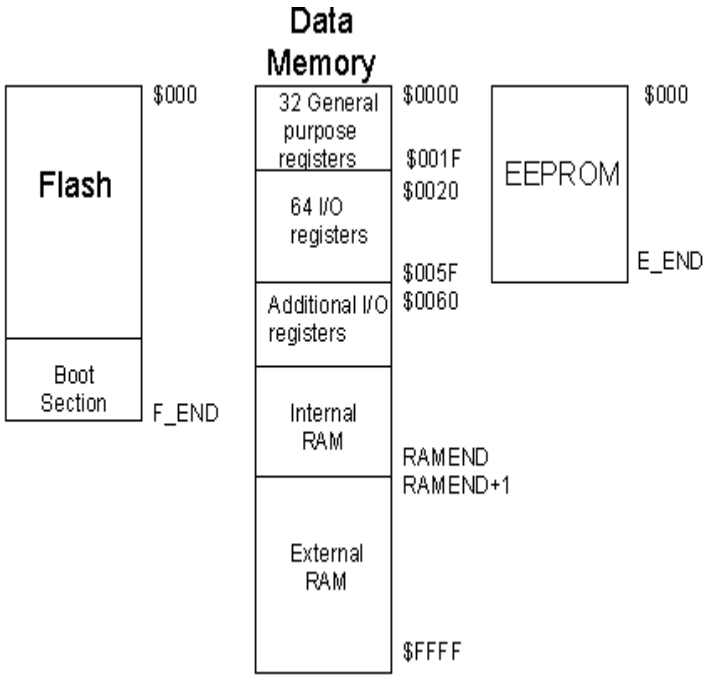
Design with Microprocessors

Year III Computer Science
1-st Semester

Lecture 2: AVR registers and instructions

AVR address spaces

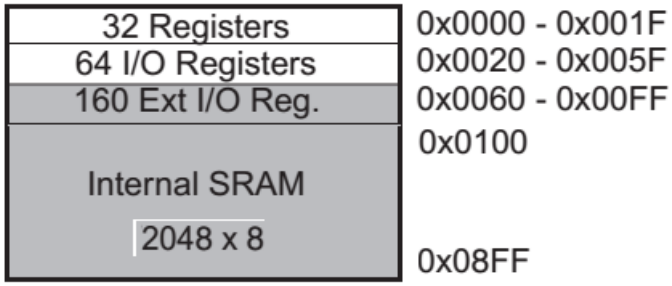


ATmega 328P / 2560

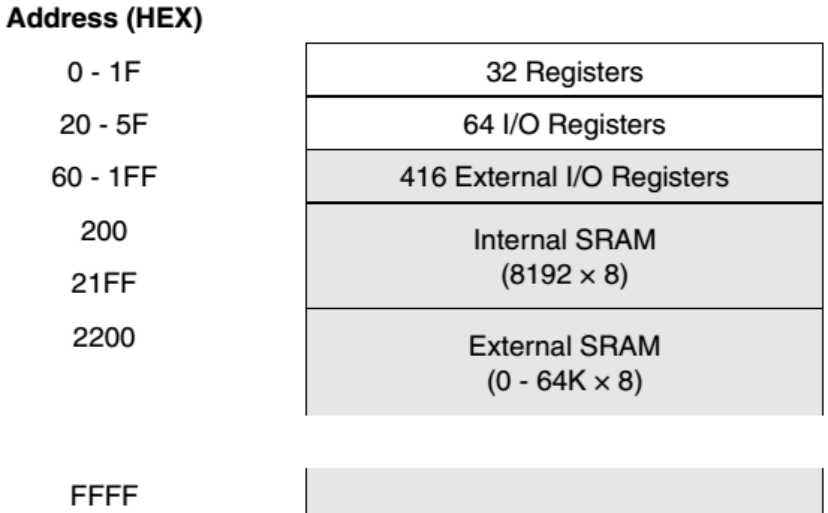
Flash: 0x0000 – 0x3FFF / 0x1FFF
- Program Counter (PC) - 14 / 17 bits
- Program memory is addressable on words (2 bytes): 16K x 16 biti (32KB) / 128Kx16 bits (256 KB)
- Program memory cannot be extended

Data memory: is addressable on bytes
Registers: 32 GPR, 64 I/O, 160/416 extended I/O
(0x0000 – 0x001F): General purpose registers (GPR):
(0x0020 – 0x005F): direct I/O access or as memory
(0x0060 – 0x00FF / 0x01FF): Extended I/O in SRAM, access with ST/STS/STD si LD/LDS/LDD
(0x0100 – 0x08FF / 0x0200 – 0x21FF): internal SRAM

Atmega 328P (Data Memory)

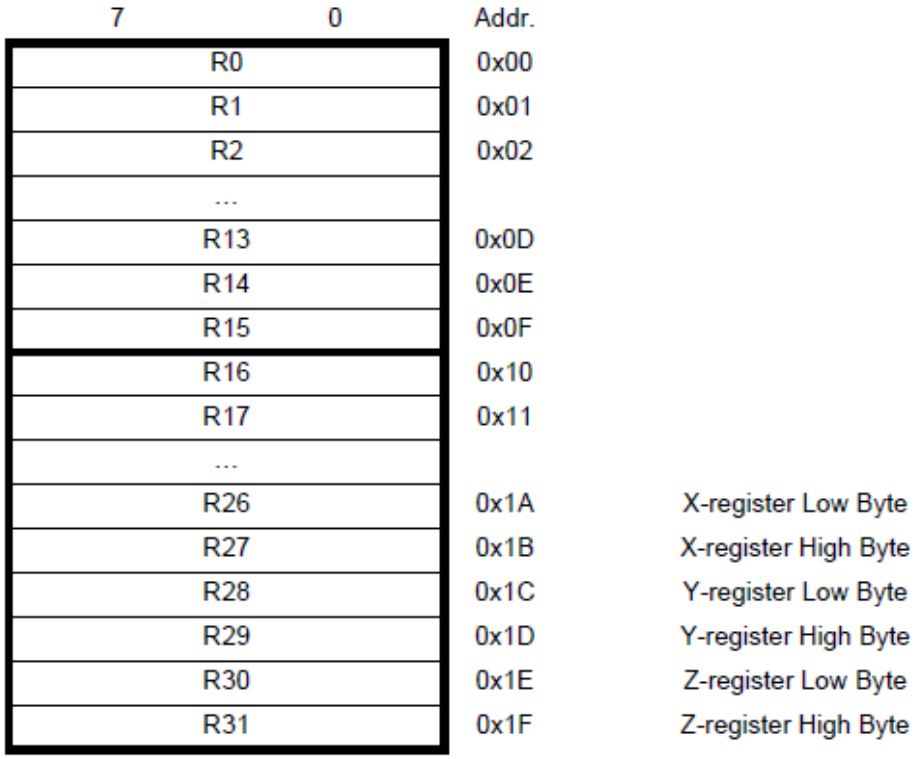


ATmega 2560 (Data Memory)



32 GPR

- 32 x 8-bit GPR (R0 – R31);
- 1 cycle read/write



Groups of registers:
R(0:15), R(16:31), R(26:31) – various possible usage
X, Y, Z, (16 bits); indirect addressing of instruction and data memory

Requirements for the register block (1 cycle operations) - input/output schemes:
•One 8-bit output operand and one 8-bit result input.
•Two 8-bit output operands and one 8-bit result input.
•Two 8-bit output operands and one 16-bit result input.
•One 16-bit output operand and one 16-bit result input.

64 I/O registers

(0x0020 – 0x005F): direct I/O access or as memory: 64

I/O register address mapping is processor dependent (see the datasheets):

Ex: ATmega 328P (UNO): http://www.atmel.com/Images/doc8161.pdf

0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–

64 I/O registers

Ex: ATmega 2560 (MEGA): http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

0x17 (0x37)	TIFR2	-	-	-	-	-	OCF2B	OCF2A	TOV2
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0
0x14 (0x34)	PORTG	-	-	PORTG5	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0
0x13 (0x33)	DDRG	-	-	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0
0x12 (0x32)	PING	-	-	PING5	PING4	PING3	PING2	PING1	PING0
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0
0x0E (0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0
0x0D (0x2D)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
0x0C (0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0

Control and status registers

SREG - Status Register

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	

Bit 7 – I: Global Interrupt Enable
I = SREG(7) ← 1/0; SEI/CLI; global validation/invalidation of interrupts

Bit 6 – T: Bit Copy Storage
Bit Copy: BLD (Bit LoaD) RegFile (Register(i)) ← SREG(T)
BST (Bit STore) SREG(T) ← RegFile (Register(i))

Bit 5 – H: Half Carry Flag; BCD arithmetic.

Bit 4 – S: Sign Bit, S = N xor V

Bit 3 – V: Two's Complement Overflow Flag

Bit 2 – N: Negative Flag; negative result in an arithmetic or logic operation.

Bit 1 – Z: Zero Flag; zero result in an arithmetic or logic operation.

Bit 0 – C: Carry Flag; carry in an arithmetic or logic operation.

SREG register can be accessed as follows:

Assembly:

```
read: in ri, SREG
write: out SREG, ri
```

C language)

```
byte sr;
sr=SREG;
SREG=sr;
```

Control and status registers

MCUCR – MCU Control Register

MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the I/O ports pull-up resistors are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-up resistor ({DDxn, PORTxn} = 0b01). See “Configuring the Pin” on page 68 for more details about this feature.

Instruction set

Instructions on 16 or 32 bits
The position of the registers addresses can vary in different instructions

Instruction types
- ARITHMETIC AND LOGIC INSTRUCTIONS
- BRANCH INSTRUCTIONS
- DATA TRANSFER INSTRUCTIONS
- BIT AND BIT-TEST INSTRUCTIONS
- MCU CONTROL INSTRUCTIONS

Registers and Operands (notations):
Rd: Destination (and source) register in the Register File
Rr: Source register in the Register File
R: Result after instruction is executed
K: Constant data
k: Constant address
b: Bit in the Register File or I/O Register (3-bit)
s: Bit in the Status Register (3-bit)
X,Y,Z: Indirect Address Register
(X=R27:R26, Y=R29:R28 and Z=R31:R30)
A: I/O location address
q: Displacement for direct addressing (6-bit)

Data Transfer Instructions

DATA TRANSFER INSTRUCTIONS			
MOV	Rd, Rr	Move Between Registers	Rd ← Rr
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr
LDI	Rd, K	Load Immediate	Rd ← K
LD	Rd, X	Load Indirect	Rd ← (X)
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)
LD	Rd, Y	Load Indirect	Rd ← (Y)
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)
LD	Rd, Z	Load Indirect	Rd ← (Z)
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)
ST	X, Rr	Store Indirect	(X) ← Rr
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr
ST	Y, Rr	Store Indirect	(Y) ← Rr
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr
ST	Z, Rr	Store Indirect	(Z) ← Rr
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr
STS	k, Rr	Store Direct to SRAM	(k) ← Rr
LPM		Load Program Memory	R0 ← (Z)
LPM	Rd, Z	Load Program Memory	Rd ← (Z)
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1
SPM		Store Program Memory	(Z) ← R1:R0
IN	Rd, P	In Port	Rd ← P
OUT	P, Rr	Out Port	P ← Rr
PUSH	Rr	Push Register on Stack	STACK ← Rr
POP	Rd	Pop Register from Stack	Rd ← STACK

Data Transfer Instructions

Instruction formats and restrictions

LDI Rd,K $16 \le d \le 31, 0 \le K \le 255$ PC ← PC + 1

1110	KKKK	dddd	KKKK

(i) LD Rd, X $0 \le d \le 31$ PC ← PC + 1
(ii) LD Rd, X+ $0 \le d \le 31$ PC ← PC + 1
(iii) LD Rd, -X $0 \le d \le 31$ PC ← PC + 1

(i)	1001	000d	dddd	1100
(ii)	1001	000d	dddd	1101
(iii)	1001	000d	dddd	1110

LD

(i) LPM None, R0 implied PC ← PC + 1
(ii) LPM Rd, Z $0 \le d \le 31$ PC ← PC + 1
(iii) LPM Rd, Z+ $0 \le d \le 31$ PC ← PC + 1

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

LPM

IN Rd,A $0 \le d \le 31, 0 \le A \le 63$ PC ← PC + 1

1011	0AAd	dddd	AAAA

IN

OUT A,Rr $0 \le r \le 31, 0 \le A \le 63$ PC ← PC + 1

1011	1AAr	rrrr	AAAA

OUT

Data Transfer Instructions - examples

Load Immediate

```
char a;
…
a = 0x10;
==============================
ldi r24, 0x10        ; Load imm 10
sts a, r24           ; Store to a
------------------------------------------
int a;
…
a = 0x2030;
==============================
???
```

Load direct

```
char a, b;
…
a = b;
================
lds r24, b           ; Load from b
sts a, r24           ; Store to a
------------------------------------------
int a, b;
…
a = b;
================
???
```

AVR Instructions: restrictions

About Code Examples – Datasheet: I/O Regs in extended I/O map:
For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

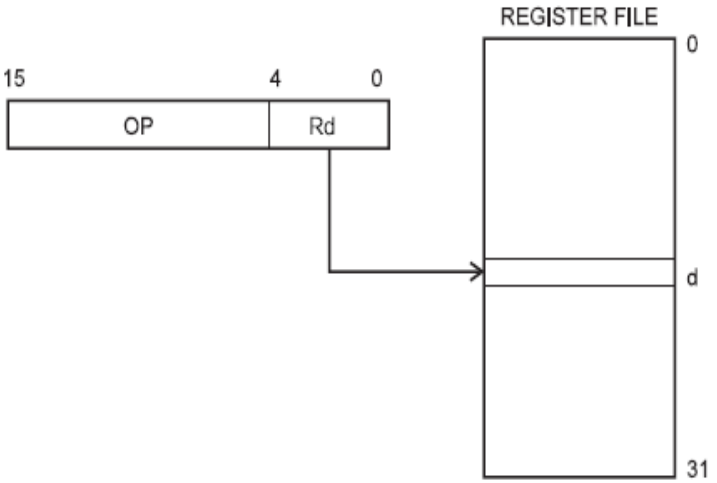
AVR - addressing modes

AVR, addressing modes

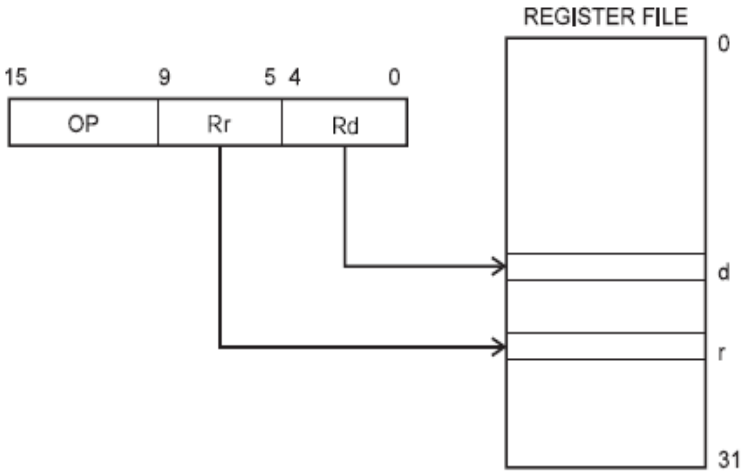
- Register Direct, with 1 and 2 registers
- I/O Direct
- Data Direct
- Data Indirect
  - with pre-decrement
  - with post-increment
- Addressing instruction memory

