NAME:_____ LECTURER:_____ SECTION _____

Faculty of Electrical Engineering, UTM
SEE 3223 Microprocessor
Test 4

1. Answer all the following questions based on Figure Q1. The system has two switches (SW0 and SW1) connected to PA0 and PA1 and four LEDS connected to PB0, PB1, PB2 and PB3. The system will read SW0 and SW1 and acts accordingly when any of the switch is pressed. SW0 will switch 'ON' all the four LEDs and SW1 will perform the running light. Do not use interrupt when answering this question. Assume subroutine Delay1s is already created.

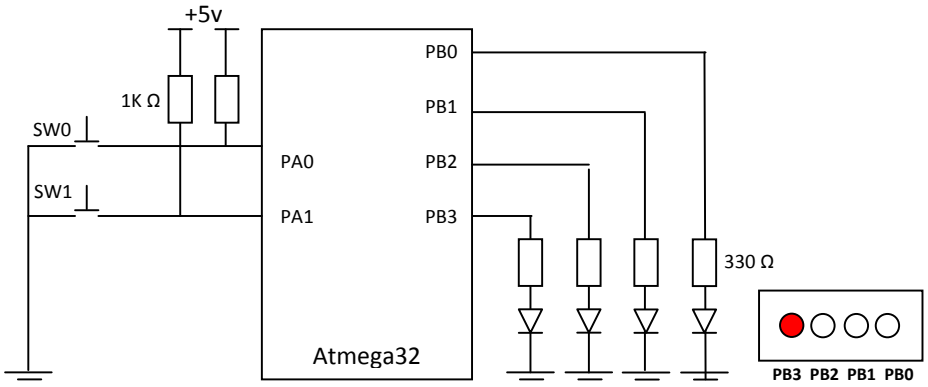


Figure Q1

a. Explain the role of DDRx and PORTx in I/O operation.

DDRx i/o register is used solely for the purpose of making a given port an input or output port.

PORX i/o register is used for either as input or output port

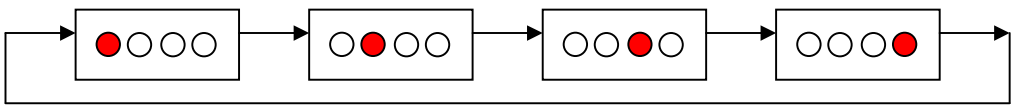
[2 marks]

b. Write a subroutine 'SUB_ON to switch 'on' all the LEDs without disturbing the rest of the bits of the same port.

```
SUB_On:
        ldi     r18,0x0f
        in      r17, portb
        or      r17,r18         ;ON only the lower 4 bits without disturbing the higher 4 bits
        out     portb,R17
        ret
```

[2 marks]

c. Write a subroutine 'SUB_Running_Light' to blink the 4 LEDS in a sequence as shown below. There is a delay of 1 second before the next LED is switched on.



```
SUB_Running_Light:
        ldi             r20,0b00001000
        out     portb,r20
        call    delay1s   ;delay
        ldi             r20,0b00000100
        out     portb,r20
        call    delay1s   ;delay
        ldi             r20,0b00000010
        out     portb,r20
        call    delay1s   ;delay
        ldi             r20,0b00000001
        out     portb,r20
        call    delay1s   ;delay
        ret
```

[2 marks]

d. Write a full assembly code that reads the two switches and execute the selected subroutine above depending on which switch is pressed. Switch SW0 to execute SUB_On and switch SW1 to execute SUB_Running_Light. You do not have to rewrite the subroutines for your answer.

```
.include "M32def.inc"
.org    0
        ldi     r20,high(RAMEND)
        out     sph,r20
        ldi     r20,low(RAMEND)
        out     spl,r20

        ldi     r20,0xff        ;Set Port B as  output
        out     ddrb,r20
        ldi     r20,0x0         ;Set Port A as  input
        out     ddra,r20

loop0:
        sbic    pina,0          ;test bit pa0
        rjmp    loop1
        call    SUB_On
loop1:
        sbic    pina,1          ;test bit pa1
        rjmp    loop0
        call    SUB_Running_Light
        rjmp    loop0
```

[2 marks]

e. Draw the flowchart of the program written in 1(d).

[2 marks]

2. A program (assembly code) is written to generate continuous square wave of a period of 20 μs at PB5 using Interrupt. Assume the clock is 4 *MHz*.

```
.INCLUDE "M32DEF.INC"
                    .ORG    0x0             ; Location for reset
                    JMP     MAIN
                    .ORG    0x14
                    JMP     T0_CM_ISR
                    .ORG    0x100
MAIN:    LDI        R20,    HIGH(RAMEND)
                    OUT     SPH,R20
                    LDI     R20,LOW(RAMEND)
                    OUT     SPL,R20
                    SBI     DDRB,5          ; PB5 as an output

                    LDI     R20, XX
                    OUT     OCR0,R20

                    LDI     R20,0x09
                    OUT     TCCR0,R20

                    LDI     R20,(1<<OCIE0)
                    OUT     TIMSK,R20
                    SEI
HERE:    JMP        HERE

;--------ISR for Timer 0
T0_CM_ISR:
                    ...
                    <Incomplete program>
                    ...
```

a. Which technique, interrupt or polling, **avoids** tying down the microcontroller?

