Atmel ATmega328PB

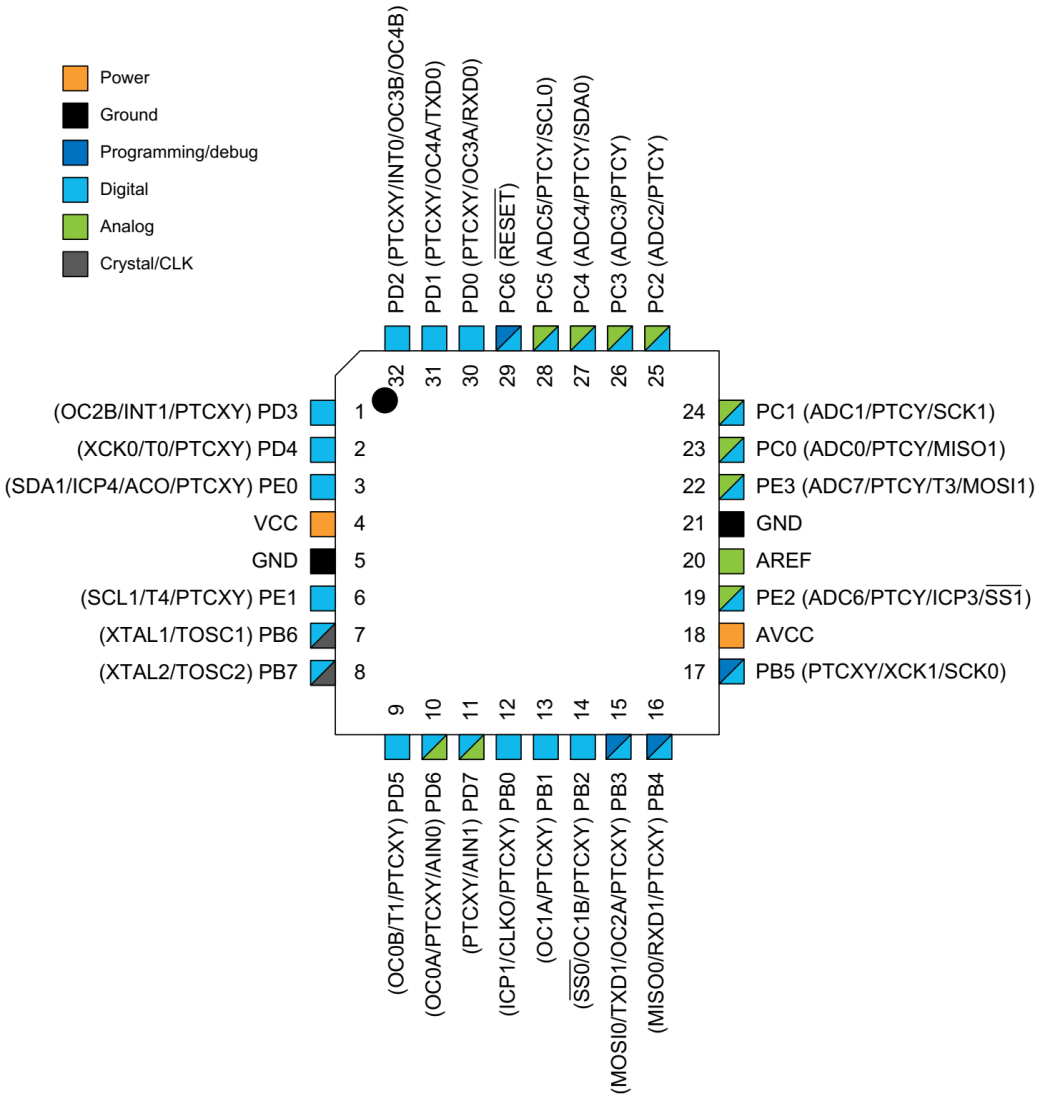
The picoPower® ATmega328PB is a low-power CMOS 8-bit microcontroller based on the AVR® enhancedRISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328PB achieves throughputs close to 1MIPS per MHz. This empowers system designers to optimize the device for power consumption versus processing speed.