The Program and Data Addressing Modes

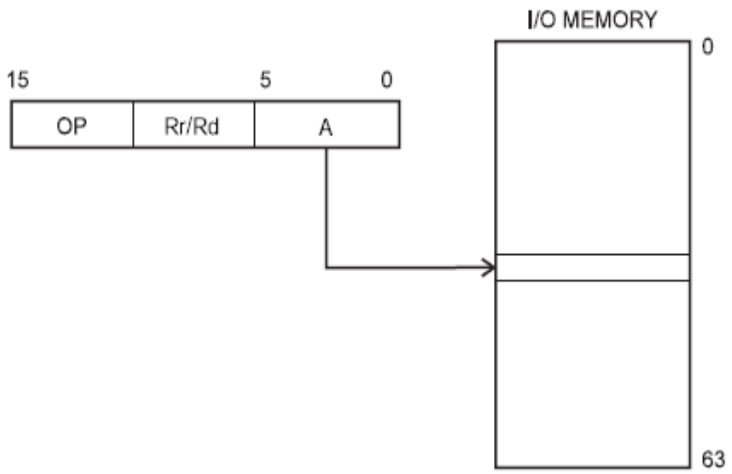
Direct Register Addressing



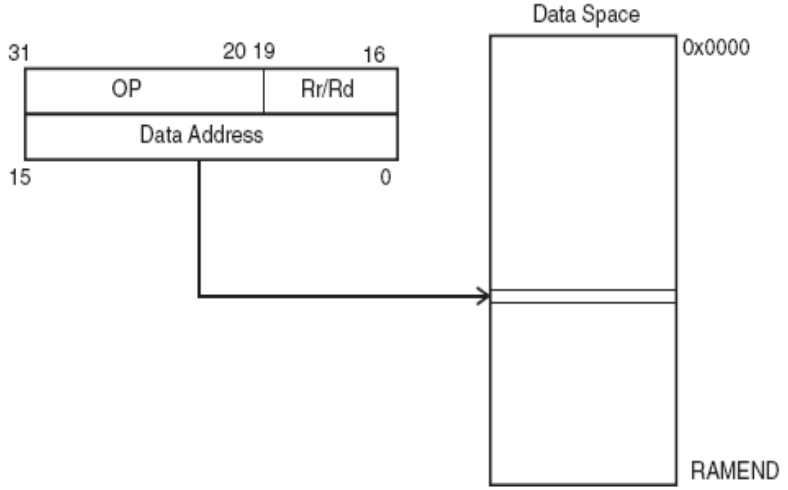
Direct Single Register Addressing



Direct Register Addressing, Two Registers



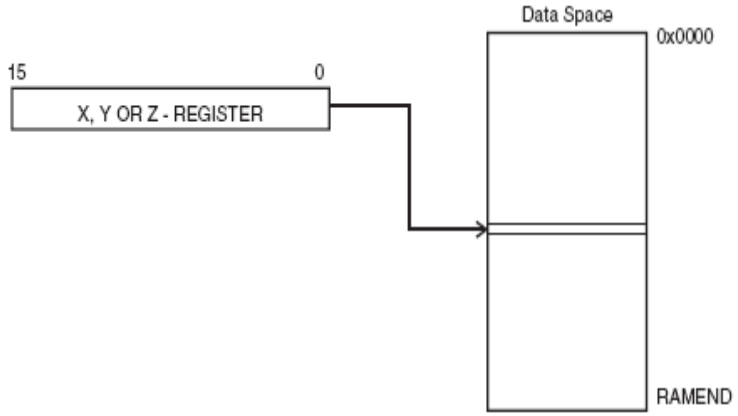
I/O Direct Addressing



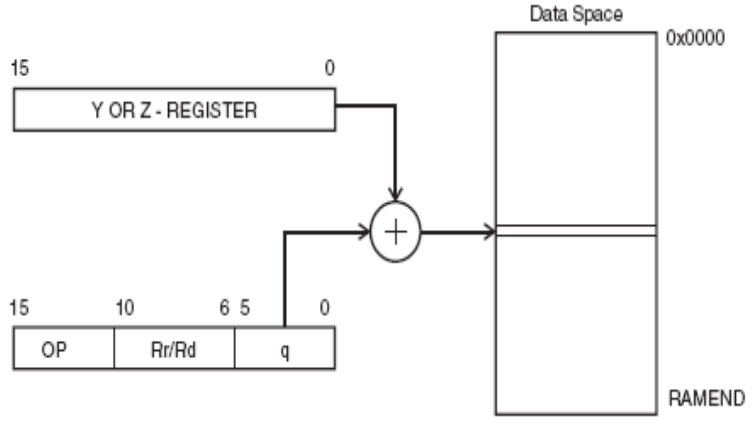
Direct Data Addressing

The Program and Data Addressing Modes

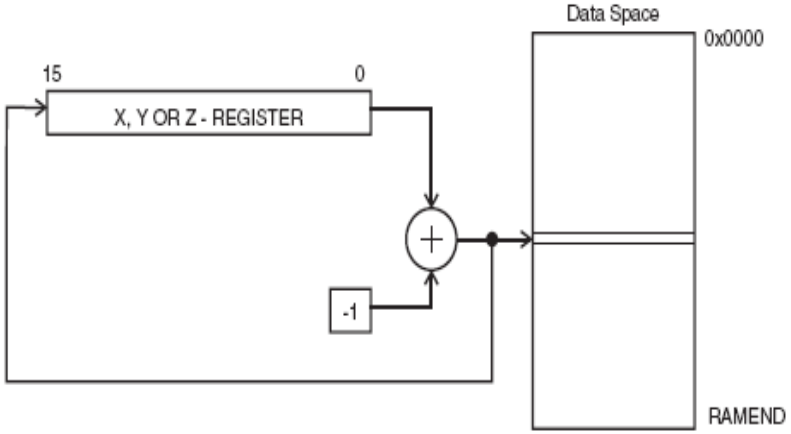
Indirect Data Addressing



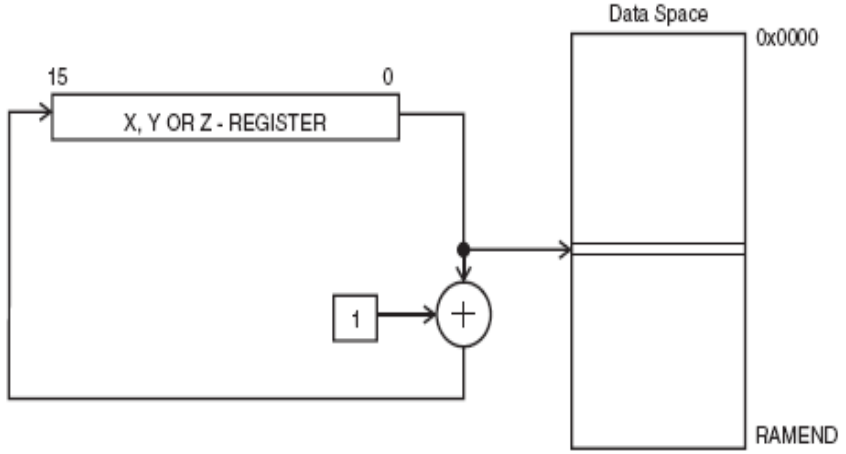
Indirect Data Addressing



Indirect Data addressing with Displacement



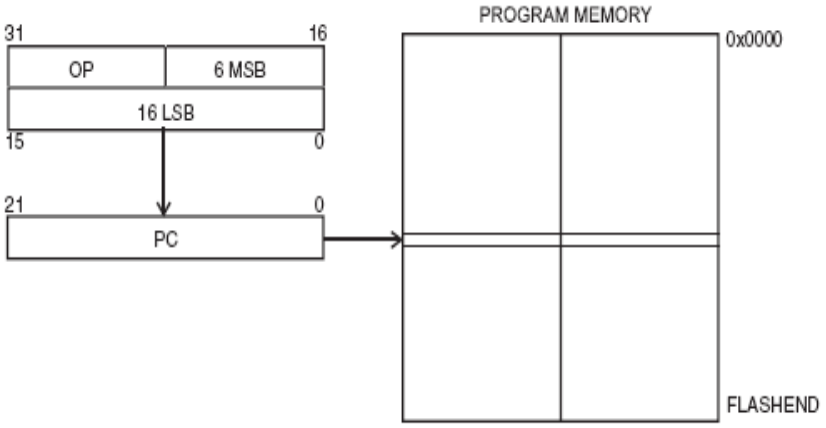
Indirect Data Addressing with Pre-decrement



