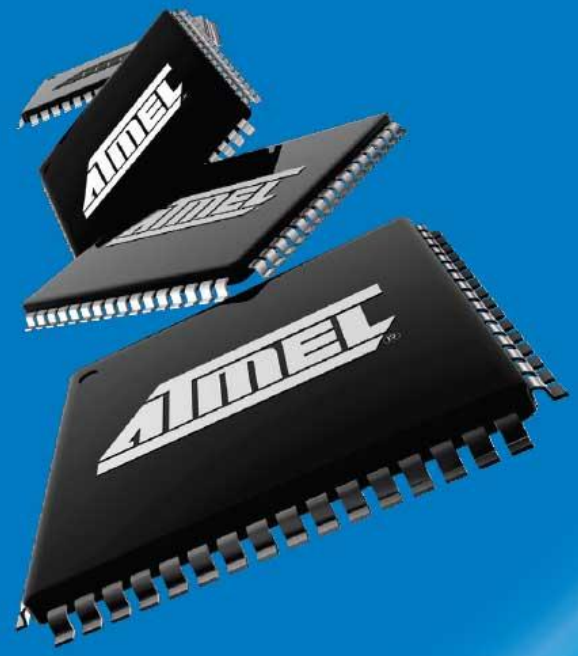
AVR®

8-bit Microcontrollers

AVR®32

32-bit Microcontrollers and Application Processors



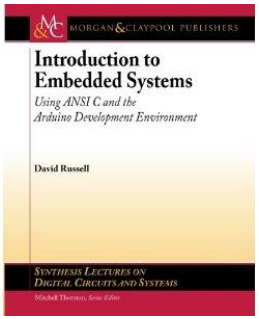
Control Transfer

February 2009

Everywhere You Are®

AVR Control Transfer - Looping

READING



While our textbook covers ANSI C Control Flow in Section 2.5, it does not address control transfer (branching and loopinh) in Assembly. Alternative sources of information that touch on this subject include:

1. Chapter 6 "General-Purpose Input/Output" *except* Section 6.3 Accessing GPIO Line in C Introduction to Embedded Systems: Using ANSI C and the Arduino Development Environment (Synthesis Lectures on Digital) by David Russell and Mitchell Thornton.
2. Introduction to AVR assembler programming for beginners, Controlling sequential execution of the program http://www.avr-asm-tutorial.net/avr_en/beginner/JUMP.html
3. AVR Assembler User Guide http://www.atmel.com/dyn/resources/prod_documents/doc1022.pdf

CONTENTS

Design Objective

When the user presses the button, read first 3 switches (least significant), if the number is less than or equal to 5 then calculate factorial. If greater than 5 turn on decimal point. Display the least significant 4 bits of the answer.

My Design Steps

Step 1: Initialized Ports

```
; Disable interrupts and configure stack pointer for 328P
cli

; Initialize Switches with Pull-up resistors and Test LEDs
in    r16,DDRC       // input Port C Data Direction Register (0x07) for switches 5 to 0
cbr   r16,0b00111111 // define bits 5 to 0 as input (clear bit register)
out   DDRC,r16       // output

in    r16,PORTC      // input Port C Register (0x08) for switches 5 to 0
sbr   r16,0b00111111 // add pull-up resistors (PUR)
out   PORTC,r16      // output

in    r16,DDRD       // input Port D Data Direction Register (0x0A) for switches 7 to 6
cbr   r16,0b11000000 // define bits 7 to 6 as input (clear)
out   DDRD,r16       // output

in    r16,PORTD      // input Port D Register (0x0B) for switches 7 to 6
sbr   r16,0b11000000 // add pull-up resistors (PUR)
out   PORTD,r16      // output

; Initialize SPI Port and Test LEDs
in    r16,DDRB       // Input from Port B Data Direction Register (DDRB) at i/o address 0x04
sbr   r16,0b00101111 // Set PB5, PB3, PB2 (SCK, MOSI, SS) and PB1, PB0 (TEST LEDs) as outputs
out   DDRB,r16       // Output to Port B Data Direction Register (DDRB) at i/o address 0x04

in    r16,PORTB      // input Port B Register (0x05) bit 2 (SS) at i/o address 0x05
cbr   r16,0b00000111 // bit 1 (TEST LED1), bit 0 (TEST LED0)
out   PORTB,r16      // output

ldi   r16,0b01010001 // Set SPCR Enable (SPE) bit 6, Master (MSTR) bit 4,
                     // clock rate fck/16 (SPR1 = 0,SPR0 = 1)
out   SPCR,r16       // Output to SPI Control Register (SPCR) at i/o address 0x2c
```

Step 2: Turned on LED 0 to indicate initialization complete

```
    sbi     PORTB, 0        // Turn on LED 0
```