Interrupt.

[1 marks]

b. What is the TIMER0 mode and prescaler of the program above?

TIMER0 mode = CTC (Clear Timer on Compare Match)

Prescale: Clk (no prescale)

[1 marks]

c. Describe the steps in executing an interrupt.

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

1. It finishes the instruction it is currently executing and saves the address of the next instruction (program counter) on the stack.
2. It jumps to a fixed location in memory called the *interrupt vector table*. The interrupt vector table directs the microcontroller to the address of the interrupt service routine (ISR).
3. The microcontroller starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI (return from interrupt).
4. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.

[4 marks]

d. What value should be put in **XX** in order to generate a continuous square wave of a period of 20 µs? Please show all the calculation.

Clock = 4Mhz
T = 1/4Mhz = 0.25 µs

Square wave period of 20µs, means need to create delay of 10µs.

Thus, count XX = (10µs/0.25µs)-1 = **39**

[2 marks]

e. Write the complete assembly code for subroutine ISR_CM_ISR to toggle the output of PB5.

```
ISR_CM_ISR:
        Ldi     r18, 0b00010000   ;for toggling PB5
        in      r17, portb
        eor     r17,r18           ;toggle Port A bit 0,1,2,3
        out     portb,R17
        reti
```

[2 marks]

APPENDIX : RESET & INTERRUPT VECTOR TABLE

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

AVR Assembler Directive

Segment	Directive	Description
Header	.DEVICE	Defines the type of the target processor and the applicable set of instructions (illegal instructions for that type trigger an error message, syntax: .DEVICE AT90S8515)
	.DEF	Defines a synonym for a register (e.g. .DEF MyReg = R16)
	.EQU	Defines a symbol and sets its value (later changes of this value remain possible, syntax: .EQU test = 1234567, internal storage of the value is 4-byte- Integer)
	.SET	Fixes the value of a symbole (later redefinition is not possible)
	.INCLUDE	Includes a file and assembles its content, just like its content would be part of the calling file (typical e.g. including the header file: .INCLUDE "C:\avrtools\appnotes\8515def.inc")
Code	.CSEG	Start of the code segment (all that follows is assembled to the code segment and will go to the program space)
	.DB	Inserts one or more constant bytes in the code segment (could be numbers from 0..255, an ASCII-character like 'c', a string like 'abcde' or a combination like 1,2,3,'abc'. The number of inserted bytes must be even, otherwise an additional zero byte will be inserted by the assembler.)
	.DW	Insert a binary word in the code segment (e.g. produces a table within the code)
	.LISTMAC	Macros will be listed in the .LST-file. (Default is that macros are not listed)
	.MACRO	Beginning of a macro (no code will be produced, call of the macro later produces code, syntax: .MACRO macroname parameters, calling by: macroname parameters)
	.ENDMACRO	End of the macro
EEPROM	.ESEG	Assemble to the EEPROM-segment (the code produced will go to the EEPROM section, the code produces an .EEP-file)
	.DB	Inserts one or more constant bytes in the EEPROM segment (could be numbers from 0..255, an ASCII-character like 'c', a string like 'abcde' or a combination like 1,2,3,'abc'.)
	.DW	Inserts a binary word to the EEPROM segment (the lower byte goes to the next adress, the higher byte follows on the incremented address)

SRAM	.DSEG	Assemble to the data segment (here only BYTE directives and labels are valid, during assembly only the labels are used)
	.BYTE	Reserves one or more bytes space in the data segment (only used to produce correct labels, does not insert any values!)
Everywhere	.ORG	Defines the address within the respective segment, where the assembler assembles to (e.g. .ORG 0x0000)
	.LIST	Switches the listing to the .LST-file on (the assembled code will be listet in a readable text file .LST)
	.NOLIST	Switches the output to the .LST-file off, suppresses listing.
	.INCLUDE	Inserts the content of another source code file, as if its content would be part of the source file (typical e.g. including the header file: .INCLUDE "C:\avrtools\appnotes\8515def.inc")
	.EXIT	End of the assembler-source code (stops the assembling process)

