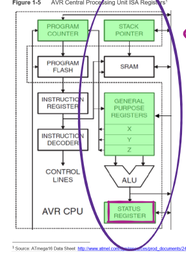
Instruction Set Architecture (Review)

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



1. Data Transfer
2. Arithmetic and Logic
3. Bit and Bit-Test
4. Control Transfer (Branch Instructions)
5. MCU Control

"Load the Program Counter"
`nop, sleep, wdr, break`

Fetch → Execute

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

ALU – Two Operand Instructions[2]

source | dest.

- All math (+, −, ×, ÷) and logic (and, or, xor) instructions work with the Register File (register to register).
- Most math and logic instructions have two operands Rd, Rr with register Rd initially containing one of the values to be operated on and ultimately the result of the operation. The initial contents of Rd are therefore destroyed by this operation.

```
add    Rd, Rr     ; Rd = Rd + Rr,  You may use any register (R0 – R31).
```

- Some math and logic operations replace the source register Rr with a constant K. Typically denoted by an "I" postfix.

```
subi   Rd, K      ; Rd = Rd - K,  You may only registers (R16 – R31).
```

add, adc, adiw — Adds two registers and the contents of the C Flag (adc only) and places the result in the destination register Rd.

sub, sbc, subi, sbci, sbiw — Subtracts the source register Rs or constant K from the source/destination register Rr and subtracts with the C Flag (sbc and sbci only) and places the result in the source/destination register Rd. Think of the C Flag as the Borrow bit within this context.

mul, muls, mulsu, fmul, fmuls, fmulsu — The multiplicand Rd and the multiplier Rr are two registers containing unsigned or fractional (f-prefix) encoded numbers. Both numbers may be unsigned (mul, fmul), or signed (muls, fmuls). Finally, the multiplicand Rd may be signed with the multiplier Rr unsigned (mulsu, fmulsu). The 16-bit unsigned product is placed in R1 (high byte) and R0 (low byte). $R1:R0 \leftarrow Rd \times Rs$

and, andi, or, ori, eor — Performs the logical AND, OR, and XOR operations between the contents of register Rd and register Rr or constant K.

[2] Source: Atmel 8-bit AVR Instruction Set Document 0856

ALU Program Example

Write an Assembly program to implement the polynomial expression

$B = A^2 + 41$

```
.INCLUDE <m32def.inc>
.DSEG                  @ala seg   SRAM
A:   .BYTE  1   ; 16 bit input
B:   .BYTE  2   ; 16 bit output
.CSEG                  code seg   FLASH
                        ; load
ldi  r16, A   ; r16 with the value of A
clr  r17        ; r17 with 0
ldi  r18, 41    ; r18 with 41
                ; do something
mul  r16, r16   ; r1:r0 = A^2
add  r0, r16
adc  r1, r17    ; r1:r0 = A^2 + A
add  r0, r18
adc  r1, r17    ; r1:r0 = A^2 + A + 41
                ; store
sts  B, r0      ; answer byte ordering
sts  B+1, r1    ; is little endian
```


$Rd \leftarrow Rd \bullet Rd$

a	b	a
0	0	0

$Rd \leftarrow Rd \oplus Rd$

a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

ALU – Single Operand Instructions[3]

- All single operand math and logic instructions only need a single register and usually the mnemonic alone is enough to tell you what it does.

Mnemonic	Operation	Description
com	Rd ← 0xFF − Rd	~~One's complement~~
neg	Rd ← 0x00 − Rd	Two's complement.
inc	Rd ← Rd + 1	Increment[4]
dec	Rd ← Rd − 1	Decrement[4]
clr	Rd ← Rd ⊕ Rd	Clear
ser	Rd ← 0xFF	Set Register, Limited to r16 – r31
tst	Rd ← Rd • Rd	Test for Zero or Minus

flip the bits

[3] Source: Atmel 8-bit AVR Instruction Set Document 0856
[4] The C Flag in SREG is not affected by the operation.

Questions answered by SREG Bits
Was answer zero? answer = Z
Is the answer correct (are you lying to me)?
answer: if +/- numbers = V otherwise C (also carry/borrow)
Is the answer positive or negative (really tell me the truth)?
answer: N (you can always trust S)

Computing ALU Status Register Bits – Addition –

	c_6	c_7	c_4	c_0
	1	1	1	
$b_7b_6b_5b_4$	0 1 0 0		1 1 0 0	
$b_3b_2b_1b_0$	1 0 1 1		1 0 1 0	
	0 0 0 0		0 1 1 0	

Unsigned
76 10
+186 10
06 10 ☹

Signed
76 10
−70 10
06 ☺

Explicit
Implicit

Subtraction
$H = \overline{c_4}$
$C = \overline{c_8}$

Unsigned
H = c_4 = 1
C = c_8 = 1

Signed
N = b_7 = 0
V = $c_6 \oplus c_7$ = 0 ← overflow ☺
S = N⊕V = 0

Arithmetic and Logical
Z = 0 answer is not zero
1 answer is zero

SREG – AVR Status Register[5]

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 (Rs → T) by the `bst` instruction, and a bit in T can be copied into a bit in a register (T → Rd) 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.

[5] Source: ATmega328P Data Sheet Document 8161 Section 6.3 Status Register

Big Endian
34
12

Little Endian
Byte Order
12
34

SRAM

	12 (most sig. byte)	0x0102
B	34	0x0101
A		0x0100

B = A² + A + 41

```
.DSEG
A:  .BYTE  1
B:  .BYTE  2
.CSEG
lds  r16, A
clr  r17
ldi  r18, 41
mul  r16, r16      // r1:r0 (most sig. byte)
add  r0, r16       r0 = r0 + r16
adc  r1, r17       r1 = r1 + ∅ + c
add  r0, r18
adc  r1, r17
sts  B, r0
sts  B+1, r1
```

SRAM address

0x0100	A
0x0101	B

r1 r0 + ∅ + c
r1 r0 + r18
r1 r0 + 41
r1 r0 + A
r1 r0 + A²
r1 r0 = 12 34
r1 r0 = 34 12
assume a

Appendix A – ATmega328P Instruction Set[2]

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 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 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	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	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,C,N,V,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H	1
CPI	Rd,K	Compare Register with Immediate	Rd - K	Z,C,N,V,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	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 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 Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag 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					
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,H	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,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),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
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
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
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
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
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 on the Stack	STACK ← Rr	None	2
POP	Rd	Pop from the Stack	Rd ← STACK	None	2

Computing ALU Status Register Bits – Subtraction –

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	

borrow
borrow
borrow

- For subtract instructions (`sub, subi, sbc, sbci, sbiw`) including compare instructions (`cp, cpc, cpi, cpse`), the carry bit is equal to: $C = \overline{c_4}$ and $H = \overline{c_3}$.
- Assume the subtract instruction `sub r16, r17` has just been run by the ATmega328P microcontroller. Complete the table provided. The "difference" column should reflect the contents of register r16 after the subtraction operation (leave the answer in 2's complement form) and not the actual difference (i.e., if done using your calculator).

r16	r17	difference	signed relationship	unsigned relationship	H	S	V	N	Z	C
3B	3B	00	+ = +	=	0	0	0	0	1	0
3B	15	26	+ > +	>	0	0	0	0	0	0
15	3B		+ < +	<						
F9	F6		− −	>						
F6	F9		− −	<						
15	F6		+ > −	<						
F6	15		− < +	>						
68	A5		+ > −	>						
A5	68		− < +	>						

- Use Atmel Studio simulation software to check your answers.

ldi r16, 0x3B
ldi r17, 0x15
• sub r16, r17

Use Simulator