Step 3: Wrote code to pulse the clock

```
start:
    cbi     PORTD, 5
    sbi     PORTD, 5
```

Step 4: Read in pin waiting for button to be pressed (**Loop Example 1**)

```
// check button
    sbic    PIND, 2
    rjmp    start
```

Step 5: Need to filter out Bounce (**Loop Example 2**)

```
delay_50:
    ldi     r16, 0          // 256
wait:
    dec     r16             //   1 clock cycle
    brne    wait            // + 2 cycle if true, 1 cycles if false
                            //   3 cycles x 256 - 1 = 599 x 1/16 MHz = 48 usec
```

Maximum delay that could be generated was only 48 usec

Step 6: Added a NOP instruction, max delay was now 64 usec

Set delay for nice even number of 50 usec

```
delay_50:
    ldi     r16, 200        // 200 = 0xC8
wait:
    nop                     //   1 clock cycle
    dec     r16             //   1 clock cycle
    brne    wait            // + 2 cycle if true, 1 cycles if true
                            //   4 cycles x 200 - 1 = 799 x 1/16 MHz = 50 usec
```

Step 7: Made an outside loop of 10 (**Loop Example 3**)

```
delay_500:
    ldi     r17, 10
delay_50:
    ldi     r16, 200        // 200 = 0xC8
wait:
    nop                     //   1 clock cycle
    dec     r16             //   1 clock cycle
    brne    wait            // + 2 cycle if true, 1 cycles if true
                            //   4 cycles x 200 - 1 = 799 x 1/16 MHz = 50 usec
    dec     r17
```

```
    brne    delay_50       // 10 x 50 usec = 500 us (approx)
```

Step 8: Converted loop to a subroutine so I could change condition to button release.

```
; -------------------------
Delay500:
    push    r16
    push    r17

    ldi     r17, 10        // was 10
delay_50:
    ldi     r16, 200       // 200 = 0xC8
wait:
    nop                    //   1 clock cycle
    dec     r16            //   1 clock cycle
    brne    wait           // + 2 cycle if true, 1 cycles if true
                           //   4 cycles x 200 - 1 = 799 x 1/16 MHz = 50 usec
    dec     r17
    brne    delay_50       // 10 x 50 usec = 500 us (approx)

    dec     r18
    brne    delay_500      // 10 x 50 usec = 500 us (approx)

    pop     r17
    pop     r16
ret
```

Step 9: Check for button pressed and then released

```
start:
    cbi     PORTD, 5
    sbi     PORTD, 5

// check button down
    sbic    PIND, 2
    rjmp    start

    rcall   Delay500   // remove bounce
check_button:
    cbi     PORTD, 5
    sbi     PORTD, 5

// check button up
    sbis    PIND, 2
    rjmp    check_button
    rcall   Delay500   // remove bounce
```

Step 10: Read Switch and check if less than or equal to 5

```
        in      r16, PINC
        cbr     r16, 0b11110000 // clear undefined bits

        cpi     r16, 6          // no unsigned less than or equal to 5
        brlo    factorial
    // error condition
        ldi     r16, 0x80       // decimal point
        mov     r8, r16
        rcall   writeDisplay
        rjmp    start
```

Step 11: Calculate Factorial (**Loop Example 4**)

```
    factorial:
        ldi     r17, 1
        mov     r0, r17
    calculate:
        mul     r0, r16        // r1:r0 = r0 x r16
        dec     r16
        brne    calculate
```

Step 12: Convert least significant nibble to 7-segment display (Flash Program **Indirect Addressing Mode**)