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	10
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	12
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								82
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	47, 67
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	68
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	82, 112, 130
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	83, 112, 130
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	248
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	177
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	32, 66
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	40, 67, 228
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	80
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								82
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								30
	OCDR	On-Chip Debug Register								224
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	56,85,131,198,218
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	107
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	110
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								111
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								111
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								111
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								111
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								111
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								111
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								111
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								111
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	125
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								127
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								127
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	128
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	42
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				164
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	162
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8	19
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								19
$1D ($3D)	EEDR	EEPROM Data Register								19
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	19
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	64
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	64
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	64
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	64
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	64
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	65
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	65
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	65
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	65
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	65
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	65
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	65
$0F ($2F)	SPDR	SPI Data Register								138
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	138
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	136
$0C ($2C)	UDR	USART I/O Data Register								159
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	160
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	161
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								164
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	199
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	214
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	216
$05 ($25)	ADCH	ADC Data Register High Byte								217
$04 ($24)	ADCL	ADC Data Register Low Byte								217
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								179
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	179
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	178
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								177

Complete Instruction Set Summary

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
Arithmetic and Logic Instructions					
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V,S	2 (1)
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V,S	2 (1)
AND	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V,S	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V,S	1
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,S,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FFh - K)	Z,N,V,S	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V,S	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd × Rr (UU)	Z,C	2 (1)
MULS	Rd,Rr	Multiply Signed	R1:R0 ← Rd × Rr (SS)	Z,C	2 (1)
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 ← Rd × Rr (SU)	Z,C	2 (1)
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd × Rr)<<1 (UU)	Z,C	2 (1)
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 ← (Rd × Rr)<<1 (SS)	Z,C	2 (1)
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd × Rr)<<1 (SU)	Z,C	2 (1)
Branch Instructions					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2 (1)

EIJMP		Extended Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	2 (1)
JMP	k	Jump	PC ← k	None	3 (1)
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3 / 4 (4)
ICALL		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	3 / 4 (1)(4)
EICALL		Extended Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	4 (1)(4)
CALL	k	Call Subroutine	PC ← k	None	4 / 5 (1)(4)
RET		Subroutine Return	PC ← STACK	None	4 / 5 (4)
RETI		Interrupt Return	PC ← STACK	I	4 / 5 (4)

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
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,C,N,V,S,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,S,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,S,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 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)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	A, b	Skip if Bit in I/O Register 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, 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
Data Transfer Instructions					
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1 (1)
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LDS	Rd, k	Load Direct from data space	Rd ← (k)	None	2 (1)(4)
LD	Rd, X	Load Indirect	Rd ← (X)	None	2 (2)(4)
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X), X ← X + 1	None	2 (2)(4)
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X)	None	2 (2)(4)
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2 (2)(4)
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2 (2)(4)
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1, Rd ← (Y)	None	2 (2)(4)
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2 (1)(4)
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2 (2)(4)
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2 (2)(4)
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2 (2)(4)
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2 (1)(4)
STS	k, Rr	Store Direct to data space	(k) ← Rd	None	2 (1)(4)
ST	X, Rr	Store Indirect	(X) ← Rr	None	2 (2)(4)

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2 (2)(4)
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2 (2)(4)
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2 (2)(4)
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2 (2)(4)
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2 (2)(4)
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2 (1)(4)
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2 (2)(4)
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr, Z ← Z + 1	None	2 (2)(4)
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1, (Z) ← Rr	None	2 (2)(4)
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2 (1)(4)
LPM		Load Program Memory	R0 ← (Z)	None	3 (3)
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3 (3)
LPM	Rd, Z+	Load Program Memory and Post-Increment	Rd ← (Z), Z ← Z + 1	None	3 (3)
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3 (1)
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3 (1)

ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← (RAMPZ:Z), Z ← Z + 1	None	3 (1)
SPM		Store Program Memory	(Z) ← R1:R0	None	- (1)
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 (1)
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2 (1)
Bit and Bit-test Instructions					
LSL	Rd	Logical Shift Left	Rd(n+1)←Rd(n),Rd(0)←0,C←Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n)←Rd(n+1),Rd(7)←0,C←Rd(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,H	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)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	2
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	2
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

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
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					
BREAK		Break	(See specific descr. for BREAK)	None	1 (1)
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR)	None	1

Notes:
1. This instruction is not available in all devices. Refer to the device specific instruction set summary.
2. Not all variants of this instruction are available in all devices. Refer to the device specific instruction set summary.
3. Not all variants of the LPM instruction are available in all devices. Refer to the device specific instruction set summary. The LPM instruction is not implemented at all in the AT90S1200 device.
4. Cycle times for Data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface. For LD, ST, LDS, STS, PUSH, POP, add one cycle plus one cycle for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI in devices with 16-bit PC, add three cycles plus two cycles for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI in devices with 22-bit PC, add five cycles plus three cycles for each wait state.

Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT(1)	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(1)	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(1)	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(1)	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

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd