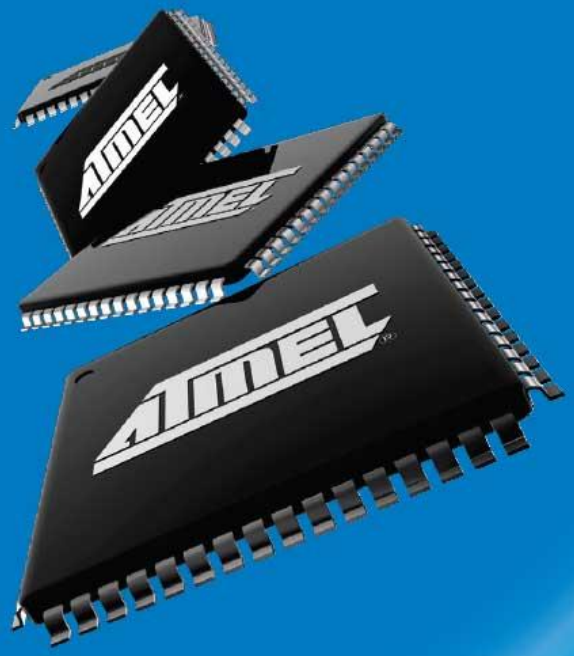
AVR®

8-bit Microcontrollers

AVR®32

32-bit Microcontrollers and Application Processors



Introduction to AVR Assembly Language Programming II

February 2009

Everywhere You Are®

CONTENTS

INSTRUCTION SET ARCHITECTURE (*REVIEW*)

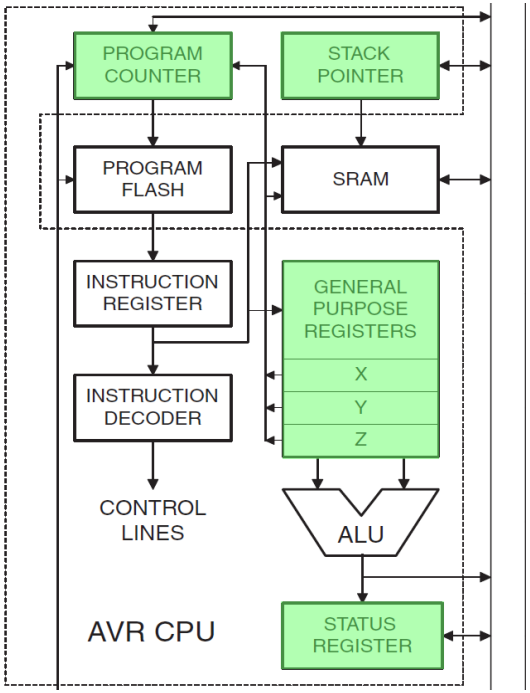
The *Instruction Set Architecture* (ISA) of a microprocessor includes all the registers that are accessible to the programmer. In other words, registers that can be modified by the *instruction set* of the processor. With respect to the AVR CPU illustrated in Figure 2-2, these ISA registers include the 32 x 8-bit general purpose resisters, status resister (SREG), the stack pointer (SP), and the program counter (PC).

Data Transfer instructions are used to load and store data to the General Purpose Registers, also known as the *Register File*. Exceptions are the push and pop instructions which modify the Stack Pointer. By definition these instructions do not modify the status register (SREG).

Arithmetic and Logic Instructions plus Bit and Bit-Test Instructions use the ALU to operate on the data contained in the general purpose registers. Flags contained in the status register (SREG) provide important information concerning the results of these operations. For example, if you are adding two signed numbers together, you will want to know if the answer is correct. The state of the overflow flag (OV) bit within SREG gives you the answer to this question (1 = error, 0 no error).

As the AVR processor fetches and executes instructions it automatically increments the program counter (PC) so it always points at the **next instruction to be executed**. Control Transfer Instructions allow you to change the contents of the PC either conditionally or unconditionally. Continuing our example if an error results from adding two signed numbers together we may want to conditionally (OV = 1) branch to an error handling routine.