Features

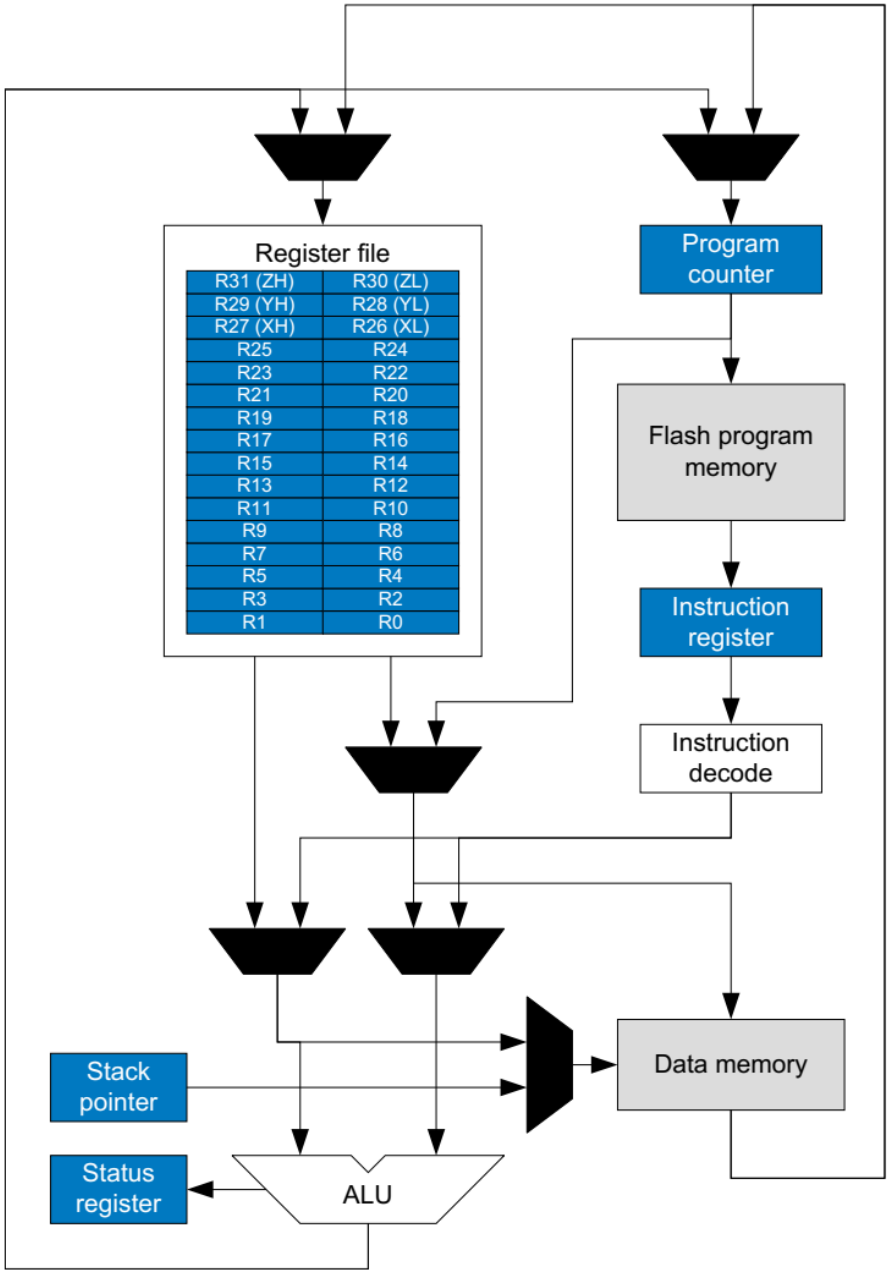
High Performance, Low Power AVR® 8-Bit Microcontroller Family
– 131 Powerful Instructions
– Most Single Clock Cycle Execution
– 32 x 8 General Purpose Working Registers
– Fully Static Operation
– Up to 20 MIPS Throughput at 20MHz
– On-Chip 2-Cycle Multiplier
– 32KBytes of In-System Self-Programmable Flash program memory
– 1KBytes EEPROM
– 2KBytes Internal SRAM
– Programming Lock for Software Security
– Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
– Three 16-bit Timer/Counters with Separate Prescaler, Compare Mode, and Capture Mode
– Real Time Counter with Separate Oscillator
– Ten PWM Channels
– 8-channel 10-bit ADC in TQFP and QFN/MLF package
– Two Programmable Serial USARTs
– Two Master/Slave SPI Serial Interfaces
– Two Byte-Oriented 2-Wire Serial Interfaces (Philips I2C Compatible)
– Programmable Watchdog Timer with Separate On-chip Oscillator
– On-Chip Analog Comparator
– Interrupt and Wake-Up on Pin Change
– Internal 8 MHz Calibrated Oscillator
– External and Internal Interrupt Sources
– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and
– 27 Programmable I/O Lines
– Operating Voltage:1.8 - 5.5V
–Temperature Range:-40°C to 105°C
• Speed Grade:
– 0 - 4MHz @ 1.8 - 5.5V
– 0 - 10MHz @ 2.7 - 5.5.V
– 0 - 20MHz @ 4.5 - 5.5V
• Power Consumption at 1MHz, 1.8V, 25°C
– Active Mode: 0.24mA
– Power-Down Mode: 0.2µA
– Power-Save Mode: 1.3µA (Including 32kHz RTC)

`

Pin Configurations



Block Diagram of the AVR Architecture



I/O Registers

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Status Register(SREG)

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. The Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

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

SREG (Offset: 0x5F)

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – I: Global Interrupt Enable

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The Ibit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Bit 6 – T: Copy Storage

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

Bit 5 – H: Half Carry Flag

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Flag is useful in BCD arithmetic.

Bit 4 – S: Sign Flag, S = N ⊕ V

The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V.

Bit 3 – V: Two's Complement Overflow Flag

The Two's Complement Overflow Flag V supports two's complement arithmetic.

Bit 2 – N: Negative Flag

The Negative Flag N indicates a negative result in an arithmetic or logic operation.

Bit 1 – Z: Zero Flag

The Zero Flag Z indicates a zero result in an arithmetic or logic operation.

Bit 0 – C: Carry Flag

The Carry Flag C indicates a carry in an arithmetic or logic operation.

AVR CPU General Purpose Working Registers

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions. As shown in the figure, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space.

The X-register, Y-register, and Z-register

The registers R26...R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in the figure:

Stack Pointer(SPH:SPL)

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack is implemented as growing from higher to lower memory locations. The Stack Pointer Register always points to the top of the Stack. The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer. The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent.

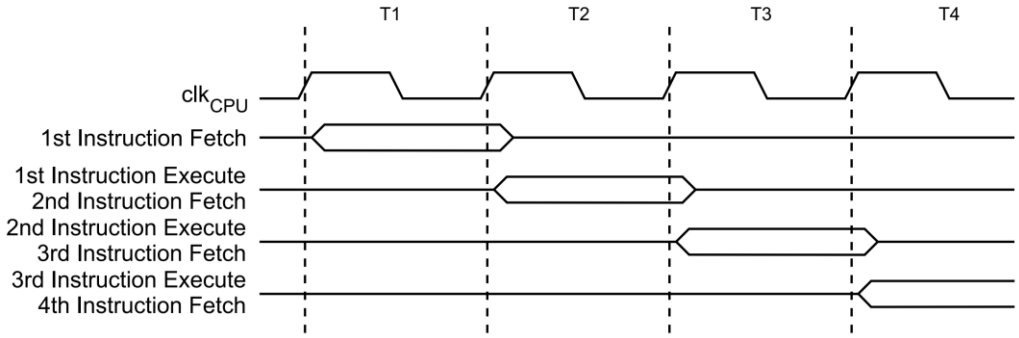
SPH-SPL (Offset: 0x5E-0x5D)

Bit	15	14	13	12	11	10	9	8
					SP11	SP10	SP9	SP8
Access	R	R	R	R	RW	RW	RW	RW
Reset	0	0	0	0	1	0	0	0

Bit	7	6	5	4	3	2	1	0
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	1	1	1	1	1	1	1	1

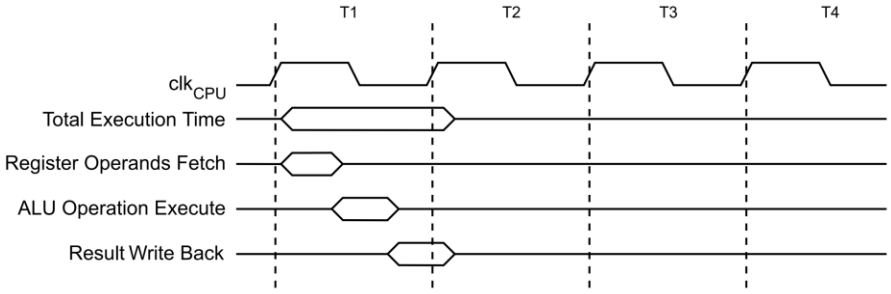
Instruction Execution Timing

he AVR CPU is driven by the CPU clock clkCPU, directly generated from the selected clock source for the chip. No internal clock division is used. The Figure below shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.



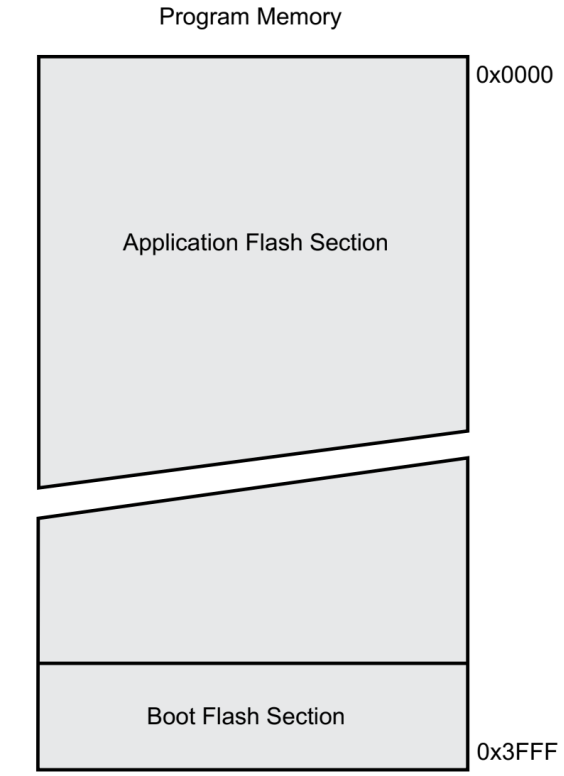
Single Cycle ALU Operation

In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

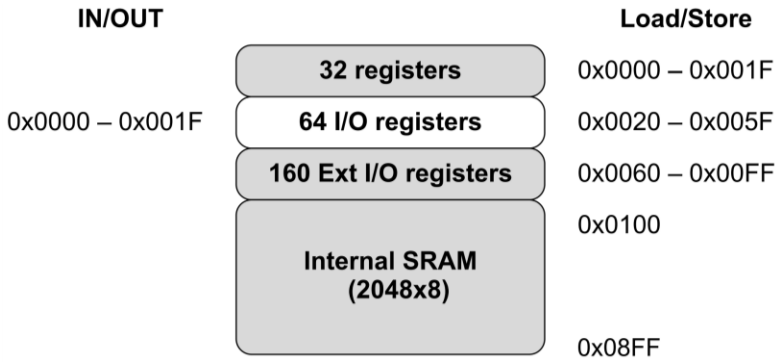


In-System Reprogrammable Flash Program Memory

The ATmega328PB contains 32Kbytes on-chip in-system reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 16K x 16. The ATmega328PB Program Counter (PC) is 14 bits wide, thus addressing the 16K program memory locations. Constant tables can be allocated within the entire program memory address space, using the Load Program Memory (LPM) instruction.



SRAM Data Memory

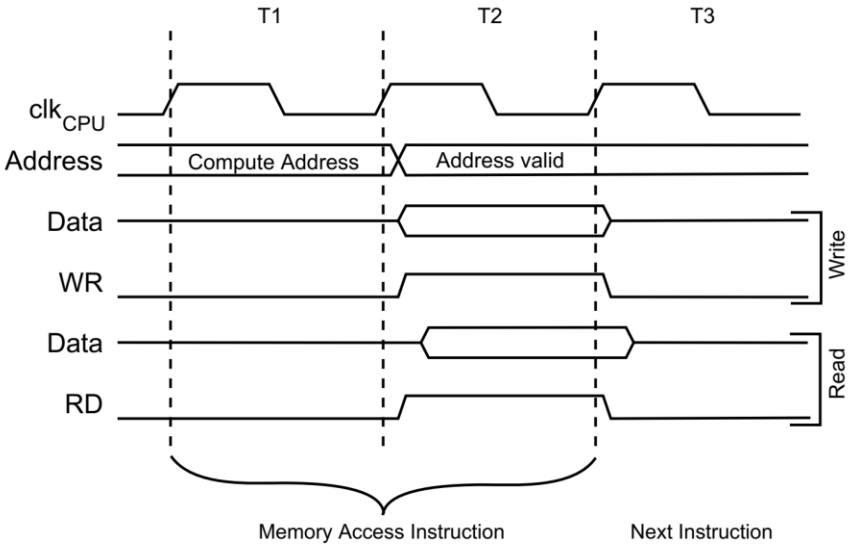


The data memory locations address both the Register File, the I/O memory, Extended I/O memory, and the internal data SRAM.
The first 32 locations address the Register File, the next 64 location the standard I/O memory, then 160 locations of Extended I/O memory, and the next 2K locations address the internal data SRAM.

The five different addressing modes for the data memory cover:
• Direct – The direct addressing reaches the entire data space.
• Indirect with Displacement – The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.
• Indirect – In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.
• Indirect with Pre-decrement – The address registers X, Y, and Z are decremented.
• Indirect with Post-increment – The address registers X, Y, and Z are incremented.

The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 2K bytes of internal data SRAM in the device are all accessible through all these addressing modes.

The internal data SRAM access is performed in two clkCPU cycles as described in the following Figure:



Reset and Interrupt Handling

During reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be a Relative Jump instruction (RJMP) to the reset handling routine.

The ATmega328PB provides several different interrupt sources. Activating an interrupt source forces the microcontroller to immediately stop the current task and execute the code contained in predetermined address, called the interrupt vector. A routine is usually attached at this point interrupt service (different for each application). After the end of the interrupt service routine, the microcontroller resumes the work it interrupted, returning to the exact point where it was interrupted.

All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. They have determined priority levels: The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts:

- The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.
- The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered. When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles the program vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set

The Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

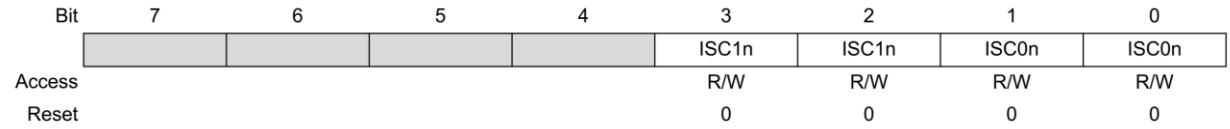
When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction.

Reset and Interrupt Vectors in ATmega328PB

Vector No	Program Address	Source	Interrupts definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Coutner0 Compare Match B
17	0x0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0x0022	SPI0 STC	SPI1 Serial Transfer Complete
19	0x0024	USART0_RX	USART0 Rx Complete
20	0x0026	USART0_UDRE	USART0, Data Register Empty
21	0x0028	USART0_TX	USART0, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface (I^2C
26	0x0032	SPM READY	Store Program Memory Ready
27	0x0034	USART0_START	USART0 Start frame detection
28	0x0036	PCINT3	Pin Change Interrupt Request 3
29	0x0038	USART1_RX	USART0 Rx Complete
30	0x003A	USART1_UDRE	USART0, Data Register Empty
31	0x003C	USART1_TX	USART0, Tx Complete
32	0x003E	USART1_START	USART1 Start frame detection
33	0x0040	TIMER3_CAPT	Timer/Counter3 Capture Event
34	0x0042	TIMER3_COMPA	Timer/Counter3 Compare Match A
35	0x0044	TIMER3_COMPB	Timer/Coutner3 Compare Match B
36	0x0046	TIMER3_OVF	Timer/Counter3 Overflow
37	0x0048	CFD	Clock failure detection interrrupt
38	0x004A	PTC_EOC	PTC End of Conversion
39	0x004C	PTC_WCOMP	PTC Window comparator mode
40	0x004E	SPI1_STC	SPI1 Serial Transfer Complete
41	0x0050	TWI1	TWI1 Transfer complete
42	0x0052	TIMER4_CAPT	Timer/Counter3 Capture Event
43	0x0054	TIMER4_COMPA	Timer/Counter3 Compare Match A
44	0x0056	TIMER4_COMPB	Timer/Coutner3 Compare Match B
45	0x0058	TIMER4_OVF	Timer/Counter3 Overflow

External interrupts on ATmega328PB

The ATmega328PB is equipped with interrupt inputs for immediate response to external conditions.



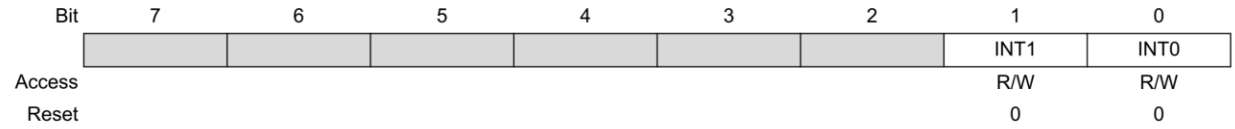
For the operation of the interrupt system of each Microcontroller, it is first necessary to activate it flags and options, which determine the exact mode of operation.

In the AVR ATmega328PB Microcontroller, the selection of the activation level of the external interrupts is done via the EICRA register (offset 0x69), writing appropriate values to the lower four bits according to the tables below:

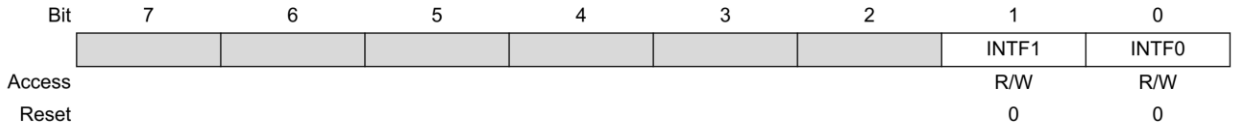
ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

External Interrupt Mask Register (EIMSK, offset = 0x3D)
When the INTx bit of EIMSK is set and the I-bit in the Status Register (SREG) is set, the External Interrupt x is enabled.



External Interrupt Flag Register(EIFR offset = 0x3C)
When an edge or logic change on the INTx pin triggers an interrupt request, INTFx will be set. If the I-bit in SREG and the INTx bit in EIMSK are set, the MCU will jump to the corresponding Interrupt Vector.



Register Summary (Page 1)

Offset	Name	Bit Pos.								
0x23	PINB	7:0	PINBn	PINBn	PINBn	PINBn	PINBn	PINBn	PINBn	PINBn
0x24	DDRB	7:0	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
0x25	PORTB	7:0	PORTBn	PORTBn	PORTBn	PORTBn	PORTBn	PORTBn	PORTBn	PORTBn
0x26	PINC	7:0		PINCn	PINCn	PINCn	PINCn	PINCn	PINCn	PINCn
0x27	DDRC	7:0		DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
0x28	PORTC	7:0		PORTCn	PORTCn	PORTCn	PORTCn	PORTCn	PORTCn	PORTCn
0x29	PIND	7:0	PINDn	PINDn	PINDn	PINDn	PINDn	PINDn	PINDn	PINDn
0x2A	DDRD	7:0	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
0x2B	PORTD	7:0	PORTDn	PORTDn	PORTDn	PORTDn	PORTDn	PORTDn	PORTDn	PORTDn
0x2C	PINE	7:0					PINE3	PINE2	PINE1	PINE0
0x2D	DDRE	7:0					DDRE3	DDRE2	DDRE1	DDRE0
0x2E	PORTE	7:0					PORTE3	PORTE2	PORTE1	PORTE0
0x2F ... 0x34	Reserved									
0x35	TIFR0	7:0						OCF0B	OCF0A	TOV0
0x36	TIFR1	7:0			ICF1			OCF1B	OCF1A	TOV1
0x37	TIFR2	7:0						OCF2B	OCF2A	TOV2
0x38	TIFR3	7:0			ICF3			OCF3B	OCF3A	TOV3
0x39	TIFR4	7:0			ICF4			OCF4B	OCF4A	TOV4
0x3A	Reserved									
0x3B	PCIFR	7:0						PCIF2	PCIF1	PCIF0
0x3C	EIFR	7:0							INTF1	INTF0
0x3D	EIMSK	7:0							INT1	INT0
0x3E	GPIOR0	7:0	GPIOR0[7:0]							
0x3F	EECR	7:0			EEPMn	EEPMn	EERIE	EEMPE	EEPE	EERE
0x40	EEDR	7:0	EEDR[7:0]							
0x41	EEAR	7:0	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
0x42		15:8							EEAR9	EEAR8
0x43	GTCCR	7:0	TSM						PSRASY	PSRSYNC
0x44	TCCR0A	7:0	COM0An	COM0An	COM0Bn	COM0Bn			WGM0n	WGM0n
0x45	TCCR0B	7:0	FOC0A	FOC0B			WGM02	CS0[2:0]		
0x46	TCNT0	7:0	TCNT0[7:0]							
0x47	OCR0A	7:0	OCR0A[7:0]							
0x48	OCR0B	7:0	OCR0B[7:0]							
0x49	Reserved									
0x4A	GPIOR1	7:0	GPIOR1[7:0]							
0x4B	GPIOR2	7:0	GPIOR2[7:0]							
0x4C	SPCR0	7:0	SPIE0	SPE0	DORD0	MSTR0	CPOL0	CPHA0	SPR0n	SPR0n
0x4D	SPSR0	7:0	SPIF0	WCOL0						SPI2X0
0x4E	SPDR0	7:0	SPID[7:0]							
0x4F	Reserved									
0x50	ACSR	7:0	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACISn	ACISn
0x51	DWDR	7:0	DWDR[7:0]							

Register Summary (Page 2)

Offset	Name	Bit Pos.								
0x52	Reserved									
0x53	SMCR	7:0					SM2	SM1	SM0	SE
0x54	MCUSR	7:0				JTRF	WDRF	BORF	EXTRF	PORF
0x55	MCUCR	7:0	JTD	BODS	BODSE	PUD			IVSEL	IVCE
0x56	Reserved									
0x57	SPMCSR	7:0	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
0x58 ... 0x5C	Reserved									
0x5D	SPL and SPH	7:0	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x5E		15:8					SP11	SP10	SP9	SP8
0x5F	SREG	7:0	I	T	H	S	V	N	Z	C
0x60	WDTCSR	7:0	WDIF	WDIE	WDP[3]	WDCE	WDE	WDP[2:0]		
0x61	CLKPR	7:0	CLKPCE				CLKPSn	CLKPSn	CLKPSn	CLKPSn
0x62	XFDCSR	7:0							XFDIF	XFDIE
0x63	Reserved									
0x64	PRR0	7:0	PRTWI0	PRTIM2	PRTIM0	PRUSART1	PRTIM1	PRSPI0	PRUSART0	PRADC
0x65	Reserved									
0x66	OSCCAL	7:0	CALn	CALn	CALn	CALn	CALn	CALn	CALn	CALn
0x67	Reserved									
0x68	PCICR	7:0						PCIE2	PCIE1	PCIE0
0x69	EICRA	7:0					ISC1n	ISC1n	ISC0n	ISC0n
0x6A	Reserved									
0x6B	PCMSK0	7:0	PCINTn	PCINTn	PCINTn	PCINTn	PCINTn	PCINTn	PCINTn	PCINTn
0x6C	PCMSK1	7:0		PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
0x6D	PCMSK2	7:0	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
0x6E	TIMSK0	7:0						OCIE0B	OCIE0A	TOIE0
0x6F	TIMSK1	7:0			ICIE1			OCIE1B	OCIE1A	TOIE1
0x70	TIMSK2	7:0						OCIE2B	OCIE2A	TOIE2
0x71	TIMSK3	7:0			ICIE3			OCIE3B	OCIE3A	TOIE3
0x72	TIMSK4	7:0			ICIE4			OCIE4B	OCIE4A	TOIE4
0x73 ... 0x77	Reserved									
0x78	ADCL and ADCH	7:0	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
0x79		15:8							ADC9	ADC8
0x7A	ADCSRA	7:0	ADEN	ADSC	ADATE	ADIF	ADIE	ADPSn	ADPSn	ADPSn
0x7B	ADCSRB	7:0		ACME				ADTSn	ADTSn	ADTSn
0x7C	ADMUX	7:0	REFSn	REFSn	ADLAR		MUXn	MUXn	MUXn	MUXn
0x7D	Reserved									
0x7E	DIDR0	7:0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
0x7F	DIDR1	7:0							AIN1D	AIN0D
0x80	TCCR1A	7:0	COM1A1	COM1A0	COM1B1	COM1B0			WGM11	WGM10
0x81	TCCR1B	7:0	ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10
0x82	TCCR1C	7:0								
0x83	Reserved									

Register Summary (Page 3)

Offset	Name	Bit Pos.								
0x84	TCNT1L and	7:0	TCNT1[7:0]							
0x85	TCNT1H	15:8	TCNT1[15:8]							
0x86	ICR1L and ICR1H	7:0	ICR1[7:0]							
0x87		15:8	ICR1[15:8]							
0x88	OCR1AL and	7:0	OCR1A[7:0]							
0x89	OCR1AH	15:8	OCR1A[15:8]							
0x8A	OCR1BL and	7:0	OCR1B[7:0]							
0x8B	OCR1BH	15:8	OCR1B[15:8]							
0x8C ... 0x8F	Reserved									
0x90	TCCR3A	7:0	COM3A1	COM3A0	COM3B1	COM3B0			WGM31	WGM30
0x91	TCCR3B	7:0	ICNC3	ICES3		WGM33	WGM32		CS3[2:0]	
0x92	TCCR3C	7:0								
0x93	Reserved									
0x94	TCNT3L and	7:0	TCNT3[7:0]							
0x95	TCNT3H	15:8	TCNT3[15:8]							
0x96	ICR3L and ICR3H	7:0	ICR3[7:0]							
0x97		15:8	ICR3[15:8]							
0x98	OCR3AL and	7:0	OCR3A[7:0]							
0x99	OCR3AH	15:8	OCR3A[15:8]							
0x9A	OCR3BL and	7:0	OCR3B[7:0]							
0x9B	OCR3BH	15:8	OCR3B[15:8]							
0x9C ... 0x9F	Reserved									
0xA0	TCCR4A	7:0	COM4A1	COM4A0	COM4B1	COM4B0			WGM43	WGM42
0xA1	TCCR4B	7:0	ICNC4	ICES4		WGM43	WGM42		CS4[2:0]	
0xA2	TCCR4C	7:0	FOC4A	FOC4B						
0xA3	Reserved									
0xA4	TCNT4L and	7:0	TCNT4[7:0]							
0xA5	TCNT4H	15:8	TCNT4[15:8]							
0xA6	ICR4L and ICR4H	7:0	ICR4[7:0]							
0xA7		15:8	ICR4[15:8]							
0xA8 ... 0xA9	Reserved									
0xAA	OCR4BL and	7:0	OCR4B[7:0]							
0xAB	OCR4BH	15:8	OCR4B[15:8]							
0xAC	SPCR1	7:0	SPIE1	SPE1	DORD1	MSTR1	CPOL1	CPHA1	SPR1n	SPR1n
0xAD	SPSR1	7:0	SPIF1	WCOL1						SPI2X1
0xAE	SPDR1	7:0	SPID1[7:0]							
0xAF	Reserved									
0xB0	TCCR2A	7:0	COM2An	COM2An	COM2Bn	COM2Bn			WGM2n	WGM2n
0xB1	TCCR2B	7:0	FOC2A	FOC2B			WGM22		CS2[2:0]	
0xB2	TCNT2	7:0	TCNT2[7:0]							
0xB3	OCR2A	7:0	OCR2A[7:0]							

Register Summary (Page 4)

Offset	Name	Bit Pos.								
0xB4	OCR2B	7:0	OCR2B[7:0]							
0xB5	Reserved									
0xB6	ASSR	7:0		EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB
0xB7	Reserved									
0xB8	TWBR0	7:0	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn
0xB9	TWSR0	7:0	TWS7	TWS6	TWS5	TWS4	TWS3		TWPS[1:0]	
0xBA	TWAR0	7:0	TWA[6:0]							TWGCE
0xBB	TWDR0	7:0	TWD[7:0]							
0xBC	TWCR0	7:0	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN		TWIE
0xBD	TWAMR0	7:0	TWAM[6:0]							
0xBE ... 0xBF	Reserved									
0xC0	UCSR0A	7:0	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
0xC1	UCSR0B	7:0	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
0xC2	UCSR0C	7:0	UMSEL[1:0]		UPM[1:0]		USBS	UCSZ1 / UDORD	UCSZ0 / UCPHA	UCPOL
0xC3	Reserved									
0xC4	UBRR0L and	7:0	UBRR[7:0]							
0xC5	UBRR0H	15:8					UBRR[11:8]			
0xC6	UDR0	7:0	TXB / RXB[7:0]							
0xC7	Reserved									
0xC8	UCSR1A	7:0	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
0xC9	UCSR1B	7:0	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
0xCA	UCSR1C	7:0	UMSEL[1:0]		UPM[1:0]		USBS	UCSZ1 / UDORD	UCSZ0 / UCPHA	UCPOL
0xCB	Reserved									
0xCC	UBRR1L and	7:0	UBRR[7:0]							
0xCD	UBRR1H	15:8					UBRR[11:8]			
0xCE	UDR1	7:0	TXB / RXB[7:0]							
0xCF ... 0xD7	Reserved									
0xD8	TWBR1	7:0	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn
0xD9	TWSR1	7:0	TWS7	TWS6	TWS5	TWS4	TWS3		TWPS[1:0]	
0xDA	TWAR1	7:0	TWA[6:0]							TWGCE
0xDB	TWDR1	7:0	TWD[7:0]							
0xDC	TWCR1	7:0	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN		TWIE
0xDD	TWAMR1	7:0	TWAM[6:0]							
0xDE ... 0x09A7	Reserved									
0x09A8	OCR4AL and	7:0	OCR4A[7:0]							
0x09A9	OCR4AH	15:8	OCR4A[15:8]							

Instruction Set Summary

ARITHMETIC AND LOGIC INSTRUCTIONS					
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**
ADD	Rd, Rr	Add two Registers without Carry	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add two Registers with Carry	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 two Registers with Carry	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Constant from Reg with Carry.	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 ← 0xFF - Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 - 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 · (0xFF - 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 ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2

BRANCH INSTRUCTIONS					
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP(1)	k	Direct Jump	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(1)	k	Direct Subroutine Call	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	1/2/3
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	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	A, b	Skip if Bit in I/O Register Cleared	if (I/O(A,b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIS	A, b	Skip if Bit in I/O Register is Set	if (I/O(A,b)=1) PC ← PC + 2 or 3	None	1/2/3
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

BIT AND BIT-TEST INSTRUCTIONS					
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**
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 Two's Complement Overflow.	V ← 1	V	1
CLV		Clear Two's 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

MCU CONTROL INSTRUCTIONS					
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**
NOP		No Operation	No Operation	None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

DATA TRANSFER INSTRUCTIONS					
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1: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-Increment	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Decrement	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-Increment	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	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	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Decrement	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-Increment	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	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	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, A	In from I/O Location	Rd ← I/O (A)	None	1
OUT	A, Rr	Out to I/O Location	I/O (A) ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2