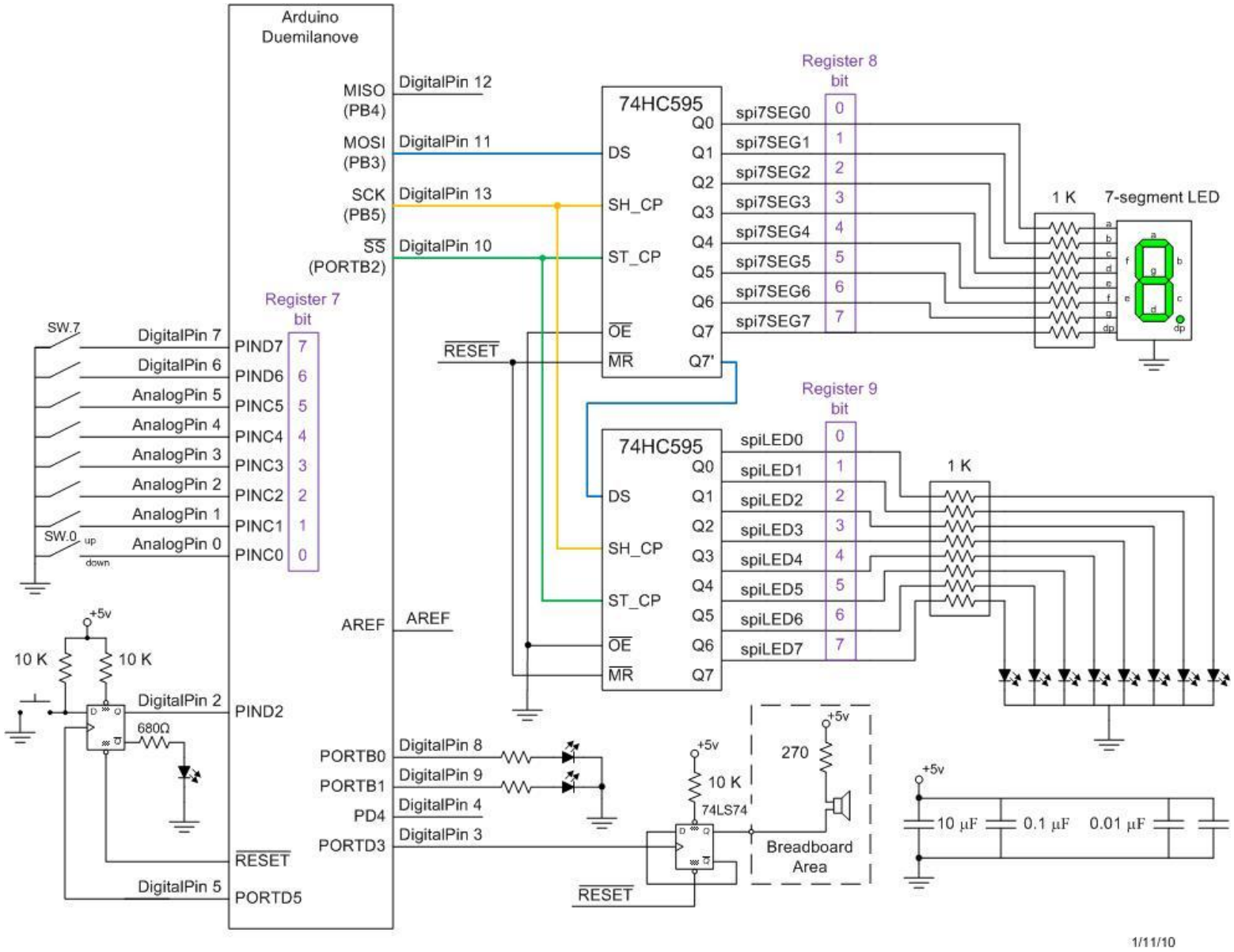
```
    display_answer:
        ldi     r16, 0b00001111     // limit to least significant nibble
        and     r0, r16

        ldi     ZL,low(table<<1)    // load address of look-up
        ldi     ZH,high(table<<1)
        clr     r1
        add     ZL, r0
        adc     ZH, r1
        lpm     spi7SEG, Z

        rcall   writeDisplay
        rjmp    start

    //             gfedcba     gfedcba     gfedcba     gfedcba     gfedcba     gfedcba
    table: .DB  0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110, 0b01101101
    //          0           1           2           3           4           5
           .DB  0b01111101, 0b00000111, 0b01111111, 0b01100111, 0b01110111, 0b01111100
    //          6           7           8           9           A           B
           .DB  0b00111001, 0b01011110, 0b01111001, 0b01110001
    //          C           D           E           F
```