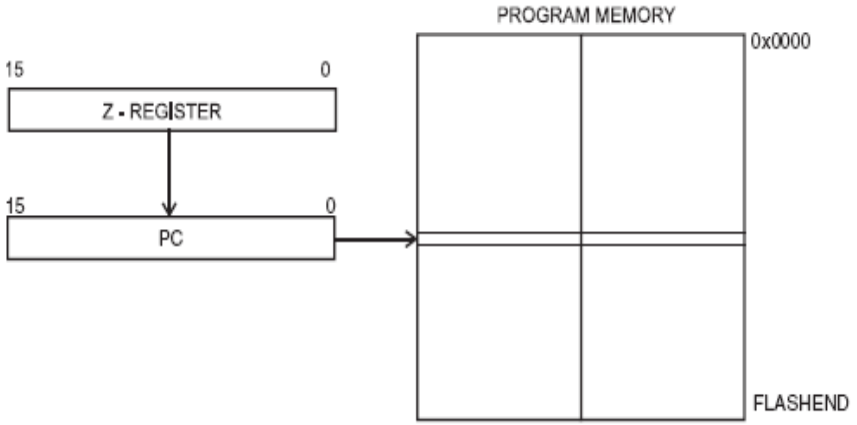
and Post-increment

The Program and Data Addressing Modes

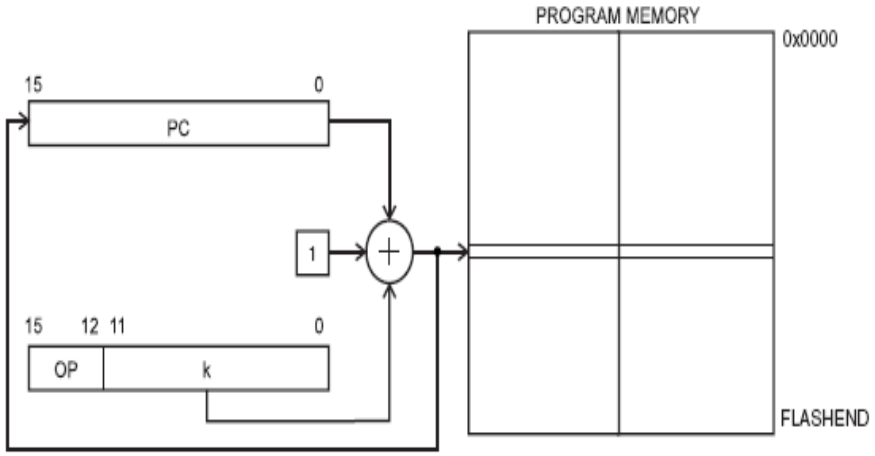
Program Addressing



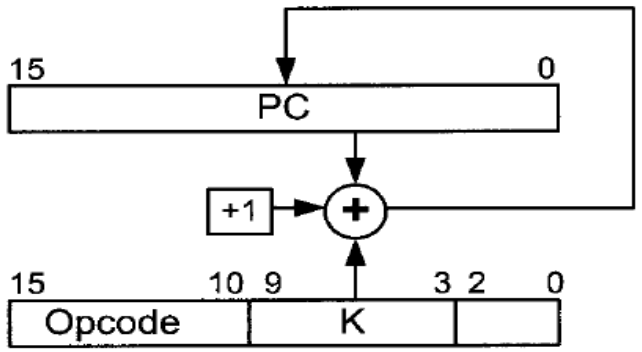
Direct Program Addressing, JMP, CALL



Indirect Program Addressing, IJMP, ICALL



Relative Program Addressing, RJMP, RCALL



Conditional Branch – short relative jump

Data Transfer Instructions - examples

Load Indirect

```
int a = *pInt;
============================
; Use the Z register (R31:R30)
lds R30, pInt      ; Load from pInt
lds R31, pInt+1    ;
ld r24, Z          ; load from (*pInt)
ldd r25, Z+1       ;
sts a, r24         ; store to a
sts a+1, r25       ;
-----------------------------------------
```

Store Indirect

```
* (int*) (0x0100) = b;
============================
; Use Z register (R31:R30)
ldi r30, 0         ; load imm 0x0100
ldi r31, 0x01      ;
lds r24, b          ; load b
lds r25, b+1       ;
st Z, r24          ; store to 0x0100
std Z+1, r25       ;
-----------------------------------------
```

String Copy

```
void strcpy (char *dst, char *src)
{ char ch;
do {
ch = *src++;
*dst++ = ch;
} while (ch);
}
```

```
; dst in R25:R24, src in R23:R22
strcpy:
movw r30, r24     ; Z<=dst
movw r26, r22     ; X<=src
loop:
  ld r20, X+      ; ch=*src++
  st Z+, r20      ; *dst++=ch
  tst r20          ; ch==0?
  brne loop       ; loop if not
ret
```

Arithmetic and Logic Instructions

Mnemonics	Operands	Description	Operation
ARITHMETIC AND LOGIC INSTRUCTIONS			
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr
COM	Rd	One's Complement	Rd ← 0xFF – Rd
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)
INC	Rd	Increment	Rd ← Rd + 1
DEC	Rd	Decrement	Rd ← Rd – 1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd
SER	Rd	Set Register	Rd ← 0xFF
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ¨ (Rd x Rr) << 1
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ¨ (Rd x Rr) << 1
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ¨ (Rd x Rr) << 1