Figure 1-5 AVR Central Processing Unit ISA Registers[1]



[1] Source: ATmega16 Data Sheet http://www.atmel.com/dyn/resources/prod_documents/2466s.pdf page 3

INSTRUCTION SET (*REVIEW*)

The ***Instruction Set*** of our AVR processor can be functionally divided (or classified) into the following parts:

- Data Transfer Instructions
- Arithmetic and Logic Instructions
- Bit and Bit-Test Instructions
- Control Transfer (Branch) Instructions
- MCU Control Instructions

SREG – AVR STATUS REGISTER[2] (*REVIEW*)

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	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	

Non ALU

- **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. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the `reti` instruction. The I-bit can also be set and cleared by the application with the `sei` and `cli` instructions.
- **Bit 6 – T: Bit Copy Storage**
  The Bit Copy instructions `bld` (Bit LoaD) and `bst` (Bit STore) use the T-bit as source or destination. A bit from a register can be copied into T ($R_b \rightarrow T$) by the `bst` instruction, and a bit in T can be copied into a bit in a register ($T \rightarrow R_b$) by the `bld` instruction.

ALU

Signed two's complement arithmetic

- **Bit 4 – S: Sign Bit, S = N ⊕ V**
  Bit set if answer is negative with no errors or if both numbers were negative and error occurred, zero otherwise.
- **Bit 3 – V: Two's Complement Overflow Flag**
  Bit set if error occurred as the result of an arithmetic operation, zero otherwise.
- **Bit 2 – N: Negative Flag**
  Bit set if result is negative, zero otherwise.

Unsigned arithmetic

- **Bit 5 – H: Half Carry Flag**
  Carry from least significant nibble to most significant nibble. Half Carry is useful in BCD arithmetic.
- **Bit 0 – C: Carry Flag**
  The Carry Flag C indicates a carry in an arithmetic operation. Bit set if error occurred as the result of an unsigned arithmetic operation, zero otherwise.

Arithmetic and Logical

- **Bit 1 – Z: Zero Flag**
  The Zero Flag Z indicates a zero result in an arithmetic or logic operation.

[2] Source: ATmega328P Data Sheet http://www.atmel.com/dyn/resources/prod_documents/8161S.pdf Section 6.3 Status Register

CONTROL TRANSFER (BRANCH) INSTRUCTIONS

Compare and Test `cp, cpc, cpi, tst, bst`

Unconditional

- Relative (1) `rjmp, rcall`
- Direct `jmp, call`
- Indirect `ijmp, icall`
- Subr. & Inter. Return `ret, reti`

Conditional

- Branch if (2) …
  - SREG Flag bit is clear (`br`*`Flag`*`c`) or set (`br`*`Flag`*`s`) by name (I, **T**, **H**, S, **V**, N, Z, **C**) or bit (`brbc`, `brbs`).
  - These SREG flag bits (I, T, H, **S**, V, **N**, **Z**, **C**) use more descriptive mnemonics.
    - ✓ Branch if equal (`breq`) or not equal (`brne`) test the Z flag.
    - ✓ Unsigned arithmetic branch if plus (`brpl`) or minus (`brmi`) test the N flag, while branch if same or higher (`brsh`) or lower (`brlo`), test the C flag and are equivalent to `brcc` and `brcs` respectively.
    - ✓ Signed 2's complement arithmetic branch if number is less than zero (`brlt`) or greater than or equal to zero (`brge`) test the S flag
- Skip if …
  - Bit (b) in a register is clear (`sbrc`) or set (`sbrs`).
  - Bit (b) in I/O register is clear (`sbic`) or set (`sbis`). Limited to I/O addresses 0-31

Note:

1. Branch relative to PC + ($-2^{k-1} \Rightarrow 2^{k-1}-1$, where k = 12) + 1 ⇨PC-2047 to PC+2048, within 16 K word address space of ATmega328P
2. All branch relative to PC + ($-2^{k-1} \Rightarrow 2^{k-1}-1$, where k = 7) + 1 ⇨PC-64 to PC+63, within 16 K word address space of ATmega328P

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

CONTROL TRANSFER (BRANCH) EXAMPLES

EXAMPLE #1 LAB 4 – DIRECTION FINDER AND TESTING SREG BITS

Design a digital circuit with two (2) switches that will turn on one of the rooms 4 LED segments indicating the direction you want your bear to walk.

			Inputs		Outputs			
Direction	SW1	SW0	dir.1	dir.0	Direction Segment ON = 1, OFF = 0			
					LDg	LDb	LDf	LDa
South	DWN	DWN	0	0	1	0	0	0
East	DWN	UP	0	1	0	1	0	0
West	UP	DWN	1	0	0	0	1	0
North	UP	UP	1	1	0	0	0	1

Table 4.5 Direction to Segment Conversion Table

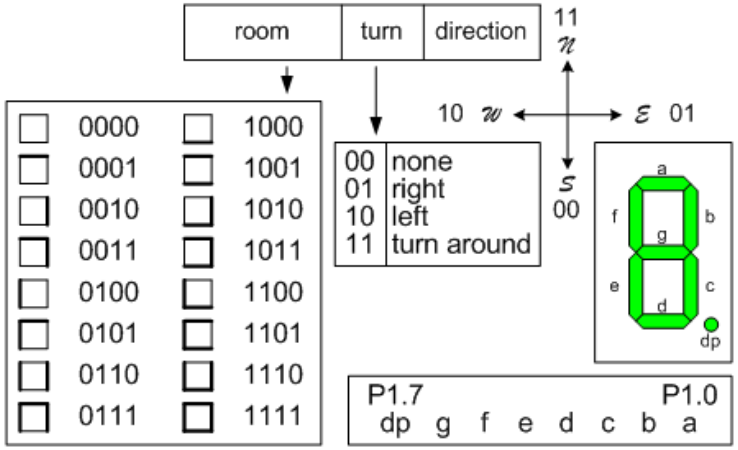


Figure 5.2 Programmer's Reference Card

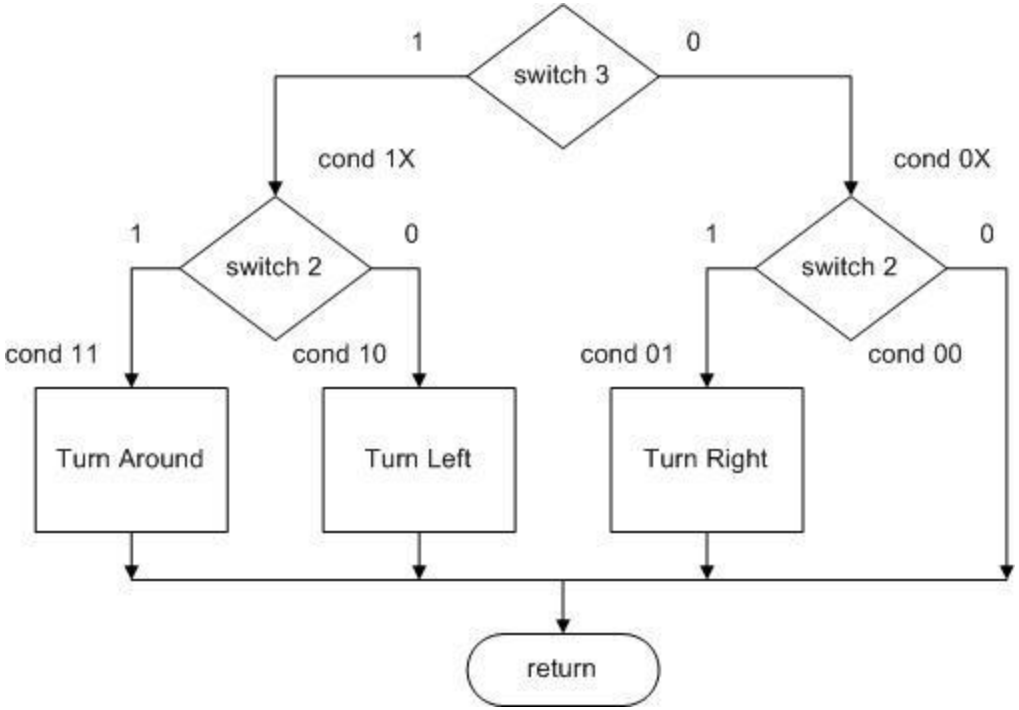
```
; -------------------------
; -- I Know the Way to Go --
; Called from main program
; Input: dir bits 1 and 0 Outputs: spi7SEG register bits seg_g, seg_f, seg_b, seg_a
; No other registers or status register flags are modified by this Subroutine
knowWay:
   push  reg_F
   in    reg_F,SREG
   push  work0
   push  reg_A
   push  reg_B
   lds   work0,dir      // move direction bits into a working register
   // facing east (segment b)
   bst   work0,0       // store r19 bit 0 into T
   bld   reg_B,0       // load r16 bit 0 from T
   bst   work0,1       // store r19 bit 1 into T
   bld   reg_A,0       // load r17 bit 0 from T
   com   reg_A         // B = /A * B
   and   reg_B,reg_A
   bst   reg_B,0       // store r16 bit 0 into T
   bld   spi7SEG,seg_b // load r8 bit 1 from T
   your direction code (circuit schematic) from this lab goes here
   pop   reg_B
   pop   reg_A
   pop   work0
   out   SREG,reg_F
   pop   reg_F
   ret
```

EXAMPLE #2 LAB 5 – WHICHWAY AND TESTING SREG BIT T

Using switches 3 and 2, located on Port C pins 3 and 2 respectively, input an action you want the bear to take. The three possible actions are do nothing, turnLeft, turnRight, and turnAround. Write a subroutine named WhichWay to take the correct action as defined by the following table.

SW.3	SW.2	Action
DWN = 0	DWN = 0	Show direction
DWN = 0	UP = 1	rcall turnRight
UP = 1	DWN = 0	rcall turnLeft
UP = 1	UP = 1	rcall turnAround

Table 5.2 Truth Table of Turn Indicators

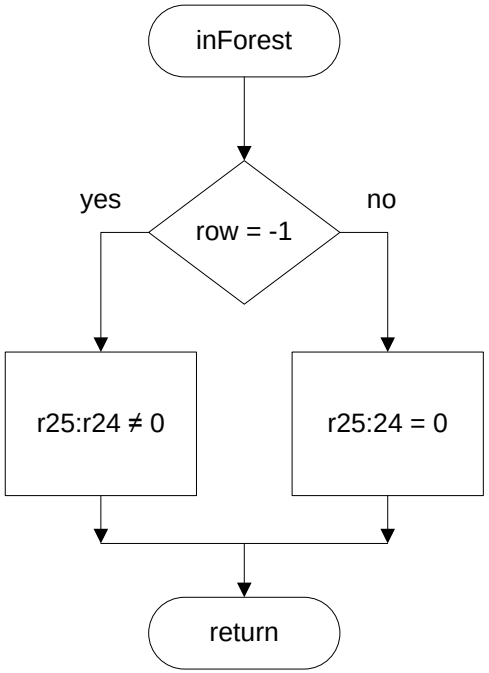


```
; --------------------------
; --- Which Way Do I Go? ---
; Called from main program
; Input: dir.1, dir.0  Outputs: dir.1, dir0
; No registers or flags are modified by this subroutine
whichWay:
   push   reg_F
   in     reg_F,SREG
   push   switch

   in     switch, PINC  // input port C pins (0x06) into register r7
   bst    switch, 3     // store switch bit 3 into T
   brts   cond_1X       // branch if T is set
   bst    switch, 2     // store switch bit 2 into T
   brts   cond_01       // branch if T is set
cond_00:
   rjmp   whichEnd
cond_01:
   rcall  turnRight
   rjmp   whichEnd
cond_1X:
   // branch based on the state of switch bit 2
   :
cond_10:
   :
cond_11:
   :
whichEnd:
   pop    switch
   out    SREG, reg_F
   pop    reg_F
   ret
```

EXAMPLE #3 LAB 5 – INFOREST AND COMPARE IMMEDIATE INSTRUCTION

In this part of the lab you will write the `inForest` routine and learn more about the AVR Studio Simulator/Debugger. The `inForest` subroutine tells us if the bear is in the forest (i.e., has found his way out of the maze). The subroutine accomplishes this by checking the row the bear is currently in. The rows and columns of the maze are numbered from 0 to 19 (13h) starting in the upper left hand corner. When the bear has found his way out of the maze he is in row minus one (-1). The subroutine is to return true (r25:r24 != 0) if the bear is in the forest and false (r25:r24 == 0) otherwise. The register pair r25:r24 is where C++ looks for return values for the BYTE data type.



EXAMPLE #3 LAB 5 – INFOREST AND COMPARE IMMEDIATE INSTRUCTION – CONTINUED –

```
; ---------------------------
; ------- In Forest --------
; Called from whichWay subroutine
; Input: row        Outputs: C++ return register (r24)
; No others registers or flags are modified by this subroutine

inForest:
   push   reg_F          // push any flags or registers modified
   in     reg_F,SREG
   push   work0
   lds    work0,row

   test if bear is in the forest

endForest:
   clr    r25            // zero extend
   pop    work0          // pop any flags or registers placed on the
stack
   out    SREG,reg_F
   pop    reg_F
   ret
```

EXAMPLE #4 LAB 6 – TEST HITWALL AND TST INSTRUCTION

To find out if pseudo-instruction hitwall works, write a subroutine named testHitWall. HitWall returns a non-zero value in register pair r25:24 if the answer is no and zero if the answer is yes. Send the yes/no answer to the question to two of the discrete LEDs on the Arduino Proto-Shield.

```
; -------------------------
; ------ Test hitWall ------
; Called from main program
; Input: none        Outputs: spiLEDs bits 1 and 0
; No other registers or flags are modified by this subroutine

testHitWall:
   push   reg_F
   in     reg_F,SREG
   push   work0

   rcall  hitWall

   mov    work0, spiLEDS
   tst    cppReg
   breq   noWall
   sbr    work0,0b00000010    // immediate instructions must use r16 to r31
   cbr    work0,0b00000001
   rjmp   overTheWall
noWall:
   sbr    work0,0b00000001
   cbr    work0,0b00000010
overTheWall:
   mov    spiLEDS,work0

   pop    work0
   out    SREG,reg_F
   pop    reg_F
   ret
```

EXAMPLE #5 LAB 7 – TEST PAWS

Using the code from the previous example, write a new test code sequence in your whichWay subroutine to find out if pseudo-instructions rightPaw and leftPaw work. These pseudo-instructions return a non-zero value in register pair r25:24 if the answer is no and zero if the answer is yes. Send the yes/no answer to the question to two of the discrete LEDs on the Arduino Proto-Shield.

```
; - Test left & right paw --
; Lab 7
; Called from whichWay program
; Input: none        Outputs: spiLEDs bits 5 and 4 (left), 3 and 2 (right)
; No other registers or flags are modified by this subroutine
; -------------------------
testPaws:
   push   reg_F
   in     reg_F,SREG
   push   work0

   rcall  leftPaw

   mov    work0, spiLEDS
   tst    cppReg              // cppReg & cppReg
   breq   noLeftWall
   sbr    work0,0b00100000    // immediate instructions must use r16 to r31
   cbr    work0,0b00010000
   rjmp   overTheLeftWall
noLeftWall:
   sbr    work0,0b00010000
   cbr    work0,0b00100000
overTheLeftWall:
   rcall  rightPaw

   tst    cppReg              // cppReg & cppReg
   breq   noRightWall
   sbr    work0,0b00001000    // immediate instructions must use r16 to r31
   cbr    work0,0b00000100
   rjmp   overTheRightWall
noRightWall:
   sbr    work0,0b00000100
   cbr    work0,0b00001000
overTheRightWall:

   mov    spiLEDS,work0

   pop    work0
   out    SREG,reg_F
   pop    reg_F
   ret
```

APPENDIX A – ATMEGA328P INSTRUCTION SET[3]

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

[3] 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.

APPENDIX B – ARDUINO PROTO-SHIELD SCHEMATIC