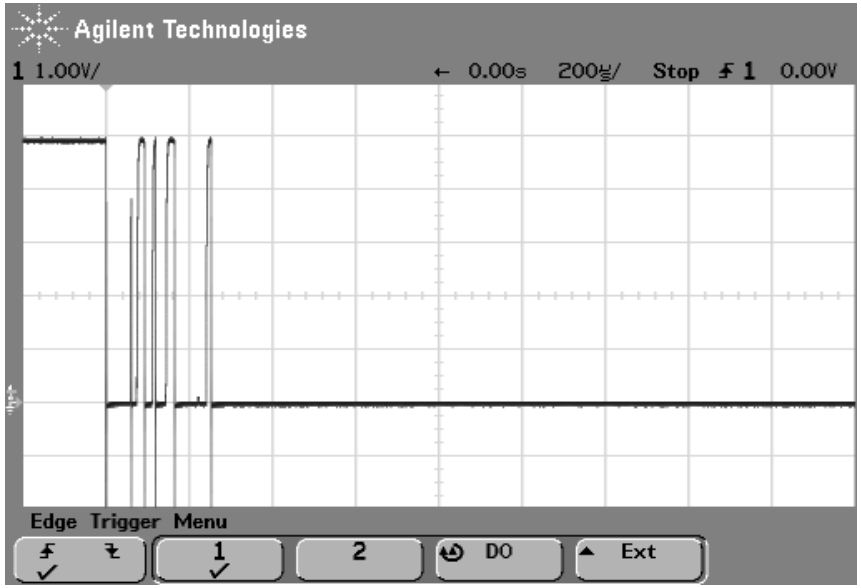
CSULB PROTO-SHIELD SCHEMATIC



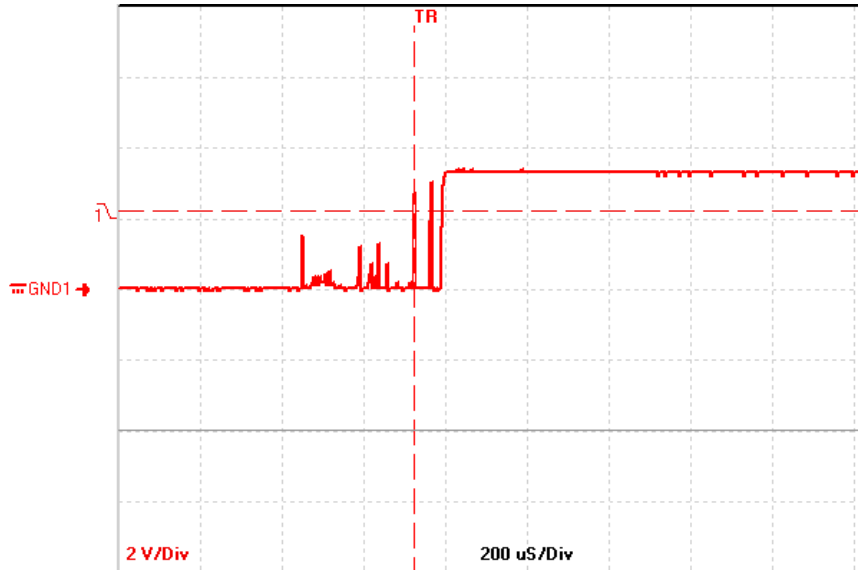
CONFIGURE GPIO PORTS

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Button Debounce



http://myweb.wit.edu/johnsont/Classes/515/Elec515QLab7_Debounce.htm



http://generichid.sourceforge.net/buttonbounceDSO.png

DEBOUNCE

In the capture above (red waveform), the button bounces for about 400us. We will design a delay of approx. 500 us to remove this noise.

Small Loop

```
wait:
     ldi     r16, ____       // Loop Count
delay:
     dec     r16             //   1 clock cycle
     brne    delay           // + 2 cycle if true, 1 cycles if true
```

Tdelay = (Nmc X Lcnt - 1)Tclk

Tdelay = Delay generated by the loop
Tclk = period of one machine cycle = 1/Fclk (note: 1 machine cycle = 1 clock cycle) = 1 / 16 MHz = 0.0625 usec
Nmc = number of machine cycles in 1 loop = 3
Lcnt = number of times loop is run (loop count) = ?

Calculate Maximum Delay

Lcnt = 0 which results in a count of 256
Tmax = (256 x 3 -1)(1/16 MHz) = 48 usec (approx)
Lets increase this delay by adding a nop instruction

Add a NOP

Nmc = number of machine cycles in 1 loop = 4

```
wait:
     ldi     r16, 0xC8       // 200
delay:
     nop                     //   1
     dec     r16             //   1 clock cycle
     brne    delay           // + 2 cycle if true, 1 cycles if true
```

Tmax = (256 x 4 -1)(1/16 MHz) = 64 usec (approx)

DEBOUNCE - CONTINUED -

Solve for Loop Count

Lcnt = (Tdelay/Tclk + 1)/Nmc = (Tdelay x Fclk + 1)/Nmc

Set Lcnt for a delay of 50 usec

Lcnt = (50 x 16 + 1)/4 = 200

Set Outside Loop Count

Create an outside loop with a count of 10 to give us an delay of approximately 500 usec

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

Source: http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf page 10
http://apachepersonal.miun.se/~mathje/ET014G/Lectures/F3-AVR.pdf

ATMEGA328P INSTRUCTION SET[1]

Mnemonics	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 ← 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					
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	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(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

[1] Source: ATmega328P Data Sheet http://www.atmel.com/dyn/resources/prod_documents/8161S.pdf Chapter 31 Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
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					
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
DATA TRANSFER INSTRUCTIONS					
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-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	2
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	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, 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

Mnemonics	Operands	Description	Operation	Flags	#Clocks
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		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

Note: 1. These instructions are only available in ATmega168PA and ATmega328P.