Arithmetic and Logic Instructions

Instruction formats and restrictions

ADC Rd,Rr $0 \le d \le 31, 0 \le r \le 31$ PC ← PC + 1

0001	11rd	dddd	rrrr

ADIW Rd+1:Rd,K d ε {24,26,28,30}, $0 \le K \le 63$ PC ← PC + 1

1001	0110	KKdd	KKKK

ANDI Rd,K $16 \le d \le 31, 0 \le K \le 255$ PC ← PC + 1

0111	KKKK	dddd	KKKK

TST Rd $0 \le d \le 31$ PC ← PC + 1

0010	00dd	dddd	dddd

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

SREG

A&L instructions - examples

C language

```
int a, b;
…
a = a + b;
```

==========

Assambly language

```
lds r18, a          ; load a
lds r19, a+1        ;
lds r24, b          ; load b
lds r25, b+1        ;
add r24, r18        ; add lower half
adc r25, r19        ; add higher half
sts a+1, r25        ; store a.byte1
sts a, r24          ; store a.byte0
```

```
int a;
…
a -= 0x4321;
```

==========

```
lds r24, a          ; load from a
lds r25, a+1        ;
subi r24, 0x21      ; sub imm 0x21
sbci r25, 0x43      ; sub imm 0x43 with
                      carry
sts a+1, r25        ; store a
sts a, r24          ;
```

Branch Instructions

BRANCH INSTRUCTIONS			
RJMP	k	Relative Jump	PC ← PC + k + 1
IJMP		Indirect Jump to (Z)	PC ← Z
JMP	k	Direct Jump	PC ← k
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1
ICALL		Indirect Call to (Z)	PC ← Z
CALL	k	Direct Subroutine Call	PC ← k
RET		Subroutine Return	PC ← STACK
RETI		Interrupt Return	PC ← STACK
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3
CP	Rd,Rr	Compare	Rd – Rr
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C
CPI	Rd,K	Compare Register with Immediate	Rd – K
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1

Branch Instructions

Instruction formats and restrictions

RJMP k -2K ≤ k < 2K PC ← PC + k + 1

1100	kkkk	kkkk	kkkk

JMP k 0 ≤ k < 4M PC ← k

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

CALL k 0 ≤ k < 64K PC ← k STAC K ← PC+2
SP ← SP-2, (2 bytes, 16 bits)

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

RCALL k -2K ≤ k < 2K PC ← PC + k + 1 STACK ← PC + 1
SP ← SP - 2 (2 bytes, 16 bits)

1101	kkkk	kkkk	kkkk

BREQ k -64 ≤ k ≤ +63 PC ← PC + k + 1
PC ← PC + 1, if condition is false

1111	00kk	kkkk	k001

CPI Rd,K 16 ≤ d ≤ 31, 0≤ K ≤ 255 PC ← PC + 1

0011	KKKK	dddd	KKKK

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

SREG

Branch Instructions - examples

```
-----------------------------------------
int a;
if (a > 0)
…
else ..
=========================
lds r24, a          ; load a
lds r25, a+1        ;
CLR R1              ; R1 ← 0
CP R1, R24          ; compare lower half
CPC R1, R25         ; compare higher half
BRGE else           ; branch if (S==0)
                    ; a < 0 )
….
else:
-----------------------------------------
```

CP: Rd –Rr
CPC: Rd - Rr - C

```
char ch;
if (ch < 0)
ch = -ch;
==============================
lds r24, ch         ; load ch
tst r24             ; test for zero or
                    ;   negative
brge end_if         ; branch to
                    ; PC+3+1 if greater-
                    ; than-or-equal
neg r24             ; ch = -ch
sts ch, r24         ; save ch
end_if: …
-----------------------------------------------
```

Branch Instructions - examples

If-else Statement

```
char ch;
…
if (ch < 0)
ch = -1;
else
ch = 1;
=========================
lds r24, ch              ; load ch
tst r24                  ; test ch
brge else ; if GE goto else
          ; part
then:
  ldi r24, -1            ; ch=-1
  rjmp endif             ; skip the else
                         ;  part
else:
  ldi r24, 1             ; ch=1
endif:
  sts ch, R24
-----------------------------------------
```

WHILE Loop

```
unsigned char sum, n;
… while (n<10) {
            sum += n;
            n++;
            }
==================
C IF version:
unsigned char sum n;
…
loop:
            if (n >= 10) goto done;
            sum += n;
            n ++;
            goto loop;
done:
=====================
; assume r16=n, r3=initial value for sum
loop:       cpi r16, 10
  brsh next   ; branch if same or higher (C==0)
  add r3, r16
  incr 16
  rjmp loop
next:
```

Bit & Bit-test Instructions. MCU Control

BIT AND BIT-TEST INSTRUCTIONS			
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)
BSET	s	Flag Set	SREG(s) ← 1
BCLR	s	Flag Clear	SREG(s) ← 0
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T
SEC		Set Carry	C ← 1
CLC		Clear Carry	C ← 0
SEN		Set Negative Flag	N ← 1
CLN		Clear Negative Flag	N ← 0
SEZ		Set Zero Flag	Z ← 1
CLZ		Clear Zero Flag	Z ← 0
SEI		Global Interrupt Enable	I ← 1
CLI		Global Interrupt Disable	I ← 0
SES		Set Signed Test Flag	S ← 1
CLS		Clear Signed Test Flag	S ← 0
SEV		Set Twos Complement Overflow.	V ← 1
CLV		Clear Twos Complement Overflow	V ← 0
SET		Set T in SREG	T ← 1
CLT		Clear T in SREG	T ← 0
SEH		Set Half Carry Flag in SREG	H ← 1
CLH		Clear Half Carry Flag in SREG	H ← 0
MCU CONTROL INSTRUCTIONS			
NOP		No Operation	
SLEEP		Sleep	(see specific descr. for Sleep function)
WDR		Watchdog Reset	(see specific descr. for WDR/timer)
BREAK		Break	For On-chip Debug Only

Bit & Bit-test Instructions. MCU Control

Instruction formats and restrictions

SBI A,b $0 \le A \le 31, 0 \le b \le 7$ PC ← PC + 1

1001	1010	AAAA	Abbb

AVR instructions - summary

Mnemonic	Description
Flow Control	
JMP ◆	Jump absolute (24-bit)
RJMP	Branch relative (12-bit)
IJMP ●	Jump indirect (Z)
RCALL	Call subroutine
ICALL ●	Call subroutine indirect (Z)
RET/RETI	Return/from interrupt
CP/CPC	Compare/with carry
CPI	Compare with 8-bit immediate
CPSE	Compare, skip if equal
SBRS/SBRC	Skip if register bit set/clear
SBIS/SBIC	Skip if I/O bit set/clear
BRcc	Conditional branch
Logical	
AND	Logical AND
ANDI	Logical AND 8-bit immediate
OR	Logical OR
ORI	Logical OR 8-bit immediate
EOR	Logical exclusive-OR
LSL/LSR	Logical shift left/right by 1 bit
ROL/ROR	Rotate left/right by 1 bit
ASR	Arithmetic shift right by 1 bit
COM/NEG	One's/two's complement
SWAP	Swap nibbles
TST	Test for zero or minus

Mnemonic	Description
Bit Manipulation	
SEC/CLC	Set/clear C flag (carry)
SEH/CLH	Set/clear H flag (half carry)
SEN/CLN	Set/clear N flag (negative)
SEZ/CLZ	Set/clear Z flag (zero)
SEI/CLI	Set/clear I flag (interrupt)
SES/CLS	Set/clear S flag (sign)
SEV/CLV	Set/clear V flag (overflow)
SET/CLT	Set/clear T bit
SBR/CBR	Set/clear bit in register
BSET/BCLR	Set/clear bit in status register
SER/CLR	Set/clear entire register
SBI/CBI	Set/clear bit in I/O space
Arithmetic	
ADD/ADC	Add/with carry
ADIW ●	Add 6-bit immediate
SUB/SUBC	Subtract/with borrow
SBIW ●	Subtract 6-bit immediate
SUBI/SBCI	Subtract 8-bit imm/w borrow
INC/DEC	Increment/decrement register
MUL ◆	Multiply $8 \times 8 \rightarrow 16$
	Can use R16–R31 only
	Can use R24–R31 only

Mnemonic	Description
Load/Store	
MOV	Copy register to register
LD	Load indirect through X/Y/Z
LD ●	Load indirect with postincremnt
LD ●	Load indirect with predecremnt
LDD ●	Load indirect with 6-bit offset
LDI	Load 8-bit immediate
LDS ●	Load from 16-bit address
LPS ●	Load from program space
ST	Store indirect through X/Y/Z
ST ●	Store indirect with postincremnt
ST ●	Store indirect with predecremnt
STD ●	Store indirect with 6-bit offset
STS ●	Store to 16-bit address
IN/OUT	Input/output to/from I/O space
PUSH/POP	Push/pop stack element
BLD/BST	Load/store T bit
Miscellaneous	
NOP	No operation
SLEEP	Wait for interrupt
WDR	Watchdog reset
●	Not available on 90S1200, 1220
◆	Future enhancement

References

Zhao Zhang, AVR Assembly Language, Translating C programs into

Assembly – lecture notes (lecture-week10.pdf, lecture-week11.pdf, lecture-week12.pdf, lecture-week13.pdf)

Atmel, 8-bit AVR instruction set (AVR Instruction Set.pdf)

AVR Instruction Set, Complete Instruction Set Summary (AVRINSTR.HLP)

ATmega 328P datasheet, http://www.atmel.com/Images/doc8161.pdf

Atmel, ATmega 2560 datasheet, http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf