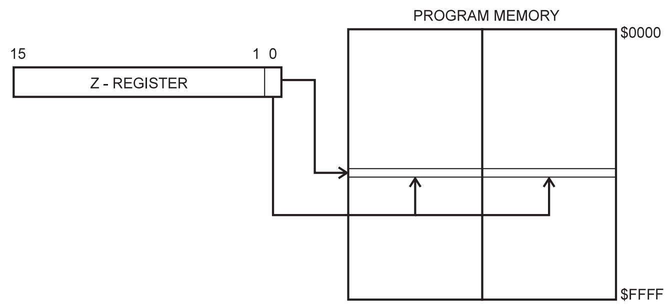
Figure 14. Data Indirect Addressing With Post-Increment



The X, Y or the Z-register is incremented after the operation. The operand address is the content of the X, Y or the Z-register prior to incrementing.

CONSTANT ADDRESSING USING THE LPM INSTRUCTION

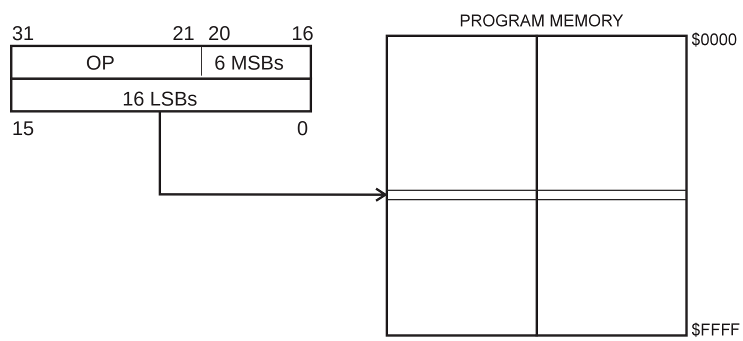
Figure 15. Code Memory Constant Addressing



Constant byte address is specified by the Z-register contents. The 15 MSBs select the word address (0 - 32K) and the LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). If ELPM is used, LSB of the RAM Page Z register - RAMPZ is used to select low or high memory page (RAMPZ0 = 0: Low Page, RAMPZ0 = 1: High Page).

DIRECT PROGRAM ADDRESS, JMP AND CALL

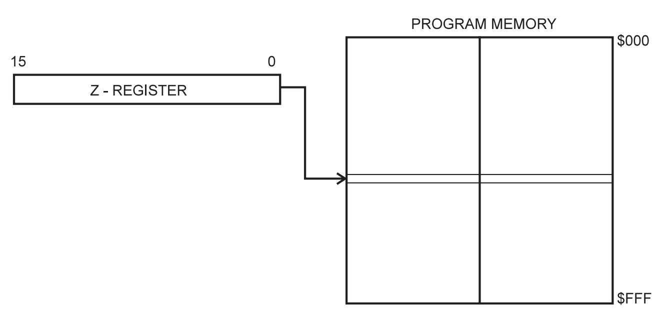
Figure 16. Direct Program Memory Addressing



Program execution continues at the address immediate in the instruction words.

INDIRECT PROGRAM ADDRESSING, IJMP AND ICALL

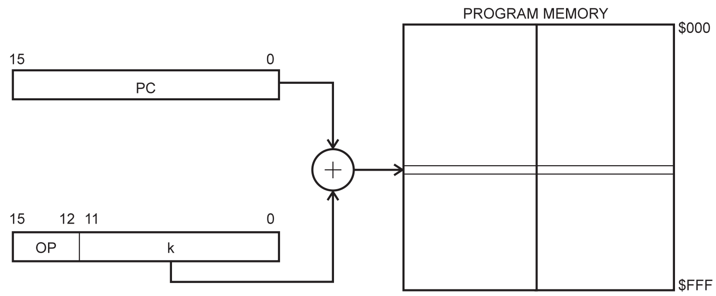
Figure 17. Indirect Program Memory Addressing



Program execution continues at address contained by the Z-register (i.e. the **pc** is loaded with the contents of the Z-register).

RELATIVE PROGRAM ADDRESSING, RJMP AND RCALL

Figure 18. Relative Program Memory Addressing



Program execution continues at address **pc** + k + 1. The relative address k is -2048 to 2047.

Data Memory Access

Data Memory is accessed in two clock cycles. During the first cycle of a write instruction, the data is driven onto **dbusout**. During the second cycle, the core issues an address on **ramadr** and the **ramwe** strobe. When **ramwe** is high and **ramadr** matches the address of an existing memory location, the memory should update its contents only if this occurs on the rising edge of **cp2**. As the new data is no longer valid on **dbusout**, the data must be latched outside the core.

During the second cycle of a read instruction, the core issues an address on **ramadr** and the **ramre** strobe. While ramre is high and **ramadr** matches the address of an existing memory location, the memory should drive its contents onto **dbusin**.

Data memory space from address $00 to $5F cannot be used for SRAM data space because it is used for the general purpse register file and the I/O registers (see Figure 3 on page 6).

Figure 19. Data Memory Access. Read is combinatorial, write is synchronous. During write, the data value disappears from **dbusin** in cycle 2 and needs to be latched for one cycle outside the core.

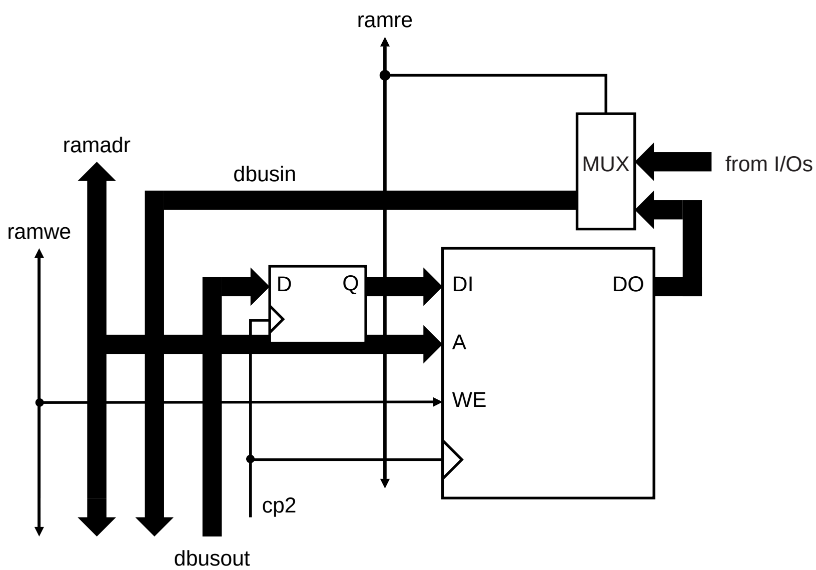


Figure 20. *AVR* SRAM Memory Read, using 'ld' or 'lds' instruction.

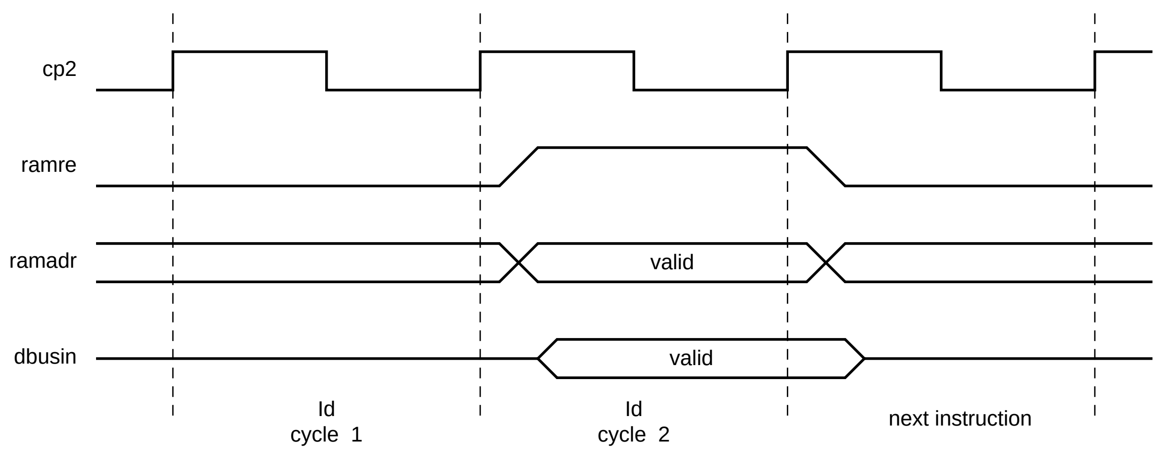


Figure 21. *AVR* SRAM Memory Read, using 'ld' or 'lds' instruction with one wait state.

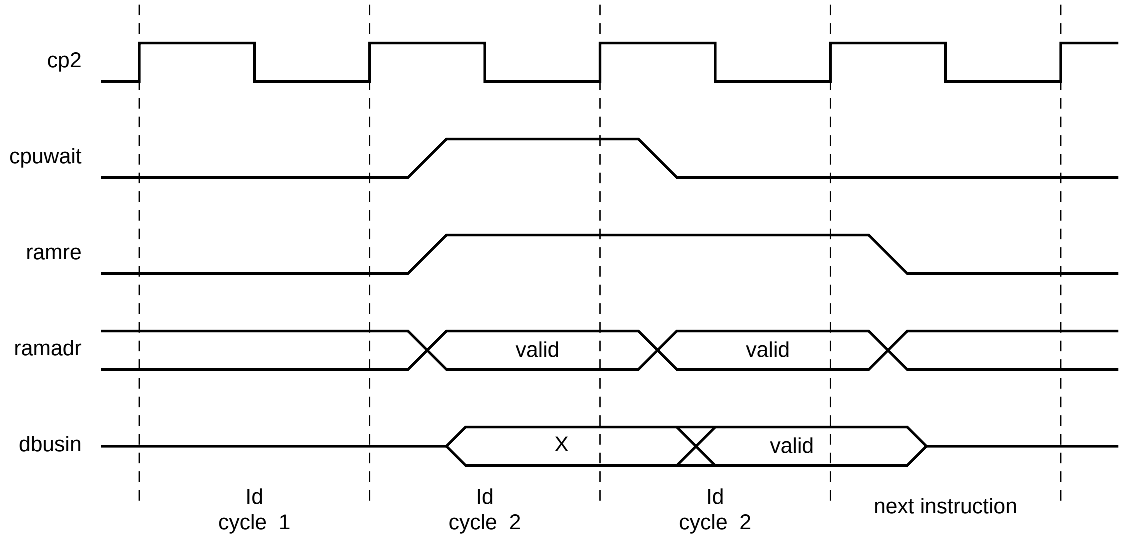
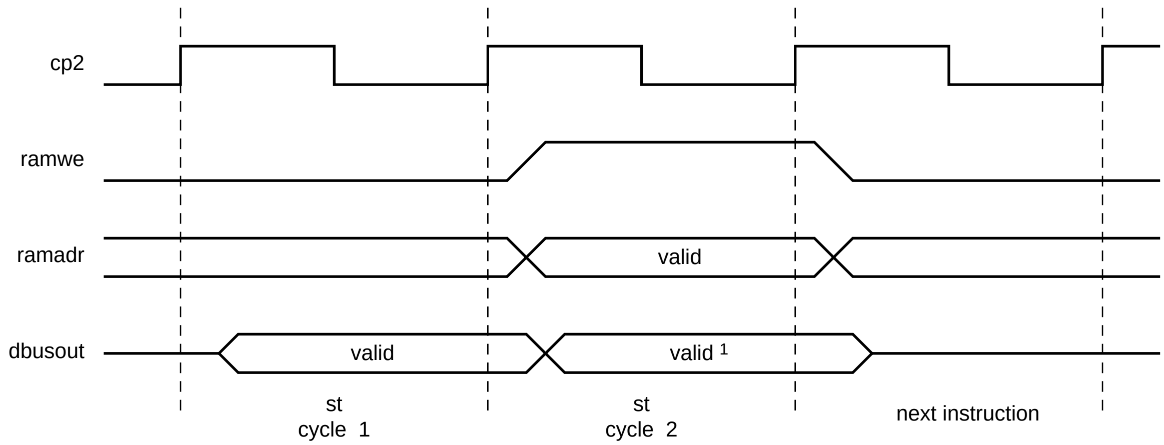
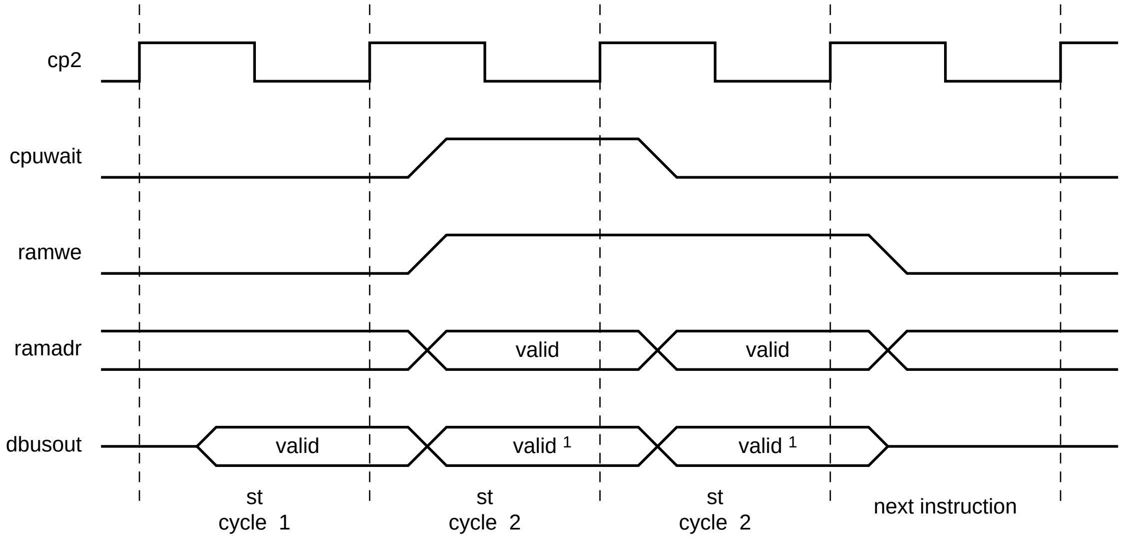


Figure 22. *AVR* SRAM Memory Write, using 'st' or 'sts' instruction.



Note: Not valid when the source register is subject to post-incrementation or pre-decrementation, i.e. the instructions 'st-Z/Z+, r30/r31', 'st-Y/Y+, r28/r29', 'st-X/X+, r26/r27'.

Figure 23. *AVR* SRAM Memory Write, using ‘st’ or ‘sts’ instruction with one wait cycle.



Note: Not valid when the source register is subject to post-incrementation or pre-decrementation, i.e. the instructions ‘st-Z/Z+, r30/r31’, ‘st-Y/Y+, r28/r29’, ‘st-X/X+, r26/r27’.

Stack Access

Figure 24. Pushing Program Counter to SRAM Stack with 'rcall/icall’ instruction.

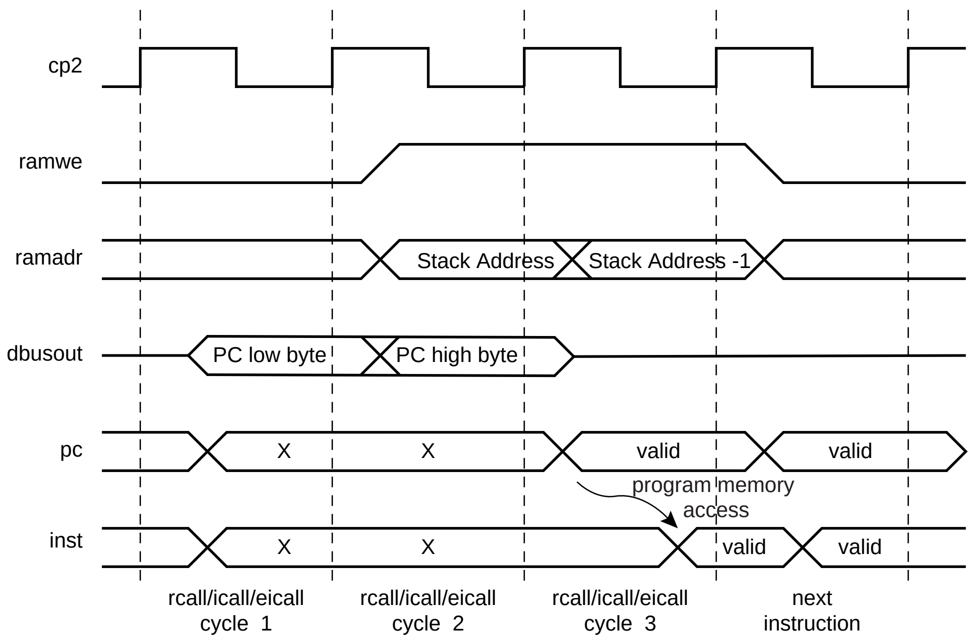


Figure 25. Popping Program Counter from SRAM Stack with 'ret/reti' instruction.

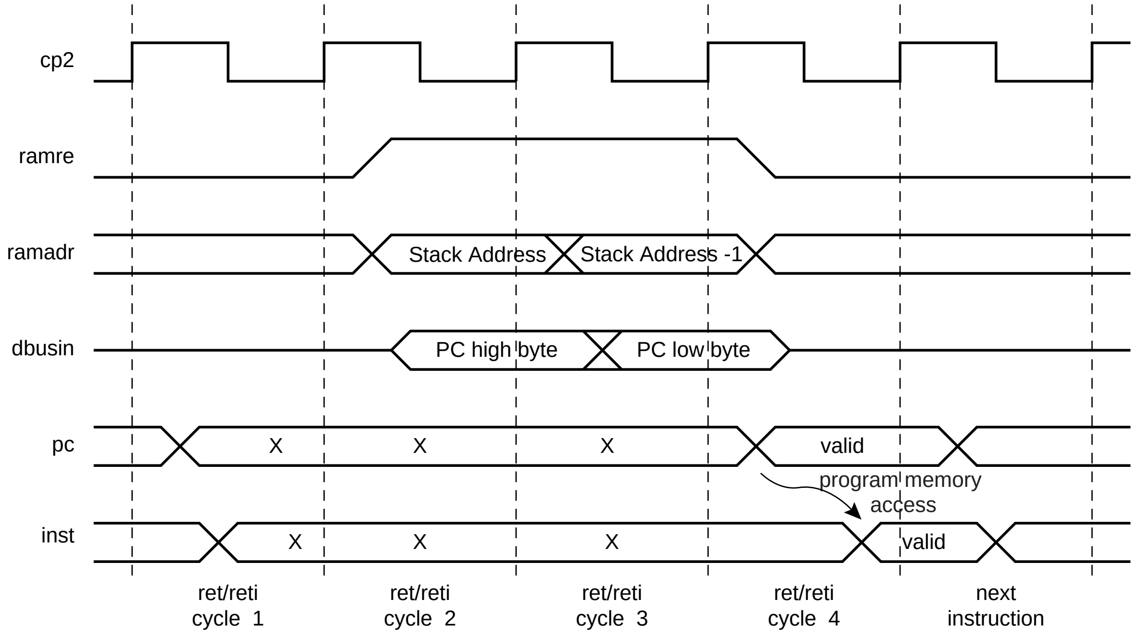


Figure 26. Pushing Register to SRAM Stack with 'push' instruction

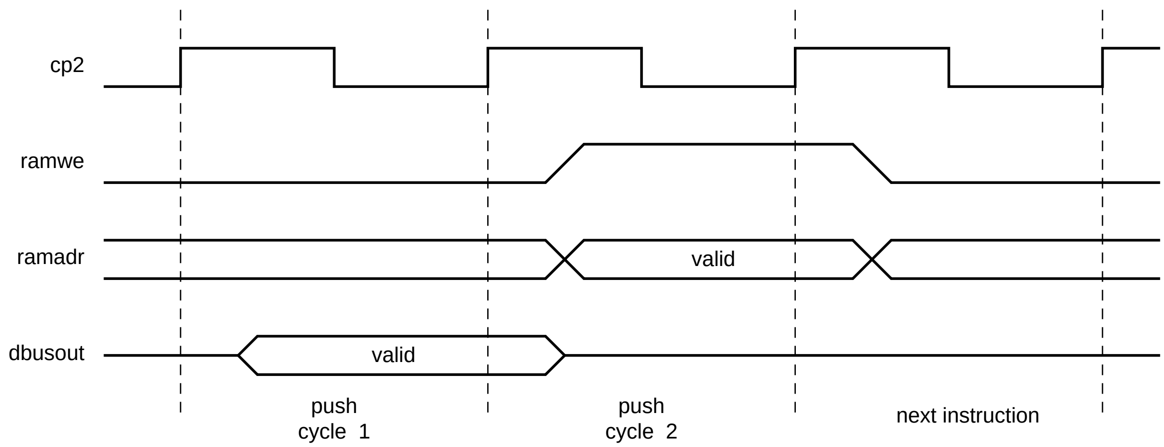
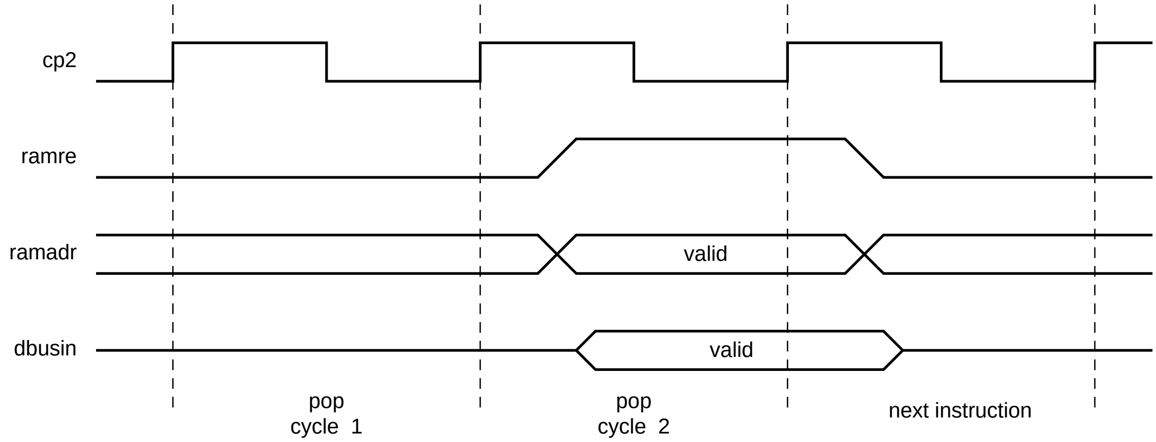


Figure 27. Popping to Register from SRAM Stack with 'pop' instruction.



I/O Memory

The I/O space definition of the *AVR* Core is shown in the following table:

Table 2. *AVR* Core I/O Space

Address Hex	Name	Function
$3F ($5F)	SREG	Status Register
$3E ($5E)	SPH	Stack Pointer High
$3D ($5D)	SPL	Stack Pointer Low
$3C ($5C)	-	Reserved for next AVR Core generation
$3B ($5B)	RAMPZ	RAMPZ Register
$3A ($5A)	-	Reserved for next AVR Core generation
$39 ($39)		
$38 ($58)		
$37 ($57)	-	User specific I/O registers
...		
$00 ($20)		

Note: In parentheses is the SRAM address as the registers can also be addressed as ordinary SRAM locations within the address space $20 - $5F as described in "I/O Registers" below.

Note: Unused locations are not shown in the table

I/O Registers

All the peripheral status, control and data registers can be accessed by making them addressable in the I/O space by connecting **adr**, **iore** and **iowe** signals. The different I/O locations are directly accessed by the IN and OUT instructions transferring data between the 32 general purpose working registers and the I/O space. When using IN and OUT (SBIS and SBIC), the I/O register address $00 - $3F must be used. As the I/O registers are also represented in the SRAM address space, they can also be addressed as ordinary SRAM locations using "ld/lds" and "st/sts" instructions within the address space $20 - $5F. The SRAM address is obtained by adding $20 to the direct I/O address. The SRAM address is given in parentheses after the I/O direct address throughout this document. I/O registers within the address range $00 ($20) - $1F ($3F) are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set chapter for more details.

The different I/O and peripherals control registers are explained in the following sections.

Status Register

The core register most commonly read and written by software is the Status Register (SREG). This register is updated on all arithmetic and logical instructions, and is also supported by special instructions in the instruction set. Software can also access this register to store or manipulate register contents directly.

The *AVR* status register - SREG - at I/O space location $3F is defined as:

Bit	7	6	5	4	3	2	1	0	
$3F ($5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Bit 7 - I: Global Interrupt Enable:

The global interrupt enable bit must be set (one) for the interrupts to be enabled. The individual interrupt enable control must be performed externally. If the global interrupt enable bit is cleared (zero), none of the interrupts are enabled, independent of the external individual enable values. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts.

Bit 6 - T: Bit Copy Storage:

The bit copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T bit as source and destination for the operated bit. A bit from a register in the register file can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the register file by the BLD instruction.

Bit 5 - H: Half Carry Flag:

The half carry flag H indicates a half carry in some arithmetic operations. See the Instruction Set Description for detailed information.

Bit 4 - S: Sign Bit, $S = N \oplus V$:

The S-bit is always an exclusive or between the negative flag N and the two’s complement overflow flag V. See the Instruction Set Description for detailed information.

Bit 3 - V: Two’s Complement Overflow Flag:

The two’s complement overflow flag V supports two’s complement arithmetics. See the Instruction Set Description for detailed information.

Bit 2 - N: Negative Flag:

The negative flag N indicates a negative result after the different arithmetic and logic operations. See the Instruction Set Description for detailed information.

Bit 1 - Z: Zero Flag:

The zero flag Z indicates a zero result after the different arithmetic and logic operations. See the Instruction Set Description for detailed information.

Bit 0 - C: Carry Flag:

The carry flag C indicates a carry in an arithmetic or logic operation. See the Instruction Set Description for detailed information.

The Stack Pointer - SP

The general *AVR* 16-bit Stack Pointer is effectively built up of two 8-bit registers in the I/O space locations $3E ($5E) and $3D ($5D). As the *AVR* Core supports up to 64K bytes SRAM, all 16 bit are used.

Bit	15	14	13	12	11	10	9	8	
$3E ($5E)	**SP15**	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
$3D ($5D)	**SP7**	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

The Stack Pointer points to the data SRAM stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when data is pushed onto the Stack with subroutine CALL and interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

Extended Memory Pointer Registers - RAMPZ

The *AVR* architecture supports four pointers, X-, Y-, Z-, and Stack-Pointer. On systems with more than 64K bytes of program memory, the Z-pointer will not reach the whole memory space with the 16 bits located in the General Purpose Register File. For the Z-pointer to reach the entire memory area, the remaining bit is read and written by software through I/O, through the register RAMPZ.

Bit	7	6	5	4	3	2	1	0	
$3B ($5B)	-	-	-	-	-	-	-	**RAMPZ0**	**RAMPZ**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

The RAMPZ register is normally used to select which 64K RAM Page is accessed by the Z pointer. As the *AVR* Core does not support more than 64K of SRAM memory, this register is used only to select which page in the program memory is accessed when the ‘elpm’ instruction is used. The different settings of the RAMPZ0 bit have the following effects.

RAMPZ0 = 0: Program memory address $0000 - $7FFF (lower 64K bytes) is accessed by ‘elpm’.

RAMPZ0 = 1: Program memory address $8000 - $FFFF (upper 64K bytes) is accessed by ‘elpm’.

I/O Memory Access

I/O registers can be accessed in a single clock cycle. During this clock cycle, the core will issue a 6-bit address on **adr**, and either an **iore** or an **iowe**. While **iore** is high and **adr** matches the address of the register, the I/O register should drive its contents onto **dbusin**. When **iowe** is high and **adr** matches the address of the register, the register should update its contents only if this occurs on a rising edge of **cp2**.

Figure 28. A typical I/O Register Construction. Write is synchronous, read is combinatorial.

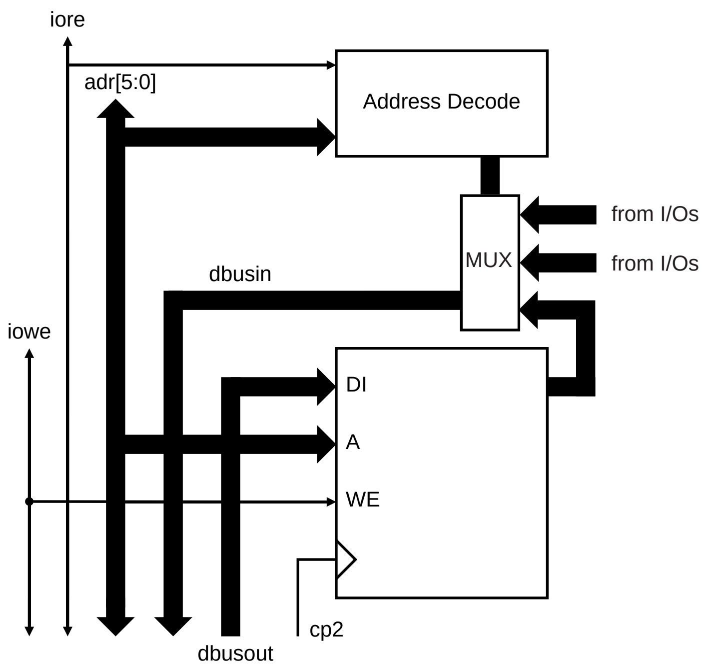


Figure 29. *AVR* I/O Register Read, using 'in' instruction. '**dbusin**' is driven by I/O Register.

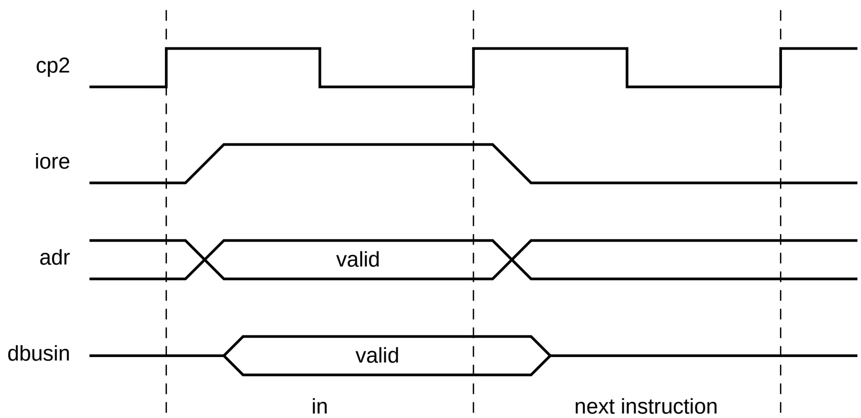
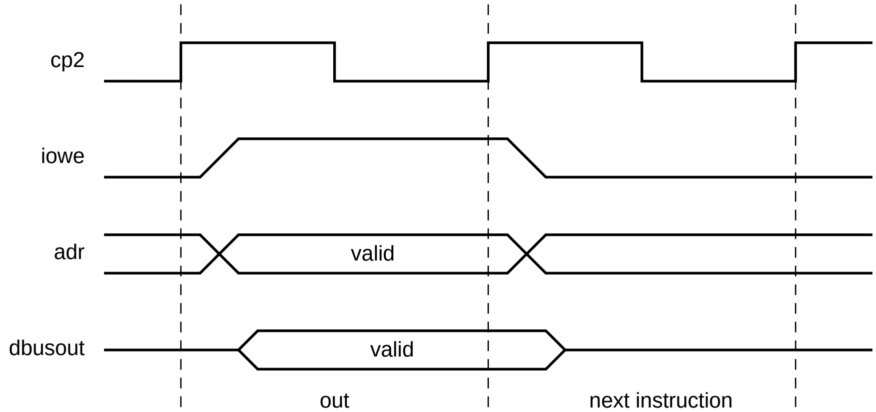


Figure 30. *AVR* I/O Register Write, using 'out' instruction. '**dbusout**' is driven by *AVR* Core.



As the I/O registers are also represented in the SRAM address space, I/O registers are accessible by regular memory access instructions 'ld/ldd/lds' and st/std/sts'. The access will appear like any other memory access, but the address will be presented on adr and mapped from the address range 0x20-0x5F used in SRAM, down to the 0x00-0x3F recognized by the I/O registers. ***This operation is performed automatically by the core***.

Figure 31. *AVR* I/O Register Read, using 'ld' instruction. '**dbusin**' is driven by I/O Register.

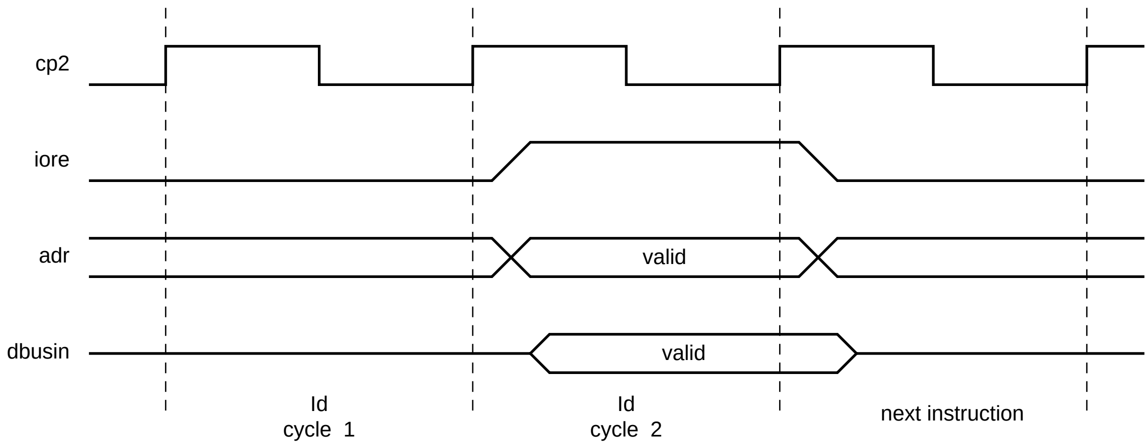
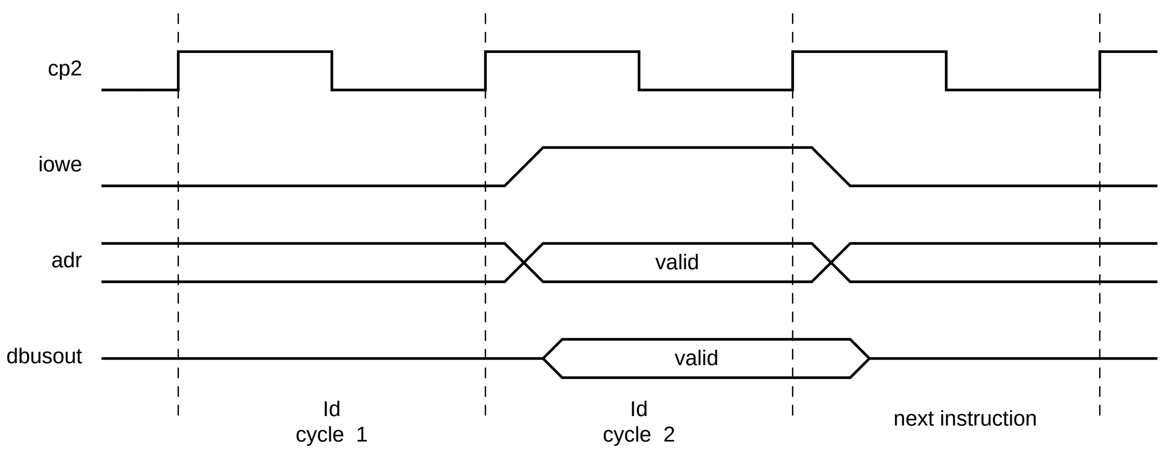


Figure 32. *AVR* I/O Register Write, using 'st' instruction.



Reset and Interrupt Handling

The *AVR* Core provides 7 different interrupt sources. These interrupts and the separate reset vector each have a separate program vector in the program memory space. All interrupts are enabled by the I-bit in the status register.

The lowest addresses in the program memory space are automatically defined as the Reset and Interrupt vectors. The complete list of vectors is shown in . The list also determines the priority levels of the different interrupts. The lower the address, the higher the priority level. **ireset** has the highest priority, and next are **irqlines[0]** to **irqlines[6]**.

Table 3. Reset and Interrupt Vectors

Vector No.	Program Address	Source	Interrupt Definition
1	$000	ireset	Internal Reset
2	$002	irqlines[0]	Interrupt Request Line 0
3	$004	irqlines[1]	Interrupt Request Line 1
4	$006	irqlines[2]	Interrupt Request Line 2
5	$008	irqlines[3]	Interrupt Request Line 3
6	$00A	irqlines[4]	Interrupt Request Line 4
7	$00C	irqlines[5]	Interrupt Request Line 5
8	$00E	irqlines[6]	Interrupt Request Line 6

The most typical and general program setup for the Reset and Interrupt Vector Addresses are:

```
Address           Labels       Code              Comments
$000                           rjmp    RESET     ; Reset Handle
$001                           nop               ;
$002                           jmp     INT0      ; IRQ0 Handle
$004                           jmp     INT1      ; IRQ1 Handle
$006                           rjmp    INT2      ; IRQ2 Handle
$007                           nop
$008                           rjmp    INT3      ; IRQ3 Handle
$009                           nop
$00A                           rjmp    INT4      ; IRQ4 Handle
$00B                           nop
$00C                           jmp     INT5      ; IRQ5 Handle
$00E                           jmp     INT6      ; IRQ6 Handle
;
$010              RESET:       <instr>  xxx      ; Main program start
  …                  …           …      …
```

Reset

The **ireset** line controls the reset of the *AVR* Core. To properly reset the *AVR*, a three cycle pulse must be applied to the **ireset** input. After this, the program counter is reset to 0000 and the *AVR* Core is ready.

Note: Setup and hold times must be respected on the **ireset** line (see timing diagrams).

Interrupts

None of the **irqlines** are latched in the *AVR* Core. For this reason, it is necessary to maintain the **irqlines[6:0]** signals until the corresponding acknowledgment. The figure below shows the interrupt acknowledgment schemes.

Figure 33. Interrupt arriving in last cycle of instruction

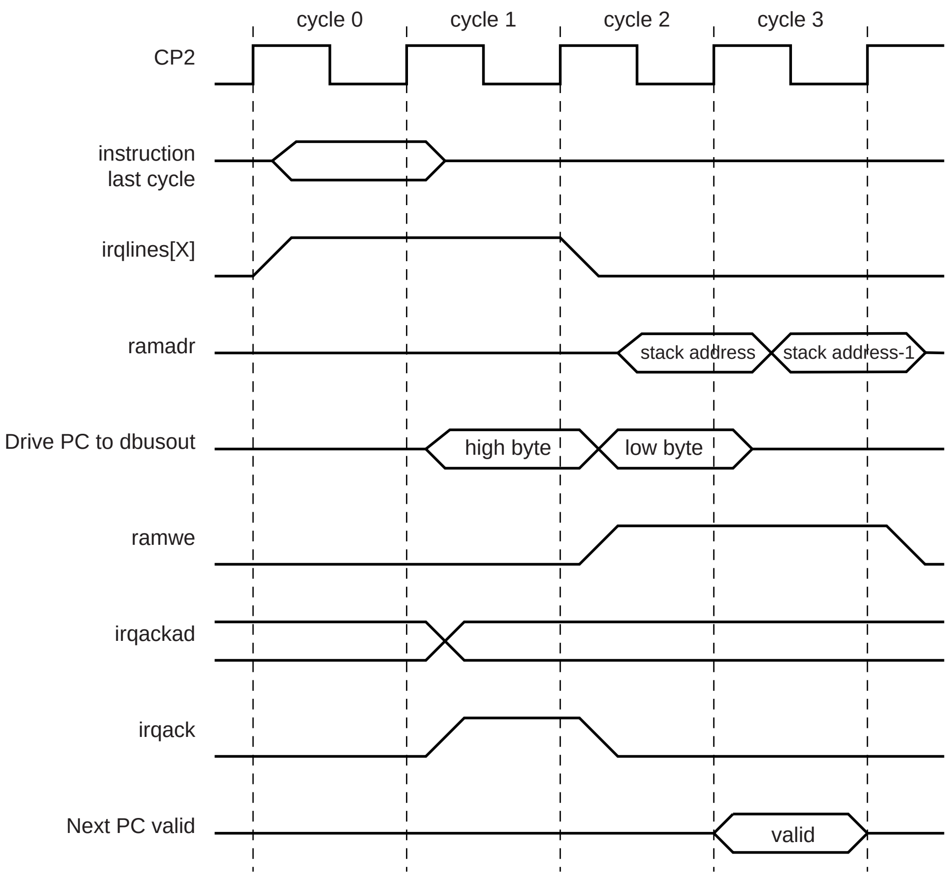
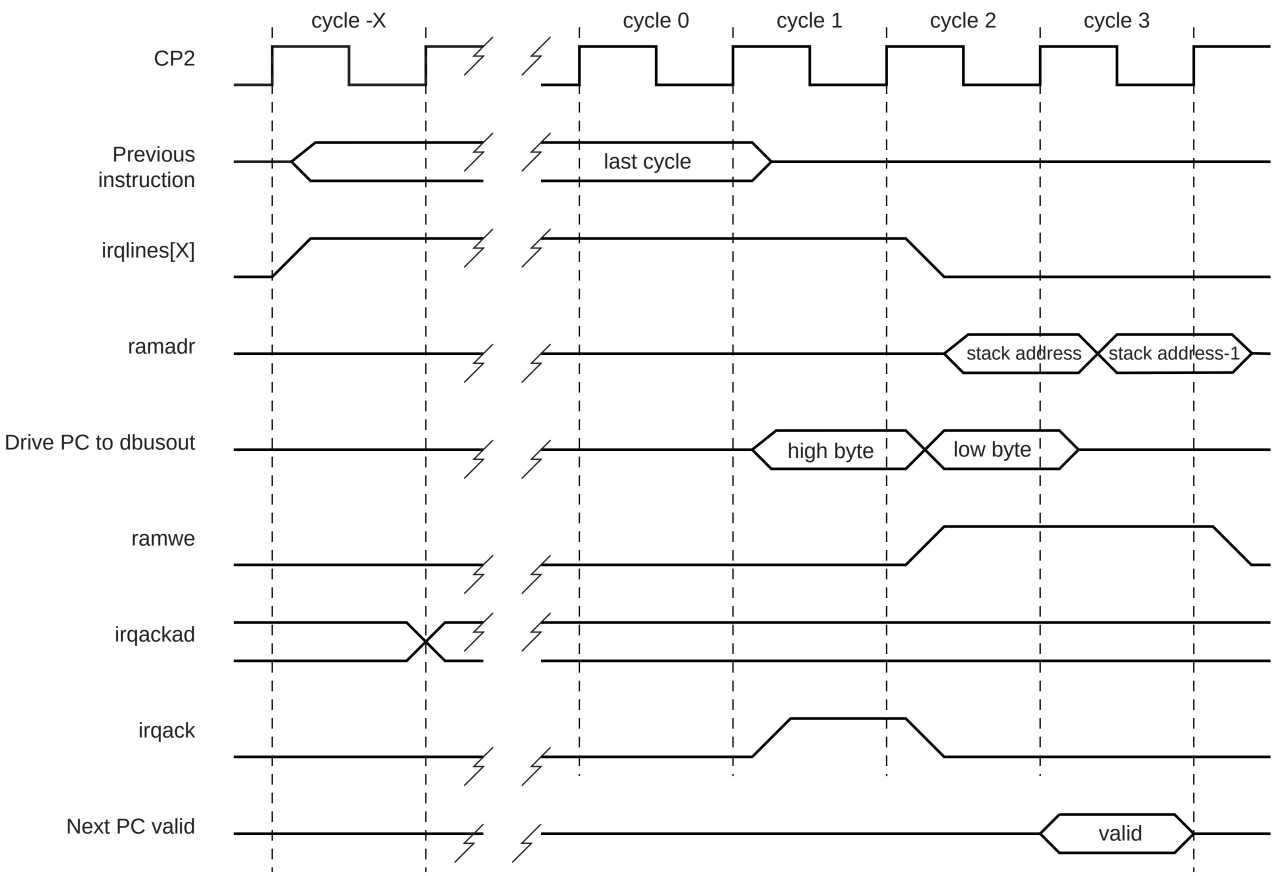


Figure 34. Interrupt arriving in cycle other than last



AVR Scalable Test Access (ASTA) Interface

The AVR Scalable Test Access (ASTA) interface provides designers with great flexibility to test the AVR Embedded Core and its peripherals. First, the ASTA architecture allows the designer to apply pre-computed ATPG test vectors with more than 99% fault coverage. Secondly, it allows ATPG vectors to be generated for the rest of the chip. The main characteristic of the ASTA architecture however, is its capability to be scaled and split into several scan chains, making it possible to test the program memory space, the RAM space and the I/O space simultaneously.

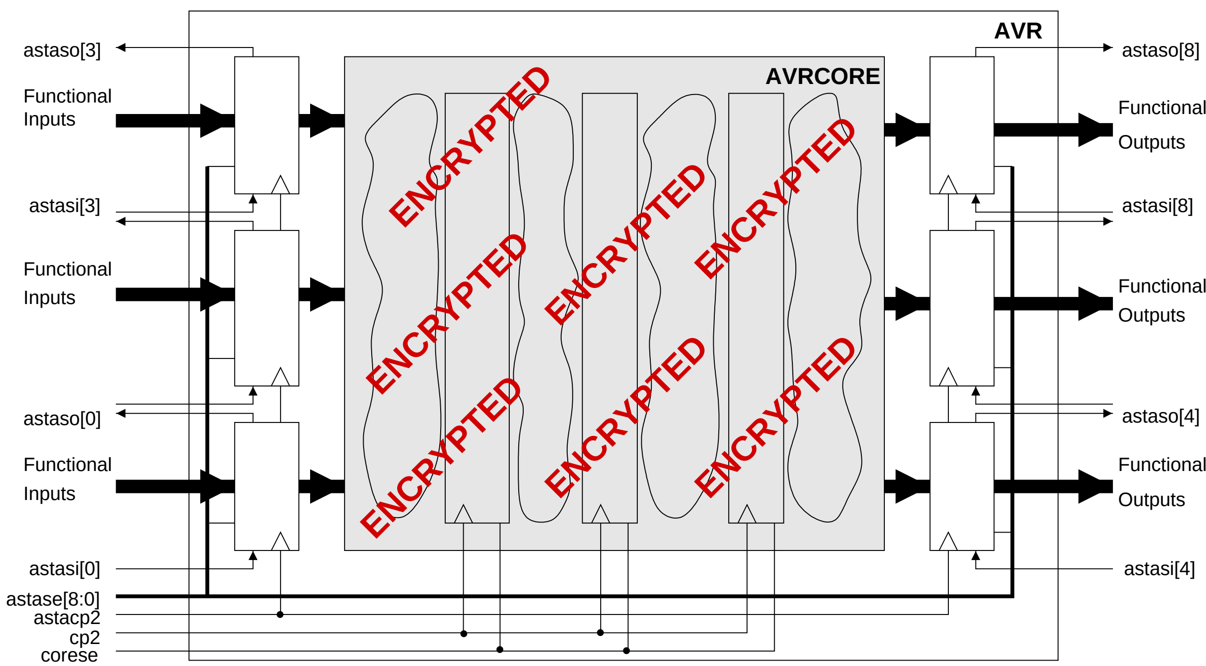
The ASTA interface can be considered as a boundary scan ring that encompasses the entire AVR Embedded Core. This scan chain allows all primary AVR inputs to be controlled and all primary AVR outputs to be observed, resulting in over a 99% fault coverage. This scan ring is actually split into nine different scan chains which can be grouped as desired, giving the flexibility to create specific tests such as RAM space testing or program memory space testing.

All of the scan chains which form the ASTA scan ring have a common clock (**astacp2**). However, they have separate scan inputs (**astasi[8:0]**) and outputs (**astaso[8:0]**) as well as separate scan enable signals (**astase[8:0]**) which gives this architecture its flexibility.

To achieve 99% fault coverage, the three internal scan chains which have **coresi[2:0]** for inputs, **coreso[2:0]** for outputs and a common **corese** for scan enable must be used for applying ATPG vectors.

The ASTA interface is shown below:

Figure 35. The ASTA Interface.



ASTA Signals

The signals which control the ASTA interface are described below.

Table 4. ASTA Signals

Signal	Description
astacp2	Clock for all ASTA flip-flops
astamode[1:0]	ASTA mode select for inputs (astamode[0]) and outputs (astamode[1])
astamode[0] = 0	**Functional Mode** and **External Capture Mode** The ASTA input interface is transparent. This implies that the device is in Functional Mode. Nevertheless, the ASTA scan chains can capture all AVR input signals.
astamode[0] = 1	**Internal Control Mode** The ASTA input interface is controlled by ASTA scan chains. The ASTA interface can control all AVR input signals.
astamode[1] = 0	**Functional Mode** and **Internal Capture Mode** The ASTA output interface is transparent. This implies that the device is in Functional Mode. Nevertheless, the ASTA scan chains can capture all AVR output signals.
astamode[1] = 1	**External Control Mode** The ASTA output interface is controlled by ASTA scan chains. The ASTA interface can control all AVR output signals.
astasi[8:0]	ASTA scan inputs
astaso[8:0]	ASTA scan outputs
astase[8:0]	ASTA scan enables

ASTA Scan Chains

The ASTA architecture is based on the possibility of joining the ASTA scan chains in order to dynamically create new scan chains dedicated to a specific test goal. The ASTA architecture is formed by nine scan chains which are defined below:

Table 5. ASTA Scan Chains

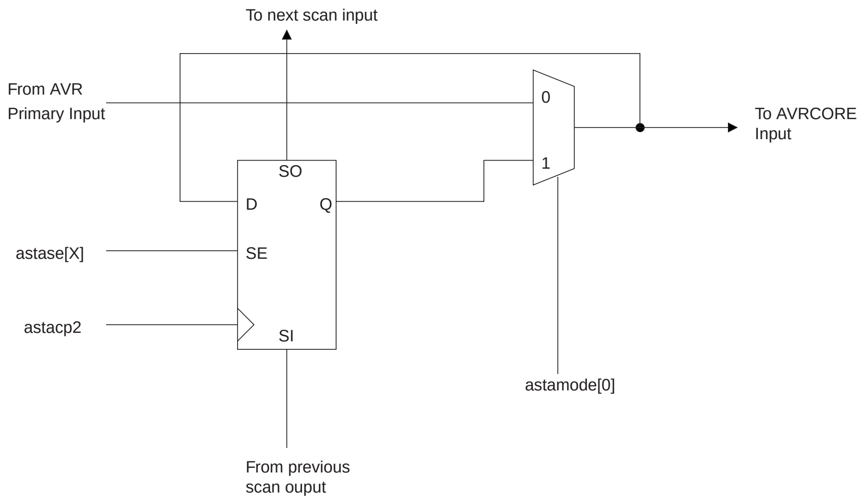
ASTA Chain	Scan Input	Scan Output	Scan Enable	AVR Inputs/Outputs Controlled/Observed
ASTA chain 0	astasi[0]	astaso[0]	astase[0]	inst[15:0]
ASTA chain 1	astasi[1]	astaso[1]	astase[1]	dbusin[7:0]
ASTA chain 2	astasi[2]	astaso[2]	astase[2]	irqlines[6:0]
ASTA chain 3	astasi[3]	astaso[3]	astase[3]	control = lbit12, pcld[1:0], pclden, leavbus, cpuwait, ireset
ASTA chain 4	astasi[4]	astaso[4]	astase[4]	irq outputs = globint, irqackad[2:0], irqack, irqok, sleepi, wdri
ASTA chain 5	astasi[5]	astaso[5]	astase[5]	RAM space = ramadr[15:0], ramwe, ramre
ASTA chain 6	astasi[6]	astaso[6]	astase[6]	I/O space = adr[5:0], iowe, iore
ASTA chain 7	astasi[7]	astaso[7]	astase[7]	dbusout[7:0]
ASTA chain 8	astasi[8]	astaso[8]	astase[8]	pc[15:0]

In all scan chains LSB is scanned in first and MSB scanned out first.

ASTA Scan Input Cell

Each primary input of the AVR Embedded core is connected to an ASTA scan input cell which can be configured to run in different modes.

Figure 36. ASTA Scan Input Cell



The following signals control the ASTA scan input cell:

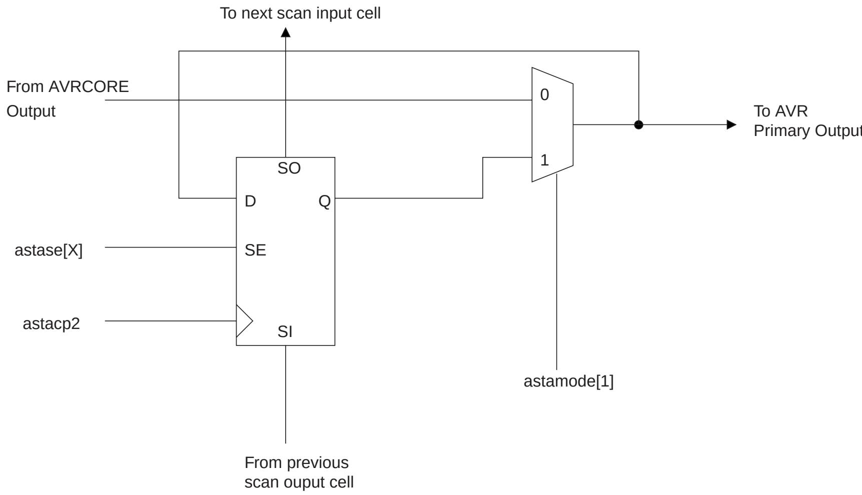
Table 6. ASTA Scan Input Cell

Signal	Description
astacp2	Clock for all ASTA flip-flops
astase[X]	Local scan enable for the selected ASTA scan chain
astamode[0]	Test mode selector
astamode[0] = 0	**Functional Mode** and **External Capture Mode** The ASTA input interface is transparent. This implies that the device is in Functional Mode. Nevertheless, the ASTA scan chains can capture all AVR input signals. See Table 4.
astamode[0] = 1	**Internal Control Mode** The ASTA input interface is controlled by ASTA scan chains. The ASTA interface can control all AVR input signals. See Table 4.

ASTA Scan Output Cell

Each primary output of the AVR Embedded core is connected to an A.S.T.A scan output cell which can be configured in different modes.

Figure 37. ASTA Scan Output Cell



The following signals control the ASTA scan output cell:

Table 7. ASTA Scan Output Cell

Signal	Description
astacp2	Clock for all ASTA flip-flops
astase[X]	Local scan enable for the selected ASTA scan chain
astamode[1]	Test mode selector
astamode[1] = 0	**Functional Mode** and **Internal Capture Mode** The ASTA output interface is transparent. This is Functional Mode. Nevertheless, the ASTA scan chains can capture all AVR output signals. See Table 4.
astamode[1] = 1	**External Control Mode** The ASTA output interface is controlled by ASTA scan chains. The ASTA interface can control all AVR output signals. See Table 4.

Testing the AVR Embedded Core

The AVR Embedded Core is shipped with a pre-computed set of ATPG test vectors ensuring over a 99% fault coverage. This set of vectors is generated with a special configuration of the ASTA interface. The designer must recreate this configuration in his design.

In order to apply the precomputed ATPG vectors:

1. The designer must have access to the following top level pins:
   cp2: global clock
   test_se: new global scan enable (to be created by the user)
   astamode[1:0]: astamode selectors
   coresi[2:0]: Internal scan chain inputs
   coreso[2:0]: Internal scan chain outputs
   asta_scin: ASTA boundary ring input (to be created by the user)
   asta_scout: ASTA boundary ring output (to be created by the user)
2. The following signals must be tied together:
   cp2 = astacp2
   test_se = corese = astase[8:0]
3. The ASTA boundary ring must be connected as follows:
   asta_scin = astasi[0]
   astaso[0] = astasi[1]

astaso[1] = astasi[2]
astaso[2] = astasi[3]
astaso[3] = astasi[4]
astaso[4] = astasi[5]
astaso[5] = astasi[6]
astaso[6] = astasi[7]
astaso[7] = astasi[8]
astaso[8] = asta_scout
which results in a scan chain linked as follows:
asta_scin, ASTA chain 0, ASTA chain 1, ASTA chain 2, ASTA chain 3, ASTA chain 4, ASTA chain 5, ASTA chain 6, ASTA chain 7, ASTA chain 8, asta_scout

Testing the AVR Peripherals

The AVR Peripherals can be separated into three classes: standard AVR peripherals (AVR UART, AVR SPI, etc.), other embedded Macros and User Defined Logic (UDL). All can be tested using the ASTA interface.

Methodologies

Because the test goal for today's designs is over a 99% fault coverage, the recommended methodology for testing designs containing the AVR Embedded core is Full Scan. This is the simplest methodology. In specific cases however, good coverage can be achieved with Partial Scan, BIST or non scan techniques. To allow a large amount of freedom in a design, the ASTA architecture makes it possible to use all of these special DFT methodologies.

Test Configuration

A fourth scan chain must be created around the AVR Embedded Core by linking the nine ASTA scan chains as described in the previous section. With this scan chain, the AVR Embedded Core can be bypassed (the core is an ATPG black box for the designer due to its encrypted format), and ATPG test vectors can then be generated for the rest of the chip, including AVR standard peripherals, embedded Macros and UDL. Partial Scan or ad-hoc methodologies can also be used by controlling this scan chain.

Special Tests with the ASTA Interface

Using the ASTA Interface allows all AVR output pins to be controlled and all AVR input pins to be observed. Specific tests can then be created as described below.

Ram Space Testing

By linking ASTA chain 5 with ASTA chain 1, test sequences can be generated to verify the correct functionality of the RAM itself or of the RAM space mapping.

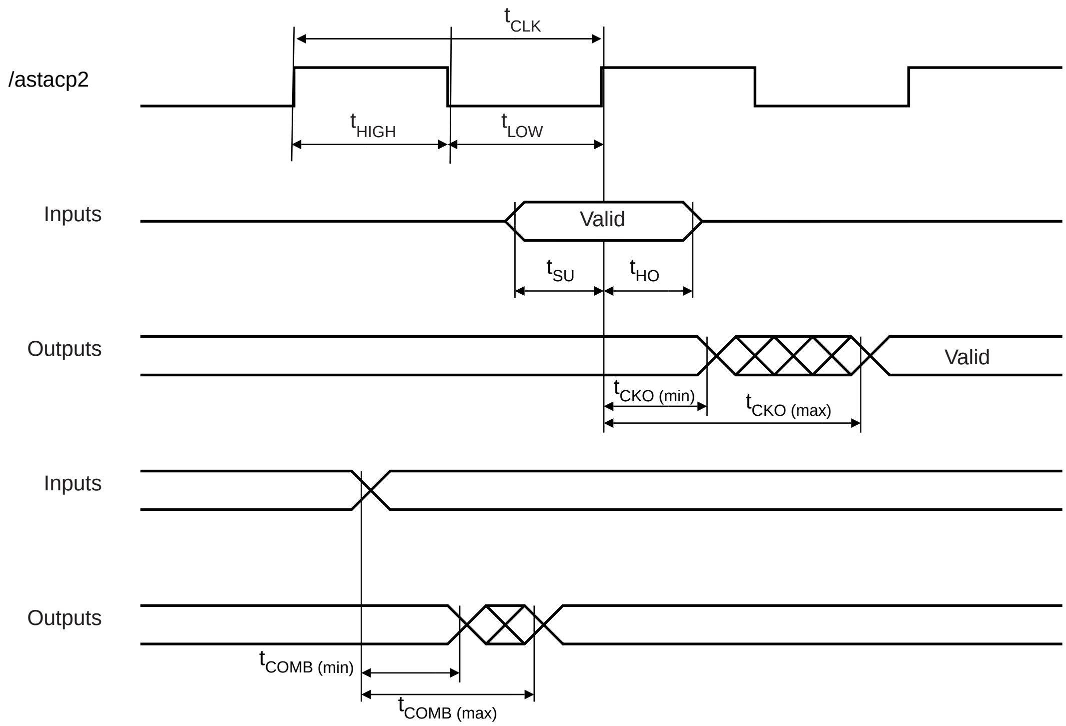
I/O Space Testing

By linking ASTA chain 6 with ASTA chain 1, test sequences can be generated to verify the correct functionality of the I/O space mapping.

Program Memory Space Testing

By linking ASTA chain 8 with ASTA chain 0, test sequences can be generated to verify the correct functionality of the Program Memory space.

Input/Output Timing



Symbol	Parameter
t_{CP2}	Clock cycle
t_{SU}	Input setup time
t_{HO}	Input hold time
t_{CKO}	Clock to output delay
t_{COMB}	Combinational delay; input to output

Note*: The delays shown in this diagram are all process specific. For the corresponding characterized values, refer to one of the following datasheets*:

- AVR Embedded Core ATC50 Electrical Characteristics
  (0.5 micron three-layer-metal CMOS process intended for use with a supply voltage of 3.3V ± 0.3V)
- AVR Embedded Core ATC50/E^2 Electrical Characteristics
  (0.5 micron three-layer-metal CMOS/NVM process intended for use with a supply voltage of 3.3V ± 0.3V)
- AVR Embedded Core ATC35 Electrical Characteristics
  (0.35 micron three-layer-metal CMOS process intended for use with a supply voltage of 3.3V ± 0.3V)

AVR Core Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	19
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	20
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	20
$3C ($5C)	-	Reserved for next AVR Core generation								
$3B ($5B)	RAMPZ	-	-	-	-	-	-	-	RAMPZ0	20
$3A ($5A)	-	Reserved for next AVR Core generation								
$39 ($59)										
$38 ($58)										
$37 ($57)	-	User Specific I/O Registers								
$36 ($56)										
...										
$01 ($21)										
$00 ($20)										

AVR Core Instruction Set

Instruction Set Nomenclature:

Status Register (SREG):
SREG: Status register
C: Carry flag in status register
Z: Zero flag in status register
N: Negative flag in status register
V: Twos complement overflow indicator
S: N ⊕ V, For signed tests
H: Half Carry flag in the status register
T: Transfer bit used by BLD and BST instructions
I: Global interrupt enable/disable flag

Registers and operands:
Rd: Destination (and source) register in the register file
Rr: Source register in the register file
R: Result after instruction is executed
K: Constant literal or byte data (8 bit)
k: Constant address data for program counter
b: Bit in the register file or I/O register (3 bit)
s: Bit in the status register (3 bit)

X,Y,Z: Indirect address register (X=R27:R26, Y=R29:R28 and Z=R31:R30)
P: I/O port address
q: Displacement for direct addressing (6 bit)

I/O Registers
RAMPZ: Register concatenated with the Z register enabling indirect addressing of the whole Program Area on MCUs with more than 64K bytes of Program Code (ELPM instruction).

Stack:
STACK:Stack for return address and pushed registers
SP: Stack Pointer to STACK

Flags:
⇔: Flag affected by instruction
0: Flag cleared by instruction
1: Flag set by instruction
-: Flag not affected by instruction

Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO*	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH*	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

* Interchange Rd and Rr in the operation before the test. i.e. CP Rd,Rr → CP Rr,Rd

Complete Instruction Set Summary

Mnemonic	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1

(continued)

Complete Instruction Set Summary (continued)

Mnemonic	Operands	Description	Operation	Flags	#Clocks
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump to k	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call to k	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2

(continued)

Complete Instruction Set Summary (continued)

Mnemonic	Operands	Description	Operation	Flags	#Clocks
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	3
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	3
LPM		Load Program Memory	R0 ← (Z)	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ, Z)	None	3
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2

(continued)

Complete Instruction Set Summary (continued)

Mnemonic	Operands	Description	Operation	Flags	#Clocks
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	3
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1

ADC - Add with Carry

Description:

Adds two registers and the contents of the C flag and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd + Rr + C$

	Syntax:	Operands:	Program Counter:
(i)	ADC Rd,Rr	$0 \le d \le 31, 0 \le r \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

0001	11rd	dddd	rrrr

Status Register (SREG) Boolean Formulae:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{Rd7} \bullet \overline{Rr7} \bullet \overline{Rr7} \bullet \overline{R7} \bullet \overline{R7} \bullet \overline{Rd7}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
                ; Add R1:R0 to R3:R2
add   r2,r0     ; Add low byte
adc   r3,r1     ; Add with carry high byte
```

Words: 1 (2 bytes)
Cycles: 1

ADD - Add without Carry

Description:

Adds two registers without the C flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16 bit Opcode:

0000	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add   r1,r2    ; Add r2 to r1 (r1=r1+r2)
add   r28,r28  ; Add r28 to itself (r28=r28+r28)
```

Words: 1 (2 bytes)
Cycles: 1

ADIW - Add Immediate to Word

Description:

Adds an immediate value (0-63) to a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

Operation:

(i) Rdh:Rdl ← Rdh:Rdl + K

	Syntax:	Operands:	Program Counter:
(i)	ADIW Rdl,K	dl ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16 bit Opcode:

1001	0110	KKdd	KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: Rdh7 R15
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $\overline{R15} \bullet Rdh7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
adiw  r24,1      ; Add 1 to r25:r24
adiw  r30,63     ; Add 63 to the Z pointer(r31:r30)
```

Words: 1 (2 bytes)
Cycles: 2

AND - Logical AND

Description:

Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd \bullet Rr$

	Syntax:	Operands:	Program Counter:
(i)	AND Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

0010	00rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
and   r2,r3      ; Bitwise and r2 and r3, result in r2
ldi   r16,1      ; Set bitmask 0000 0001 in r16
and   r2,r16     ; Isolate bit 0 in r2
```

Words: 1 (2 bytes)

Cycles: 1

ANDI - Logical AND with Immediate

Description:

Performs the logical AND between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • K

	Syntax:	Operands:	Program Counter:
(i)	ANDI Rd,K	$16 \le d \le 31$, $0 \le K \le 255$	PC ← PC + 1

16 bit Opcode:

0111	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
andi  r17,$0F    ; Clear upper nibble of r17
andi  r18,$10    ; Isolate bit 4 in r18
andi  r19,$AA    ; Clear odd bits of r19
```

Words: 1 (2 bytes)

Cycles: 1

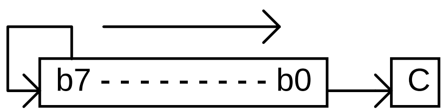
ASR - Arithmetic Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C flag of the SREG. This operation effectively divides a twos complement value by two without changing its sign. The carry flag can be used to round the result.

Operation:

(i)



	Syntax:	**Operands:**	**Program Counter:**
(i)	ASR Rd	$0 \le d \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	010d	dddd	0101

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $N \oplus C$ (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
ldi   r16,$10      ; Load decimal 16 into r16
asr   r16          ; r16=r16 / 2
ldi   r17,$FC      ; Load -4 in r17
asr   r17          ; r17=r17/2
```

Words: 1 (2 bytes)
Cycles: 1

BCLR - Bit Clear in SREG

Description:

Clears a single flag in SREG.

Operation:

(i) SREG(s) ← 0

	Syntax:	Operands:	Program Counter:
(i)	BCLR s	0 ≤ s ≤ 7	PC ← PC + 1

16 bit Opcode:

1001	0100	1sss	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I: 0 if s = 7; Unchanged otherwise.

T: 0 if s = 6; Unchanged otherwise.

H: 0 if s = 5; Unchanged otherwise.

S: 0 if s = 4; Unchanged otherwise.

V: 0 if s = 3; Unchanged otherwise.

N: 0 if s = 2; Unchanged otherwise.

Z: 0 if s = 1; Unchanged otherwise.

C: 0 if s = 0; Unchanged otherwise.

Example:

```
bclr   0      ; Clear carry flag
bclr   7      ; Disable interrupts
```

Words: 1 (2 bytes)

Cycles: 1

BLD - Bit Load from the T Flag in SREG to a Bit in Register.

Description:

Copies the T flag in the SREG (status register) to bit b in register Rd.

	Operation:
(i)	Rd(b) ← T

	Syntax:	**Operands:**	**Program Counter:**
(i)	BLD Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16 bit Opcode:

1111	100d	dddd	0bbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
                ; Copy bit
    bst   r1,2    ; Store bit 2 of r1 in T flag
    bld   r0,4    ; Load T flag into bit 4 of r0
```

Words: 1 (2 bytes)

Cycles: 1

BRBC - Branch if Bit in SREG is Cleared

Description:

Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is cleared. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form.

Operation:

(i) If SREG(s) = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRBC s,k	0 ≤ s ≤ 7, -64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	ksss

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
cpi   r20,5     ; Compare r20 to the value 5
brbc  1,noteq   ; Branch if zero flag cleared
...
noteq:nop       ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRBS - Branch if Bit in SREG is Set

Description:

Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is set. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form.

Operation:

(i) If SREG(s) = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRBS s,k	0 ≤ s ≤ 7, -64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	ksss

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        bst   r0,3      ; Load T bit with bit 3 of r0
        brbs  6,bitset  ; Branch T bit was set
        ...
bitset: nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRCC - Branch if Carry Cleared

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is cleared. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k).

Operation:

(i) If C = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRCC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            addr22,r23    ; Add r23 to r22
            brccnocarry  ; Branch if carry cleared
            ...
   nocarry: nop          ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRCS - Branch if Carry Set

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is set. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

(i) If C = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRCS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        cpi   r26,$56    ; Compare r26 with $56
        brcs  carry      ; Branch if carry set
        ...
carry:  nop              ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BREQ - Branch if Equal

Description:

Conditional relative branch. Tests the Zero flag (Z) and branches relatively to PC if Z is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 1,k).

Operation:

(i) If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BREQ k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k001

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        cpr1,r0    ; Compare registers r1 and r0
        breqequal  ; Branch if registers equal
        ...
equal:  nop        ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRGE - Branch if Greater or Equal (Signed)

Description:

Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 4,k).

Operation:

(i) If Rd ≥ Rr ($N \oplus V = 0$) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	**Operands:**	**Program Counter:**
(i)	BRGE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k100

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            cpr11,r12     ; Compare registers r11 and r12
            brgegreateq   ; Branch if r11 >= r12 (signed)
            ...
   greateq: nop           ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRHC - Branch if Half Carry Flag is Cleared

Description:

Conditional relative branch. Tests the Half Carry flag (H) and branches relatively to PC if H is cleared. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 5,k).

Operation:

(i) If H = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRHC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k101

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            brhc hclear        ; Branch if half carry flag cleared
            ...
    hclear: nop                ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRHS - Branch if Half Carry Flag is Set

Description:

Conditional relative branch. Tests the Half Carry flag (H) and branches relatively to PC if H is set. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 5,k).

Operation:

(i) If H = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRHS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k101

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        brhshset     ; Branch if half carry flag set
        ...
hset:   nop          ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

BRID - Branch if Global Interrupt is Disabled

Description:

Conditional relative branch. Tests the Global Interrupt flag (I) and branches relatively to PC if I is cleared. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 7,k).

Operation:

(i) If I = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRID k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k111

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            brid intdis    ; Branch if interrupt disabled
            ...
    intdis: nop            ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRIE - Branch if Global Interrupt is Enabled

Description:

Conditional relative branch. Tests the Global Interrupt flag (I) and branches relatively to PC if I is set. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 7,k).

Operation:

(i) If I = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRIE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k111

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        brie inten     ; Branch if interrupt enabled
        ...
inten:  nop            ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRLO - Branch if Lower (Unsigned)

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned binary number represented in Rd was smaller than the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

(i) If Rd < Rr (C = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRLO k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        eor   r19,r19     ; Clear r19
loop:   inc   r19         ; Increase r19
        ...
        cpi   r19,$10     ; Compare r19 with $10
        brlo  loop        ; Branch if r19 < $10 (unsigned)
        nop               ; Exit from loop (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRLT - Branch if Less Than (Signed)

Description:

Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was less than the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 4,k).

Operation:

(i) If Rd < Rr ($N \oplus V = 1$) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRLT k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k100

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        cp    r16,r1       ; Compare r16 to r1
        brlt  less         ; Branch if r16 < r1 (signed)
        ...
less:   nop                ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRMI - Branch if Minus

Description:

Conditional relative branch. Tests the Negative flag (N) and branches relatively to PC if N is set. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 2,k).

Operation:

(i) If N = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRMI k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k010

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            subi    r18,4       ; Subtract 4 from r18
            brmi    negative    ; Branch if result negative
            ...
negative:   nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRNE - Branch if Not Equal

Description:

Conditional relative branch. Tests the Zero flag (Z) and branches relatively to PC if Z is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was not equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 1,k).

Operation:

(i) If Rd $\neq$ Rr (Z = 0) then PC $\leftarrow$ PC + k + 1, else PC $\leftarrow$ PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRNE k	$-64 \leq k \leq +63$	PC $\leftarrow$ PC + k + 1 PC $\leftarrow$ PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k001

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        eor     r27,r27   ; Clear r27
loop:   inc     r27       ; Increase r27
        ...
        cpi     r27,5     ; Compare r27 to 5
        brne    loop      ; Branch if r27<>5
        nop               ; Loop exit (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRPL - Branch if Plus

Description:

Conditional relative branch. Tests the Negative flag (N) and branches relatively to PC if N is cleared. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 2,k).

Operation:

(i) If N = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRPL k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k010

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
                subi  r26,$50          ; Subtract $50 from r26
                brpl  positive         ; Branch if r26 positive
                ...
   positive:    nop                    ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

BRSH - Branch if Same or Higher (Unsigned)

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is cleared. If the instruction is executed immediately after execution of any of the instructions CP, CPI, SUB or SUBI the branch will occur if and only if the unsigned binary number represented in Rd was greater than or equal to the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k).

Operation:

(i) If Rd ≥Rr (C = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRSH k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            subi  r19,4        ; Subtract 4 from r19
            brsh  highsm       ; Branch if r19 >= 4 (unsigned)
            ...
    highsm: nop                ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRTC - Branch if the T Flag is Cleared

Description:

Conditional relative branch. Tests the T flag and branches relatively to PC if T is cleared. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 6,k).

Operation:

(i) If T = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRTC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k110

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            bst     r3,5        ; Store bit 5 of r3 in T flag
            brtc    tclear      ; Branch if this bit was cleared
            ...
    tclear: nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRTS - Branch if the T Flag is Set

Description:

Conditional relative branch. Tests the T flag and branches relatively to PC if T is set. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 6,k).

Operation:

(i) If T = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRTS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k110

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        bst   r3,5      ; Store bit 5 of r3 in T flag
        brts  tset      ; Branch if this bit was set
        ...
tset:   nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

BRVC - Branch if Overflow Cleared

Description:

Conditional relative branch. Tests the Overflow flag (V) and branches relatively to PC if V is cleared. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 3,k).

Operation:

(i) If V = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRVC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	01kk	kkkk	k011

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
          add   r3,r4       ; Add r4 to r3
          brvc  noover      ; Branch if no overflow
          ...
noover:   nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRVS - Branch if Overflow Set

Description:

Conditional relative branch. Tests the Overflow flag (V) and branches relatively to PC if V is set. This instruction branches relatively to PC in either direction (PC-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 3,k).

Operation:

(i) If V = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRVS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k011

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            add     r3,r4       ; Add r4 to r3
            brvs    overfl      ; Branch if overflow
            ...
overfl:     nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BSET - Bit Set in SREG

Description:

Sets a single flag or bit in SREG.

Operation:

(i) SREG(s) ← 1

	Syntax:	Operands:	Program Counter:
(i)	BSET s	0 ≤ s ≤ 7	PC ← PC + 1

16 bit Opcode:

1001	0100	0sss	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I: 1 if s = 7; Unchanged otherwise.

T: 1 if s = 6; Unchanged otherwise.

H: 1 if s = 5; Unchanged otherwise.

S: 1 if s = 4; Unchanged otherwise.

V: 1 if s = 3; Unchanged otherwise.

N: 1 if s = 2; Unchanged otherwise.

Z: 1 if s = 1; Unchanged otherwise.

C: 1 if s = 0; Unchanged otherwise.

Example:

```
bset    6         ; Set T flag
bset    7         ; Enable interrupt
```

Words: 1 (2 bytes)

Cycles: 1

BST - Bit Store from Bit in Register to T Flag in SREG

Description:

Stores bit b from Rd to the T flag in SREG (status register).

Operation:

(i) T ← Rd(b)

	Syntax:	Operands:	Program Counter:
(i)	BST Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16 bit Opcode:

1111	101d	dddd	Xbbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	⇔	-	-	-	-	-	-

T: 0 if bit b in Rd is cleared. Set to 1 otherwise.

Example:

```
                ; Copy bit
    bst   r1,2  ; Store bit 2 of r1 in T flag
    bld   r0,4  ; Load T into bit 4 of r0
```

Words: 1 (2 bytes)

Cycles: 1

CALL - Long Call to a Subroutine

Description:

Calls to a subroutine within the entire program memory. The return address (to the instruction after the CALL) will be stored onto the stack. (See also RCALL).

	Operation:	
(i)	PC ← k	Devices with 16 bits PC, 128K bytes program memory maximum.
(ii)	PC ← k	Devices with 22 bits PC, 8M bytes program memory maximum.

	Syntax:	**Operands:**	**Program Counter:Stack**
(i)	CALL k	$0 \leq k \leq 64K$	PC ← kSTACK ← PC+2 SP ← SP-2, (2 bytes, 16 bits)
(ii)	CALL k	$0 \leq k \leq 4M$	PC ← kSTACK ← PC+2 SP ← SP-3 (3 bytes, 22 bits)

32 bit Opcode:

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        mov     r16,r0          ; Copy r0 to r16
        call    check           ; Call subroutine
        nop                     ; Continue (do nothing)
        ...
check:  cpi     r16,$42         ; Check if r16 has a special value
        breq    error           ; Branch if equal
        ret                     ; Return from subroutine
        ...
error:  rjmp    error           ; Infinite loop
```

Words: 2 (4 bytes)

Cycles: 4

CBI - Clear Bit in I/O Register

Description:

Clears a specified bit in an I/O register. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:

(i) I/O(P,b) ← 0

	Syntax:	Operands:	Program Counter:
(i)	CBI P,b	$0 \leq P \leq 31, 0 \leq b \leq 7$	PC ← PC + 1

16 bit Opcode:

1001	1000	pppp	pbbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
cbi     $12,7           ; Clear bit 7 in Port D
```

Words: 1 (2 bytes)

Cycles: 2

CBR - Clear Bits in Register

Description:

Clears the specified bits in register Rd. Performs the logical AND between the contents of register Rd and the complement of the constant mask K. The result will be placed in register Rd.

Operation:

(i) Rd ← Rd • ($FF - K)

	Syntax:	Operands:	Program Counter:
(i)	CBR Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16 bit Opcode: See ANDI with K complemented.

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
cbr     r16,$F0     ; Clear upper nibble of r16
cbr     r18,1       ; Clear bit 0 in r18
```

Words: 1 (2 bytes)

Cycles: 1

CLC - Clear Carry Flag

Description:

Clears the Carry flag (C) in SREG (status register).

Operation:

(i) C ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLC	None	PC ← PC + 1

16 bit Opcode:

1001	0100	1000	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	0

C: 0
Carry flag cleared

Example:

```
add   r0,r0     ; Add r0 to itself
clc             ; Clear carry flag
```

Words: 1 (2 bytes)

Cycles: 1

CLH - Clear Half Carry Flag

Description:

Clears the Half Carry flag (H) in SREG (status register).

Operation:

(i) H ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLH	None	PC ← PC + 1

16 bit Opcode:

1001	0100	1101	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	0	-	-	-	-	-

H: 0
Half Carry flag cleared

Example:

```
clh          ; Clear the Half Carry flag
```

Words: 1 (2 bytes)

Cycles: 1

CLI - Clear Global Interrupt Flag

Description:

Clears the Global Interrupt flag (I) in SREG (status register).

Operation:

(i) I ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLI	None	PC ← PC + 1

16 bit Opcode:

1001	0100	1111	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
0	-	-	-	-	-	-	-

I: 0
Global Interrupt flag cleared

Example:

```
cli                 ; Disable interrupts
in      r11,$16     ; Read port B
sei                 ; Enable interrupts
```

Words: 1 (2 bytes)
Cycles: 1

CLN - Clear Negative Flag

Description:

Clears the Negative flag (N) in SREG (status register).

Operation:

(i) N ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLN	None	PC ← PC + 1

16 bit Opcode:

1001	0100	1010	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	**0**	-	-

N: 0

Negative flag cleared

Example:

```
add     r2,r3     ; Add r3 to r2
cln               ; Clear negative flag
```

Words: 1 (2 bytes)

Cycles: 1

CLR - Clear Register

Description:

Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear all bits in the register.

Operation:

(i) Rd ← Rd ⊕ Rd

	Syntax:	Operands:	Program Counter:
(i)	CLR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16 bit Opcode: (see EOR Rd,Rd)

0010	01dd	dddd	dddd

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	0	0	0	1	-

S: 0
Cleared

V: 0
Cleared

N: 0
Cleared

Z: 1
Set

R (Result) equals Rd after the operation.

Example:

```
        clr   r18        ; clear r18
loop:   inc   r18        ; increase r18
        ...
        cpi   r18,$50    ; Compare r18 to $50
        brne  loop
```

Words: 1 (2 bytes)
Cycles: 1

CLS - Clear Signed Flag

Description:

Clears the Signed flag (S) in SREG (status register).

Operation:

(i) $S \leftarrow 0$

	Syntax:	Operands:	Program Counter:
(i)	CLS	None	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	0100	1100	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	0	-	-	-	-

S: 0
Signed flag cleared

Example:

```
add   r2,r3      ; Add r3 to r2
cls              ; Clear signed flag
```

Words: 1 (2 bytes)

Cycles: 1

CLT - Clear T Flag

Description:

Clears the T flag in SREG (status register).

Operation:

(i) T ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLT	None	PC ← PC + 1

16 bit Opcode:

1001	0100	1110	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	0	-	-	-	-	-	-

T: 0
T flag cleared

Example:

```
clt             ; Clear T flag
```

Words: 1 (2 bytes)
Cycles: 1

CLV - Clear Overflow Flag

Description:

Clears the Overflow flag (V) in SREG (status register).

Operation:

(i) V ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLV	None	PC ← PC + 1

16 bit Opcode:

1001	0100	1011	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	0	-	-	-

V: 0
Overflow flag cleared

Example:

```
add   r2,r3    ; Add r3 to r2
clv            ; Clear overflow flag
```

Words: 1 (2 bytes)

Cycles: 1

CLZ - Clear Zero Flag

Description:

Clears the Zero flag (Z) in SREG (status register).

Operation:

(i) Z ← 0

	Syntax:	Operands:	Program Counter:
(i)	CLZ	None	PC ← PC + 1

16 bit Opcode:

1001	0100	1001	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	0	-

Z: 0
Zero flag cleared

Example:

```
add   r2,r3   ; Add r3 to r2
clz           ; Clear zero
```

Words: 1 (2 bytes)
Cycles: 1

COM - One's Complement

Description:

This instruction performs a one's complement of register Rd.

Operation:

(i) Rd ← \$FF - Rd

	Syntax:	Operands:	Program Counter:
(i)	COM Rd	$0 \le d \le 31$	PC ← PC + 1

16 bit Opcode:

1001	010d	dddd	0000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	1

S: $N \oplus V$
For signed tests.

V: 0
Cleared.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; Cleared otherwise.

C: 1
Set.

R (Result) equals Rd after the operation.

Example:

```
        com     r4          ; Take one's complement of r4
        breq    zero        ; Branch if zero
        ...
zero:   nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

CP - Compare

Description:

This instruction performs a compare between two registers Rd and Rr. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr

	Syntax:	Operands:	Program Counter:
(i)	CP Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

0001	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{Rd7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $Rd7 \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet Rd7$
Set if the result is \$00; cleared otherwise.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
        cp    r4,r19      ; Compare r4 with r19
        brne  noteq       ; Branch if r4 <> r19
        ...
noteq:  nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

CPC - Compare with Carry

Description:

This instruction performs a compare between two registers Rd and Rr and also takes into account the previous carry. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - Rr - C

	Syntax:	Operands:	Program Counter:
(i)	CPC Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16 bit Opcode:

0000	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
Previous value remains unchanged when the result is zero; cleared otherwise.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
                                    ; Compare r3:r2 with r1:r0
           cp      r2,r0            ; Compare low byte
           cpc     r3,r1            ; Compare high byte
           brne    noteq            ; Branch if not equal
           ...
    noteq: nop                      ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1

CPI - Compare with Immediate

Description:

This instruction performs a compare between register Rd and a constant. The register is not changed. All conditional branches can be used after this instruction.

Operation:

(i) Rd - K

	Syntax:	Operands:	Program Counter:
(i)	CPI Rd,K	$16 \le d \le 31$, $0 \le K \le 255$	$PC \leftarrow PC + 1$

16 bit Opcode:

0011	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: $R7$
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
            cpi     r19,3     ; Compare r19 with 3
            brne    error     ; Branch if r19<>3
            ...
error:      nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

CPSE - Compare Skip if Equal

Description:

This instruction performs a compare between two registers Rd and Rr, and skips the next instruction if Rd = Rr.

Operation:

(i) If Rd = Rr then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	CPSE Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16 bit Opcode:

0001	00rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
inc     r4          ; Increase r4
cpse    r4,r0       ; Compare r4 to r0
neg     r4          ; Only executed if r4<>r0
nop                 ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1

DEC - Decrement

Description:

Subtracts one -1- from the contents of register Rd and places the result in the destination register Rd.

The C flag in SREG is not affected by the operation, thus allowing the DEC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned values, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

(i) $Rd \leftarrow Rd - 1$

	Syntax:	Operands:	Program Counter:
(i)	DEC Rd	$0 \le d \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	010d	dddd	1010

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	-

S: $N \oplus V$
For signed tests.

V: $\overline{R7} \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0$
Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $80 before the operation.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; Cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
        ldi   r17,$10       ; Load constant in r17
loop:   add   r1,r2         ; Add r2 to r1
        dec   r17           ; Decrement r17
        brne  loop          ; Branch if r17<>0
        nop                 ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1

ELPM - Extended Load Program Memory

Description:

Loads one byte pointed to by the (RAMPZ, Z) register into register 0 (R0). This instruction features a 100% space effective constant initialization or constant data fetch. The program memory is organized in 16 bits words and the LSB of the (RAMPZ, Z) (17 bits) pointer selects either low byte (0) or high byte (1). This instruction can address 128K bytes (64K words) of program memory.

	Operation:	**Comment:**
(i)	R0 ← (RAMPZ, Z)	(RAMPZ, Z) points to program memory

	Syntax:	**Operands:**	**Program Counter:**
(i)	ELPM	None	PC ← PC + 1

16 bit Opcode:

1001	0101	1101	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r31           ; Clear Z high byte
ldi   r30,$F0       ; Set Z low byte
elpm                ; Load constant from program
                    ; memory pointed to by Z (r31:r30)
```

Words: 1 (2 bytes)

Cycles: 3

EOR - Exclusive OR

Description:

Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd ⊕ Rr

	Syntax:	Operands:	Program Counter:
(i)	EOR Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16 bit Opcode:

0010	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
eor     r4,r4          ; Clear r4
eor     r0,r22         ; Bitwise exclusive or between r0 and r22
```

Words: 1 (2 bytes)

Cycles: 1

ICALL - Indirect Call to Subroutine

Description:

Indirect call of a subroutine pointed to by the Z (16 bits) pointer register in the register file. The Z pointer register is 16 bits wide and allows call to a subroutine within the current 64K words (128K bytes) section in the program memory space.

Operation:

(i) PC(15-0) ← Z(15 - 0)Devices with 16 bits PC, 128K bytes program memory maximum.

(ii) PC(15-0) ← Z(15 - 0)Devices with 22 bits PC, 8M bytes program memory maximum.
PC(21-16) is unchanged

	Syntax:	Operands:	Program Counter:	Stack
(i)	ICALL	None	See Operation	STACK ← PC+1 SP ← SP-2 (2 bytes, 16 bits)
(ii)	ICALL	None	See Operation	STACK ← PC+1 SP ← SP-3 (3 bytes, 22 bits)

16 bit Opcode:

1001	0101	XXXX	1001

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        mov     r30,r0      ; Set offset to call table
        icall               ; Call routine pointed to by r31:r30
```

Words: 1 (2 bytes)

Cycles: 3

IJMP - Indirect Jump

Description:

Indirect jump to the address pointed to by the Z (16 bits) pointer register in the register file. The Z pointer register is 16 bits wide and allows jump within the current 64K words (128K bytes) section of program memory.

Operation:

(i)	PC ← Z(15 - 0)	Devices with 16 bits PC, 128K bytes program memory maximum.
(ii)	PC(15-0) ← Z(15-0)	Devices with 22 bits PC, 8M bytes program memory maximum.
	PC(21-16) is unchanged	

	Syntax:	Operands:	Program Counter:	Stack
(ii)	IJMP	None	See Operation	Not Affected
(iii)	IJMP	None	See Operation	Not Affected

16 bit Opcode:

1001	0100	XXXX	1001

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
mov     r30,r0    ; Set offset to jump table
ijmp              ; Jump to routine pointed to by r31:r30
```

Words: 1 (2 bytes)

Cycles: 2

IN - Load an I/O Port to Register

Description:

Loads data from the I/O Space (Ports, Timers, Configuration registers etc.) into register Rd in the register file.

Operation:

(i) Rd ← P

	Syntax:	Operands:	Program Counter:
(i)	IN Rd,P	$0 \le d \le 31$, $0 \le P \le 63$	PC ← PC + 1

16 bit Opcode:

1011	0PPd	dddd	PPPP

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        in      r25,$16     ; Read Port B
        cpi     r25,4       ; Compare read value to constant
        breq    exit        ; Branch if r25=4
        ...
exit:   nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1

INC - Increment

Description:

Adds one -1- to the contents of register Rd and places the result in the destination register Rd.

The C flag in SREG is not affected by the operation, thus allowing the INC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned numbers, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

(i) $Rd \leftarrow Rd + 1$

	Syntax:	Operands:	Program Counter:
(i)	INC Rd	$0 \le d \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	010d	dddd	0011

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	-

S: $N \oplus V$

For signed tests.

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $7F before the operation.

N: R7

Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; Cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
        clr     r22         ; clear r22
loop:   inc     r22         ; increment r22
        ...
        cpi     r22,$4F     ; Compare r22 to $4f
        brne    loop        ; Branch if not equal
        nop                 ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1

JMP - Jump

Description:

Jump to an address within the entire 4M (words) program memory. See also RJMP.

	Operation:
(i)	PC ← k

	Syntax:	**Operands:**	**Program Counter:**	**Stack**
(i)	JMP k	$0 \leq k \leq 4M$	PC ← k	Unchanged

32 bit Opcode:

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        mov     r1,r0     ; Copy r0 to r1
        jmp     farplc    ; Unconditional jump
        ...
farplc: nop               ; Jump destination (do nothing)
```

Words: 2 (4 bytes)

Cycles: 3

LD - Load Indirect from SRAM to Register using Index X

Description:

Loads one byte indirect from SRAM, I/O location or register file to register. This memory location is pointed to by the X (16 bits) pointer register in the register file. Memory access is limited to the current SRAM, I/O location or register file page of 64K bytes.

The X pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the X pointer register.

The results loaded by the following instructions are undefined.

ld XL, X+   ld XH, X+   ld XL, -X   ld XH, -X

Using the X pointer:

	Operation:		Comment:
(i)	Rd ← (X)		X: Unchanged
(ii)	Rd ← (X)	X ← X + 1	X: Post incremented
(iii)	X ← X - 1	Rd ← (X)	X: Pre decremented

	Syntax:	Operands:	Program Counter:
(i)	LD Rd, X	$0 \le d \le 31$	PC ← PC + 1
(ii)	LD Rd, X+	$0 \le d \le 31$	PC ← PC + 1
(iii)	LD Rd,-X	$0 \le d \le 31$	PC ← PC + 1

16 bit Opcode :

(i)	1001	000d	dddd	1100
(ii)	1001	000d	dddd	1101
(iii)	1001	000d	dddd	1110

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr  r27          ; Clear X high byte
ldi  r26,$1F      ; Set X low byte to $1F
ld   r0,X+        ; Load r0 with memory loc. $1F-R31(X post inc)
ld   r1,X         ; Load r1 with memory loc. $20-I/O loc. $00
ldi  r26,$60      ; Set X low byte to $60
ld   r2,X         ; Load r2 with memory loc. $60-SRAM loc. $60
ld   r3,-X        ; Load r3 with memory loc. $5F-I/O loc. $3F(X pre dec)
```

Words: 1 (2 bytes)

Cycles: 2

LD (LDD) - Load Indirect from SRAM to Register using Index Y

Description:

Loads one byte indirect with or without displacement from SRAM, I/O location or register file to register. This memory location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current SRAM, I/O location or register file page of 64K bytes.

The Y pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the Y pointer register.

The results loaded by the following instructions are undefined.

ld YL, Y+   ld YH, Y+   ld YL, -Y   ld YH, -Y

Using the Y pointer:

	Operation:		**Comment:**
(i)	Rd ← (Y)		Y: Unchanged
(ii)	Rd ← (Y)	Y ← Y + 1	Y: Post incremented
(iii)	Y ← Y - 1	Rd ← (Y)	Y: Pre decremented
(iiii)	Rd ← (Y+q)		Y: Unchanged, q: Displacement

	Syntax:	**Operands:**	**Program Counter:**
(i)	LD Rd, Y	$0 \le d \le 31$	PC ← PC + 1
(ii)	LD Rd, Y+	$0 \le d \le 31$	PC ← PC + 1
(iii)	LD Rd,-Y	$0 \le d \le 31$	PC ← PC + 1
(iiii)	LDD Rd, Y+q	$0 \le d \le 31$, $0 \le q \le 63$	PC ← PC + 1

16 bit Opcode :

(i)	1000	000d	dddd	1000
(ii)	1001	000d	dddd	1001
(iii)	1001	000d	dddd	1010
(iiii)	10q0	qq0d	dddd	1qqq

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r29          ; Clear Y high byte
ldi   r28,$1F      ; Set Y low byte to $1F
ld    r0,Y+        ; Load r0 with memory loc. $1F-R31(Y post inc)
ld    r1,Y         ; Load r1 with memory loc. $20-I/O loc. $00
ldi   r28,$60      ; Set Y low byte to $60
ld    r2,Y         ; Load r2 with memory loc. $60-SRAM loc. $60
ld    r3,-Y        ; Load r3 with memory loc. $5F-I/O loc. $3F(Y pre dec)
ldd   r4,Y+2       ; Load r4 with memory loc. $61-SRAM loc. $61
```

Words: 1 (2 bytes)

Cycles: 2

LD (LDD) - Load Indirect From SRAM to Register using Index Z

Description:

Loads one byte indirectly with or without displacement from SRAM, I/O location or register file to register. This memory location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current SRAM, I/O location or register file page of 64K bytes.

The Z pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for stack pointer usage of the Z pointer register, however because the Z pointer register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y pointer as a dedicated stack pointer.

For using the Z pointer for table lookup in program memory see the LPM and ELPM instructions.

The results loaded by the following instructions are undefined.

ld ZL, Z+   ld ZH, Z+   ld ZL, -Z   ld ZH, -Z

Using the Z pointer:

	Operation:	Comment:	
(i)	Rd ← (Z)		Z: Unchanged
(ii)	Rd ← (Z)	Z ← Z + 1	Z: Post increment
(iii)	Z ← Z -1	Rd ← (Z)	Z: Pre decrement
(iiii)	Rd ← (Z+q)		Z: Unchanged, q: Displacement

	Syntax:	Operands:	Program Counter:
(i)	LD Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(ii)	LD Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LD Rd,-Z	0 ≤ d ≤ 31	PC ← PC + 1
(iiii)	LDD Rd, Z+q	0 ≤ d ≤ 31, 0 ≤ q ≤ 63	PC ← PC + 1

16 bit Opcode :

(i)	1000	000d	dddd	0000
(ii)	1001	000d	dddd	0001
(iii)	1001	000d	dddd	0010
(iiii)	10q0	qq0d	dddd	0qqq

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r29          ; Clear Z high byte
ldi   r28,$10      ; Set Z low byte to $10
ld    r0,Z+        ; Load r0 with memory loc. $10-R16(Z post inc)
ld    r1,Z         ; Load r1 with memory loc. $11-R17
ldi   r28,$60      ; Set Z low byte to $60
ld    r2,Z         ; Load r2 with memory loc. $60-SRAM loc. $60
ld    r3,-Z        ; Load r3 with memory loc. $5F-I/O loc. $3F(Z pre dec)
ldd   r4,Z+2       ; Load r4 with memory loc. $61-SRAM loc. $61
```

Words: 1 (2 bytes)

Cycles: 2

LDI - Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	$16 \leq d \leq 31$, $0 \leq K \leq 255$	PC ← PC + 1

16 bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r31          ; Clear Z high byte
ldi   r30,$F0      ; Set Z low byte to $F0
lpm                ; Load constant from program
                   ; memory pointed to by Z
```

Words: 1 (2 bytes)

Cycles: 1

LDS - Load Direct from SRAM

Description:

Loads one byte from the SRAM to a Register. A 16-bit address must be supplied. Memory access is limited to the current SRAM Page of 64K bytes. The LDS instruction uses the RAMPZ register to access memory above 64K bytes.

Operation:

(i) Rd ← (k)

	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	$0 \leq d \leq 31$, $0 \leq k \leq 65535$	PC ← PC + 2

32 bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
lds   r2,$FF00     ; Load r2 with the contents of SRAM location $FF00
add   r2,r1        ; add r1 to r2
sts   $FF00,r2     ; Write back
```

Words: 2 (4 bytes)

Cycles: 3

LPM - Load Program Memory

Description:

Loads one byte pointed to by the Z register into register 0 (R0). This instruction features a 100% space effective constant initialization or constant data fetch. The program memory is organized in 16 bits words and the LSB of the Z (16 bits) pointer selects either low byte (0) or high byte (1). This instruction can address the first 64K bytes (32K words) of program memory.

	Operation:		**Comment:**
(i)	R0 ← (Z)		Z points to program memory
	Syntax:	**Operands:**	**Program Counter:**
(i)	LPM	None	PC ← PC + 1

16 bit Opcode:

1001	0101	1100	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r31            ; Clear Z high byte
ldi   r30,$F0        ; Set Z low byte
lpm                  ; Load constant from program
                     ; memory pointed to by Z (r31:r30)
```

Words: 1 (2 bytes)

Cycles: 3

LSL - Logical Shift Left

Description:

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C flag of the SREG. This operation effectively multiplies an unsigned value by two.

Operation:

(i)

←

C ← b7 - - - - - - - - - - - - - - - - - - b0 ← 0

	Syntax:	Operands:	Program Counter:
(i)	LSL Rd	$0 \leq d \leq 31$	PC ← PC + 1

16 bit Opcode: (see ADD Rd,Rd)

0000	11dd	dddd	dddd

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: $N \oplus V$, For signed tests.

V: $N \oplus C$ (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add     r0,r4       ; Add r4 to r0
lsl     r0          ; Multiply r0 by 2
```

Words: 1 (2 bytes)

Cycles: 1

LSR - Logical Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C flag of the SREG. This operation effectively divides an unsigned value by two. The C flag can be used to round the result.

Operation:

$\rightarrow$

0 $\rightarrow$ [b7 - - - - - - - - - - - - - - - - - - b0] $\rightarrow$ [C]

	Syntax:	Operands:	Program Counter:
(i)	LSR Rd	$0 \le d \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	010d	dddd	0110

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	0	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $N \oplus C$ (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).

N: 0

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add     r0,r4       ; Add r4 to r0
lsr     r0          ; Divide r0 by 2
```

Words: 1 (2 bytes)
Cycles: 1

MOV - Copy Register

Description:

This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the destination register Rd is loaded with a copy of Rr.

Operation:

(i) Rd ← Rr

	Syntax:	Operands:	Program Counter:
(i)	MOV Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16 bit Opcode:

0010	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        mov     r16,r0   ; Copy r0 to r16
        call    check    ; Call subroutine
        ...
check:  cpi     r16,$11  ; Compare r16 to $11
        ...
        ret              ; Return from subroutine
```

Words: 1 (2 bytes)

Cycles: 1

MUL - Multiply

Description:

This instruction performs 8-bit × 8-bit → 16-bit unsigned multiplication.

Rr		Rd		R1	R0
Multiplicand	×	Multiplier	→	Product High	Product Low
8		8		16	

The multiplicand Rr and the multiplier Rd are two registers. The 16-bit product is placed in R1 (high byte) and R0 (low byte). Note that if the multiplicand and the multiplier is selected from R0 or R1 the result will overwrite those after multiplication.

Operation:

(i) R1,R0 ← Rr × Rd

	Syntax:	**Operands:**	**Program Counter:**
(i)	MUL Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16 bit Opcode:

1001	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	⇔

C: R15
Set if bit 15 of the result is set; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
mulr6,r5; Multiply r6 and r5
movr6,r1; Copy result back in r6:r5
movr5,r0; Copy result back in r6:r5
```

Words: 1 (2 bytes)
Cycles: 2

Not available in AVR Embedded Core.

NEG - Two's Complement

Description:

Replaces the contents of register Rd with its two's complement; the value $80 is left unchanged.

Operation:

(i) Rd ← $00 - Rd

	Syntax:	Operands:	Program Counter:
(i)	NEG Rd	0 ≤ d ≤ 31	PC ← PC + 1

16 bit Opcode:

1001	010d	dddd	0001

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $R3 \bullet Rd3$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$
For signed tests.

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. A two's complement overflow will occur if and only if the contents of the Register after operation (Result) is $80.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; Cleared otherwise.

C: $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$
Set if there is a borrow in the implied subtraction from zero; cleared otherwise. The C flag will be set in all cases except when the contents of Register after operation is $00.

R (Result) equals Rd after the operation.

Example:

```
            sub   r11,r0       ; Subtract r0 from r11
            brpl  positive     ; Branch if result positive
            neg   r11          ; Take two's complement of r11
positive:   nop                ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

NOP - No Operation

Description:

This instruction performs a single cycle No Operation.

	Operation:
(i)	No

	Syntax:	**Operands:**	**Program Counter:**
(i)	NOP	None	PC ← PC + 1

16 bit Opcode:

0000	0000	0000	0000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r16       ; Clear r16
ser   r17       ; Set r17
out   $18,r16   ; Write zeros to Port B
nop             ; Wait (do nothing)
out   $18,r17   ; Write ones to Port B
```

Words: 1 (2 bytes)

Cycles: 1

OR - Logical OR

Description:

Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd v Rr

	Syntax:	**Operands:**	**Program Counter:**
(i)	OR Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16 bit Opcode:

0010	10rd	dddd	rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
        or      r15,r16     ; Do bitwise or between registers
        bst     r15,6       ; Store bit 6 of r15 in T flag
        brts    ok          ; Branch if T flag set
        ...
ok:     nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1

ORI - Logical OR with Immediate

Description:

Performs the logical OR between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd v K

	Syntax:	Operands:	Program Counter:
(i)	ORI Rd,K	$16 \le d \le 31, 0 \le K \le 255$	PC ← PC + 1

16 bit Opcode:

0110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
ori     r16,$F0        ; Set high nibble of r16
ori     r17,1          ; Set bit 0 of r17
```

Words: 1 (2 bytes)
Cycles: 1

OUT - Store Register to I/O port

Description:

Stores data from register Rr in the register file to I/O space (Ports, Timers, Configuration registers etc.).

	Operation:
(i)	$P \leftarrow Rr$

	Syntax:	**Operands:**	**Program Counter:**
(i)	OUT P,Rr	$0 \le r \le 31$, $0 \le P \le 63$	$PC \leftarrow PC + 1$

16 bit Opcode:

1011	1PPr	rrrr	PPPP

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r16           ; Clear r16
ser   r17           ; Set r17
out   $18,r16       ; Write zeros to Port B
nop                 ; Wait (do nothing)
out   $18,r17       ; Write ones to Port B
```

Words: 1 (2 bytes)

Cycles: 1

POP - Pop Register from Stack

Description:

This instruction loads register Rd with a byte from the STACK.

Operation:

(i) Rd ← STACK

	Syntax:	Operands:	Program Counter:Stack
(i)	POP Rd	$0 \le d \le 31$	PC ← PC + 1SP ← SP + 1

16 bit Opcode:

1001	000d	dddd	1111

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            call   routine     ; Call subroutine
            ...
routine:    push   r14         ; Save r14 on the stack
            push   r13         ; Save r13 on the stack
            ...
            pop    r13         ; Restore r13
            pop    r14         ; Restore r14
            ret                ; Return from subroutine
```

Words: 1 (2 bytes)

Cycles: 2

PUSH - Push Register on Stack

Description:

This instruction stores the contents of register Rr on the STACK.

Operation:

(i) STACK ← Rr

	Syntax:	**Operands:**	**Program Counter:Stack:**
(i)	PUSH Rr	$0 \le r \le 31$	PC ← PC + 1SP ← SP - 1

16 bit Opcode:

1001	001d	dddd	1111

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            call    routine   ; Call subroutine
            ...
routine:    push    r14       ; Save r14 on the stack
            push    r13       ; Save r13 on the stack
            ...
            pop     r13       ; Restore r13
            pop     r14       ; Restore r14
            ret               ; Return from subroutine
```

Words: 1 (2 bytes)

Cycles: 2

RCALL - Relative Call to Subroutine

Description:

Calls a subroutine within ± 2K words (4K bytes). The return address (the instruction after the RCALL) is stored onto the stack. (See also CALL).

Operation:

(i)	PC ← PC + k + 1	Devices with 16 bits PC, 128K bytes program memory maximum.
(ii)	PC ← PC + k + 1	Devices with 22 bits PC, 8M bytes program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack
(i)	RCALL k	-2K ≤ k ≤ 2K	PC ← PC + k + 1	STACK ← PC+1 SP ← SP-2 (2 bytes, 16 bits)
(ii)	RCALL k	-2K ≤ k ≤ 2K	PC ← PC + k + 1	STACK ← PC+1 SP ← SP-3 (3 bytes, 22 bits)

16 bit Opcode:

1101	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            rcall   routine     ; Call subroutine
            ...
routine:    push    r14         ; Save r14 on the stack
            ...
            pop     r14         ; Restore r14
            ret                 ; Return from subroutine
```

Words: 1 (2 bytes)

Cycles: 3

RET - Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK.

	Operation:	
(i)	PC(15-0) ← STACK	Devices with 16 bits PC, 128K bytes program memory maximum.
(ii)	PC(21-0) ← STACK	Devices with 22 bits PC, 8M bytes program memory maximum.

	Syntax:	**Operands:**	**Program Counter:**	**Stack**
(i)	RET	None	See Operation	SP←SP+2,(2 bytes,16 bits pulled)
(ii)	RET	None	See Operation	SP←SP+3,(3 bytes,22 bits pulled)

16 bit Opcode:

1001	0101	0XX0	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
                call    routine     ; Call subroutine
                ...
routine:        push    r14         ; Save r14 on the stack
                ...
                pop     r14         ; Restore r14
                ret                 ; Return from subroutine
```

Words: 1 (2 bytes)

Cycles: 4

RETI - Return from Interrupt

Description:

Returns from interrupt. The return address is loaded from the STACK and the global interrupt flag is set.

Operation:

(i) PC(15-0) ← STACK Devices with 16 bits PC, 128K bytes program memory maximum.

(ii) PC(21-0) ← STACK Devices with 22 bits PC, 8M bytes program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack
(i)	RETI	None	See Operation	SP ← SP +2 (2 bytes, 16 bits)
(ii)	RETI	None	See Operation	SP ← SP +3 (3 bytes, 22 bits)

16 bit Opcode:

1001	0101	0XX1	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
1	-	-	-	-	-	-	-

I: 1

The I flag is set.

Example:

```
            ...
extint:     push   r0        ; Save r0 on the stack
            ...
            pop    r0        ; Restore r0
            reti             ; Return and enable interrupts
```

Words: 1 (2 bytes)

Cycles: 4

RJMP - Relative Jump

Description:

Relative jump to an address within PC-2K and PC + 2K (words). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location.

Operation:

(i) PC ← PC + k + 1

	Syntax:	Operands:	Program Counter:	Stack
(i)	RJMP k	-2K ≤ k ≤ 2K	PC ← PC + k + 1	Unchanged

16 bit Opcode:

1100	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
        cpi   r16,$42     ; Compare r16 to $42
        brne  error       ; Branch if r16 <> $42
        rjmp  ok          ; Unconditional branch
error:  add   r16,r17     ; Add r17 to r16
        inc   r16         ; Increment r16
ok:     nop               ; Destination for rjmp (do nothing)
```

Words: 1 (2 bytes)

Cycles: 2

ROL - Rotate Left trough Carry

Description:

Shifts all bits in Rd one place to the left. The C flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C flag.

Operation:

←

C ← b7 - - - - - - - - - - - - - - - - - - b0 ← C

	Syntax:	**Operands:**	**Program Counter:**
(i)	ROL Rd	$0 \le d \le 31$	PC ← PC + 1

16 bit Opcode: (see ADC Rd,Rd)

0001	11dd	dddd	dddd

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: $N \oplus V$, For signed tests.

V: $N \oplus C$ (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
          rolr15           ; Rotate left
          brcsoneenc       ; Branch if carry set
          ...
oneenc:   nop              ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

ROR - Rotate Right trough Carry

Description:

Shifts all bits in Rd one place to the right. The C flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C flag.

Operation:

→

C → b7 - - - - - - - - - - - - - - - - - - b0 → C

	Syntax:	**Operands:**	**Program Counter:**
(i)	ROR Rd	$0 \le d \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	010d	dddd	0111

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $N \oplus C$ (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
            rorr15         ; Rotate right
            brcczeroenc    ; Branch if carry cleared
            ...
zeroenc:    nop            ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

SBC - Subtract with Carry

Description:

Subtracts two registers and subtracts with the C flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd - Rr - C

	Syntax:	Operands:	Program Counter:
(i)	SBC Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16 bit Opcode:

0000	10rd	dddd	rrrr

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet R5 \bullet \overline{R4} \bullet \overline{R3} \bullet R2 \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
Previous value remains unchanged when the result is zero; cleared otherwise.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of the Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
                          ; Subtract r1:r0 from r3:r2
     sub    r2,r0         ; Subtract low byte
     sbc    r3,r1         ; Subtract with carry high byte
```

Words: 1 (2 bytes)
Cycles: 1

SBCI - Subtract Immediate with Carry

Description:

Subtracts a constant from a register and subtracts with the C flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd - K - C

	Syntax:	Operands:	Program Counter:
(i)	SBCI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16 bit Opcode:

0100	KKKK	dddd	KKKK

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3}\bullet K3 + K3\bullet R3 + R3 \bullet\overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet\overline{K7}\bullet \overline{R7} +\overline{Rd7} \bullet K7 \bullet R7$
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7}\bullet \overline{R6} \bullet\overline{R5}\bullet \overline{R4}\bullet \overline{R3} \bullet\overline{R2}\bullet \overline{R1}\bullet \overline{R0}\bullet Z$
Previous value remains unchanged when the result is zero; cleared otherwise.

C: $\overline{Rd7} \bullet K7+ K7 \bullet R7 +R7 \bullet\overline{Rd7}$
Set if the absolute value of the constant plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
                            ; Subtract $4F23 from r17:r16
         subi  r16,$23      ; Subtract low byte
         sbci  r17,$4F      ; Subtract with carry high byte
```

Words: 1 (2 bytes)
Cycles: 1

SBI - Set Bit in I/O Register

Description:

Sets a specified bit in an I/O register. This instruction operates on the lower 32 I/O registers - addresses 0-31.

	Operation:
(i)	I/O(P,b) ← 1

	Syntax:	**Operands:**	**Program Counter:**
(i)	SBI P,b	$0 \le P \le 31$, $0 \le b \le 7$	PC ← PC + 1

16 bit Opcode:

1001	1010	pppp	pbbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
out   $1E,r0        ; Write EEPROM address
sbi   $1C,0         ; Set read bit in EECR
in    r1,$1D        ; Read EEPROM data
```

Words: 1 (2 bytes)

Cycles: 2

SBIC - Skip if Bit in I/O Register is Cleared

Description:

This instruction tests a single bit in an I/O register and skips the next instruction if the bit is cleared. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:

(i) If I/O(P,b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBIC P,b	$0 \le P \le 31$, $0 \le b \le 7$	PC ← PC + 1, If condition is false, no skip. PC ← PC + 2, If next instruction is one word. PC ← PC + 3, If next instruction is JMP or CALL

16 bit Opcode:

1001	1001	pppp	pbbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
e2wait:  sbic  $1C,1       ; Skip next inst. if EEWE cleared
         rjmp  e2wait      ; EEPROM write not finished
         nop               ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false (no skip)

2 if condition is true (skip is executed)

SBIS - Skip if Bit in I/O Register is Set

Description:

This instruction tests a single bit in an I/O register and skips the next instruction if the bit is set. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:

(i) If I/O(P,b) = 1 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBIS P,b	$0 \le P \le 31$, $0 \le b \le 7$	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a JMP or a CALL

16 bit Opcode:

1001	1011	pppp	pbbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
waitset: sbis  $10,0          ; Skip next inst. if bit 0 in Port D set
         rjmp  waitset        ; Bit not set
         nop                  ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false (no skip)
2 if condition is true (skip is executed)

SBIW - Subtract Immediate from Word

Description:

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

Operation:

(i) Rdh:Rdl ← Rdh:Rdl - K

	Syntax:	Operands:	Program Counter:
(i)	SBIW Rdl,K	dl ∈ {24,26,28,30}, $0 \le K \le 63$	PC ← PC + 1

16 bit Opcode:

1001	0111	KKdd	KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $Rdh7 \bullet \overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $R15 \bullet \overline{Rdh7}$
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
sbiw   r24,1         ; Subtract 1 from r25:r24
sbiw   r28,63        ; Subtract 63 from the Y pointer(r29:r28)
```

Words: 1 (2 bytes)
Cycles: 2

SBR - Set Bits in Register

Description:

Sets specified bits in register Rd. Performs the logical ORI between the contents of register Rd and a constant mask K and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd \vee K$

	Syntax:	Operands:	Program Counter:
(i)	SBR Rd,K	$16 \le d \le 31, 0 \le K \le 255$	$PC \leftarrow PC + 1$

16 bit Opcode:

0110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
sbr   r16,3          ; Set bits 0 and 1 in r16
sbr   r17,$F0        ; Set 4 MSB in r17
```

Words: 1 (2 bytes)
Cycles: 1

SBRC - Skip if Bit in Register is Cleared

Description:

This instruction tests a single bit in a register and skips the next instruction if the bit is cleared.

Operation:

(i) If Rr(b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBRC Rr,b	$0 \le r \le 31$, $0 \le b \le 7$	PC ← PC + 1, If condition is false, no skip. PC ← PC + 2, If next instruction is one word. PC ← PC + 3, If next instruction is JMP or CALL

16 bit Opcode:

1111	110r	rrrr	Xbbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
sub   r0,r1        ; Subtract r1 from r0
sbrc  r0,7         ; Skip if bit 7 in r0 cleared
sub   r0,r1        ; Only executed if bit 7 in r0 not cleared
nop                ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false (no skip)
2 if condition is true (skip is executed)

SBRS - Skip if Bit in Register is Set

Description:

This instruction tests a single bit in a register and skips the next instruction if the bit is set.

Operation:

(i) If Rr(b) = 1 then PC ← PC + 2 (or 3) else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	SBRS Rr,b	$0 \le r \le 31$, $0 \le b \le 7$	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a JMP or a CALL

16 bit Opcode:

1111	111r	rrrr	Xbbb

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
sub     r0,r1       ; Subtract r1 from r0
sbrs    r0,7        ; Skip if bit 7 in r0 set
neg     r0          ; Only executed if bit 7 in r0 not set
nop                 ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false (no skip)
2 if condition is true (skip is executed)

SEC - Set Carry Flag

Description:

Sets the Carry flag (C) in SREG (status register).

Operation:

(i) C ← 1

	Syntax:	Operands:	Program Counter:
(i)	SEC	None	PC ← PC + 1

16 bit Opcode:

1001	0100	0000	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	1

C: 1
Carry flag set

Example:

```
sec             ; Set carry flag
adc     r0,r1   ; r0=r0+r1+1
```

Words: 1 (2 bytes)
Cycles: 1

SEH - Set Half Carry Flag

Description:

Sets the Half Carry (H) in SREG (status register).

Operation:

(i) H ← 1

	Syntax:	Operands:	Program Counter:
(i)	SEH	None	PC ← PC + 1

16 bit Opcode:

1001	0100	0101	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	1	-	-	-	-	-

H: 1

Half Carry flag set

Example:

```
seh          ; Set Half Carry flag
```

Words: 1 (2 bytes)

Cycles: 1

SEI - Set Global Interrupt Flag

Description:

Sets the Global Interrupt flag (I) in SREG (status register).

Operation:

(i) I ← 1

	Syntax:	Operands:	Program Counter:
(i)	SEI	None	PC ← PC + 1

16 bit Opcode:

1001	0100	0111	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
1	-	-	-	-	-	-	-

I: 1
Global Interrupt flag set

Example:

```
cli                 ; Disable interrupts
in    r13,$16       ; Read Port B
sei                 ; Enable interrupts
```

Words: 1 (2 bytes)

Cycles: 1

SEN - Set Negative Flag

Description:

Sets the Negative flag (N) in SREG (status register).

Operation:

(i) N ← 1

	Syntax:	Operands:	Program Counter:
(i)	SEN	None	PC ← PC + 1

16 bit Opcode:

1001	0100	0010	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	1	-	-

N: 1
Negative flag set

Example:

```
add   r2,r19        ; Add r19 to r2
sen                 ; Set negative flag
```

Words: 1 (2 bytes)

Cycles: 1

SER - Set all bits in Register

Description:

Loads $FF directly to register Rd.

Operation:

(i) Rd ← $FF

	Syntax:	Operands:	Program Counter:
(i)	SER Rd	16 ≤ d ≤ 31	PC ← PC + 1

16 bit Opcode:

1110	1111	dddd	1111

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r16             ; Clear r16
ser   r17             ; Set r17
out   $18,r16         ; Write zeros to Port B
nop                   ; Delay (do nothing)
out   $18,r17         ; Write ones to Port B
```

Words: 1 (2 bytes)

Cycles: 1

SES - Set Signed Flag

Description:

Sets the Signed flag (S) in SREG (status register).

Operation:

(i) S ← 1

	Syntax:	Operands:	Program Counter:
(i)	SES	None	PC ← PC + 1

16 bit Opcode:

1001	0100	0100	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	1	-	-	-	-

S: 1
Signed flag set

Example:

```
add   r2,r19          ; Add r19 to r2
ses                   ; Set negative flag
```

Words: 1 (2 bytes)
Cycles: 1

SET - Set T Flag

Description:

Sets the T flag in SREG (status register).

Operation:

(i) T ← 1

	Syntax:	**Operands:**	**Program Counter:**
(i)	SET	None	PC ← PC + 1

16 bit Opcode:

1001	0100	0110	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	1	-	-	-	-	-	-

T: 1
T flag set

Example:

```
    set              ; Set T flag
```

Words: 1 (2 bytes)

Cycles: 1

SEV - Set Overflow Flag

Description:

Sets the Overflow flag (V) in SREG (status register).

Operation:

(i) $V \leftarrow 1$

	Syntax:	Operands:	Program Counter:
(i)	SEV	None	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	0100	0011	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	1	-	-	-

V: 1
Overflow flag set

Example:

```
add   r2,r19         ; Add r19 to r2
sev                  ; Set overflow flag
```

Words: 1 (2 bytes)

Cycles: 1

SEZ - Set Zero Flag

Description:

Sets the Zero flag (Z) in SREG (status register).

Operation:

(i) $Z \leftarrow 1$

	Syntax:	Operands:	Program Counter:
(i)	SEZ	None	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	0100	0001	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	1	-

Z: 1
Zero flag set

Example:

```
add   r2,r19          ; Add r19 to r2
sez                   ; Set zero flag
```

Words: 1 (2 bytes)
Cycles: 1

SLEEP

Description:

This instruction sets the circuit in sleep mode defined by the MCU control register. When an interrupt wakes up the MCU from a sleep state, the instruction following the SLEEP instruction will be executed before the interrupt handler is executed.

Operation:

Syntax:	**Operands:**	**Program Counter:**
SLEEP	None	$PC \leftarrow PC + 1$

16 bit Opcode:

1001	0101	100X	1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
mov     r0,r11          ; Copy r11 to r0
sleep                   ; Put MCU in sleep mode
```

Words: 1 (2 bytes)

Cycles: 1

ST - Store Indirect From Register to SRAM using Index X

Description:

Stores one byte indirect from Register to SRAM, I/O location or register file. This Memory location is pointed to by the X (16 bits) pointer register in the register file. Memory access is limited to the current SRAM, I/O location or register file Page of 64K bytes.

The X pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for stack pointer usage of the X pointer register.

The results stored by the following instructions are undefined.

st X+, XL   st X+, XH   st -X, XL   st -X, XH

Using the X pointer:

	Operation:		**Comment:**
(i)	(X) ← Rr		X: Unchanged
(ii)	(X) ← Rr	X ← X+1	X: Post incremented
(iii)	X ← X - 1	(X) ← Rr	X: Pre decremented

	Syntax:	**Operands:**	**Program Counter:**
(i)	ST X, Rr	$0 \leq r \leq 31$	PC ← PC + 1
(ii)	ST X+, Rr	$0 \leq r \leq 31$	PC ← PC + 1
(iii)	ST -X, Rr	$0 \leq r \leq 31$	PC ← PC + 1

16 bit Opcode :

(i)	1001	001r	rrrr	1100
(ii)	1001	001r	rrrr	1101
(iii)	1001	001r	rrrr	1110

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr     r27         ; Clear X high byte
ldi     r26,$1F     ; Set X low byte to $1F
st      X+,r0       ; Store r0 in memory loc. $1F-R31(X post inc)
st      X,r1        ; Store r1 in memory loc. $20-I/O loc. $00
ldi     r26,$60     ; Set X low byte to $60
st      X,r2        ; Store r2 in memory loc. $60-SRAM loc. $60
st      -X,r3       ; Store r3 in memory loc. $5F-I/O loc. $3F(X pre dec)
```

Words: 1 (2 bytes)

Cycles: 2

ST (STD) - Store Indirect From Register to SRAM using Index Y

Description:

Stores one byte indirect with or without displacement from Register to SRAM. The SRAM location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current SRAM Page of 64K bytes. To access another SRAM page the RAMPY register in the I/O area has to be changed.

The Y pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for stack pointer usage of the Y pointer register.

The results stored by the following instructions are undefined.

st Y+, YL   st Y+, YH   st -Y, YL   st -Y, YH

Using the Y pointer:

	Operation:		Comment:
(i)	$(Y) \leftarrow Rr$		Y: Unchanged
(ii)	$(Y) \leftarrow Rr$	$Y \leftarrow Y+1$	Y: Post incremented
(iii)	$Y \leftarrow Y - 1$	$(Y) \leftarrow Rr$	Y: Pre decremented
(iiii)	$(Y+q) \leftarrow Rr$		Y: Unchanged, q: Displacement

	Syntax:	Operands:	Program Counter:
(i)	ST Y, Rr	$0 \le r \le 31$	$PC \leftarrow PC + 1$
(ii)	ST Y+, Rr	$0 \le r \le 31$	$PC \leftarrow PC + 1$
(iii)	ST -Y, Rr	$0 \le r \le 31$	$PC \leftarrow PC + 1$
(iiii)	STD Y+q, Rr	$0 \le r \le 31, 0 \le q \le 63$	$PC \leftarrow PC + 1$

16 bit Opcode :

(i)	1000	001r	rrrr	1000
(ii)	1001	001r	rrrr	1001
(iii)	1001	001r	rrrr	1010
(iiii)	10q0	qq1r	rrrr	1qqq

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr    r29          ; Clear Y high byte
ldi    r28,$1F      ; Set Y low byte to $1F
st     Y+,r0        ; Store r0 in memory loc. $1F-R31(Y post inc)
st     Y,r1         ; Store r1 in memory loc. $20-I/O loc. $00
ldi    r28,$60      ; Set Y low byte to $60
st     Y,r2         ; Store r2 in memory loc. $60-SRAM loc. $60
st     -Y,r3        ; Store r3 in memory loc. $5F-I/O loc. $3F(Y pre dec)
std    Y+2,r4       ; Store r4 in memory loc. $61-SRAM loc. $61
```

Words: 1 (2 bytes)

Cycles: 2

ST (STD) - Store Indirect From Register to SRAM using Index Z

Description:

Stores one byte indirect with or without displacement from Register to SRAM. The SRAM location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current SRAM Page of 64K bytes. To access another SRAM page the RAMPZ register in the I/O area has to be changed.

The Z pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are very suited for stack pointer usage of the Z pointer register, but because the Z pointer register can be used for indirect subroutine calls, indirect jumps and table lookup it is often more convenient to use the X or Y pointer as a dedicated stack pointer.

The results stored by the following instructions are undefined.

st Z+, ZL   st Z+, ZH   st -Z, ZL   st -Z, ZH

Using the Z pointer:

	Operation:		Comment:
(i)	(Z) ← Rr		Z: Unchanged
(ii)	(Z) ← Rr	Z ← Z+1	Z: Post incremented
(iii)	Z ← Z - 1	(Z) ← Rr	Z: Pre decremented
(iiii)	(Z+q) ← Rr		Z: Unchanged, q: Displacement

	Syntax:	Operands:	Program Counter:
(i)	ST Z, Rr	$0 \le r \le 31$	PC ← PC + 1
(ii)	ST Z+, Rr	$0 \le r \le 31$	PC ← PC + 1
(iii)	ST -Z, Rr	$0 \le r \le 31$	PC ← PC + 1
(iiii)	STD Z+q, Rr	$0 \le r \le 31$, $0 \le q \le 63$	PC ← PC + 1

16 bit Opcode :

(i)	1000	001r	rrrr	0000
(ii)	1001	001r	rrrr	0001
(iii)	1001	001r	rrrr	0010
(iiii)	10q0	qq1r	rrrr	0qqq

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
clr   r31        ; Clear Z high byte
ldi   r30,$10    ; Set Z low byte to $10
st    Z+,r0      ; Store r0 in memory loc. $10-R16(Z post inc)
st    Z,r1       ; Store r1 in memory loc. $11-R17
ldi   r30,$60    ; Set Z low byte to $60
st    Z,r2       ; Store r2 in memory loc. $60-SRAM loc. $60
st    -Z,r3      ; Store r3 in memory loc. $5F-I/O loc. $3F(Z pre dec)
std   Z+2,r4     ; Store r4 in memory loc. $61-SRAM loc. $61
```

Words: 1 (2 bytes)

Cycles: 2

STS - Store Direct to SRAM

Description:

Stores one byte from a Register to the SRAM. A 16-bit address must be supplied. Memory access is limited to the current SRAM Page of 64K bytes. The SDS instruction uses the RAMPZ register to access memory above 64K bytes.

Operation:

(i) (k) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	STS k,Rr	$0 \le r \le 31$, $0 \le k \le 65535$	PC ← PC + 2

32 bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
lds     r2,$FF00        ; Load r2 with the contents of SRAM location $FF00
add     r2,r1           ; add r1 to r2
sts     $FF00,r2        ; Write back
```

Words: 2 (4 bytes)

Cycles: 3

SUB - Subtract without Carry

Description:

Subtracts two registers and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd - Rr

	Syntax:	**Operands:**	**Program Counter:**
(i)	SUB Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16 bit Opcode:

0001	10rd	dddd	rrrr

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
        sub     r13,r12       ; Subtract r12 from r13
        brne    noteq         ; Branch if r12<>r13
        ...
noteq:  nop                   ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

SUBI - Subtract Immediate

Description:

Subtracts a register and a constant and places the result in the destination register Rd. This instruction is working on Register R16 to R31 and is very well suited for operations on the X, Y and Z pointers.

Operation:

(i) $Rd \leftarrow Rd - K$

	Syntax:	Operands:	Program Counter:
(i)	SUBI Rd,K	$16 \leq d \leq 31, 0 \leq K \leq 255$	$PC \leftarrow PC + 1$

16 bit Opcode:

0101	KKKK	dddd	KKKK

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
Set if there was a borrow from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

C: $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
          subir22,$11      ; Subtract $11 from r22
          brnenoteq        ; Branch if r22<>$11
          ...
 noteq:   nop              ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

SWAP - Swap Nibbles

Description:

Swaps high and low nibbles in a register.

Operation:

(i) R(7-4) ← Rd(3-0), R(3-0) ← Rd(7-4)

	Syntax:	Operands:	Program Counter:
(i)	SWAP Rd	$0 \le d \le 31$	PC ← PC + 1

16 bit Opcode:

1001	010d	dddd	0010

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

R (Result) equals Rd after the operation.

Example:

```
inc     r1          ; Increment r1
swap    r1          ; Swap high and low nibble of r1
inc     r1          ; Increment high nibble of r1
swap    r1          ; Swap back
```

Words: 1 (2 bytes)

Cycles: 1

TST - Test for Zero or Minus

Description:

Tests if a register is zero or negative. Performs a logical AND between a register and itself. The register will remain unchanged.

Operation:

(i) $Rd \leftarrow Rd \bullet Rd$

	Syntax:	Operands:	Program Counter:
(i)	TST Rd	$0 \le d \le 31$	$PC \leftarrow PC + 1$

16 bit Opcode:

0010	00dd	dddd	dddd

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd.

Example:

```
        tst   r0          ; Test r0
        breq  zero        ; Branch if r0=0
        ...
zero:   nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

WDR - Watchdog Reset

Description:

This instruction resets the Watchdog Timer. This instruction must be executed within a limited time given by the WD prescaler. See the Watchdog Timer hardware specification.

Operation:

(i) WD timer restart.

	Syntax:	Operands:	Program Counter:
(i)	WDR	None	PC ← PC + 1

16 bit Opcode:

1001	0101	101X	1000

Status Register and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
wdr        ; Reset watchdog timer
```

Words: 1 (2 bytes)

Cycles: 1

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686677
FAX (44) 1276-686697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road
Tsimshatsui East
Kowloon, Hong Kong
TEL (852) 27219778
FAX (852) 27221369

Japan
Atmel Japan K.K.
Tonetsu Shinkawa Bldg., 9F
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4 42 53 60 00
FAX (33) 4 42 53 60 01

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

Printed on recycled paper.

0890A–02/99/xM