Instruction	Description	Op 1	Op 2	Effect on operands	Effect on specials	Example
Load/Store instructions						
ldi	Load immediate	reg 16-31	8-bit value	op1 = op2	incPC; sreg: none	ldi r16, 45
mov	Move	reg	reg	op1 = op2	incPC; sreg: none	mov r1, r2
ld	Load from RAM	reg	X	op1 = RAM[r26 + 256*r27]	incPC; sreg: none	ld r1, X
st	Store in RAM	X	reg	RAM[r26 + 256*r27] = op2	incPC; sreg: none	st X, r1
Computation Instructions						
add	Add	reg	reg	op1 = op1 + op2	incPC; sreg: NZC	add r1, r2
sub	Subtract	reg	reg	op1 = op1 - op2	incPC; sreg: NZC	sub r1, r2
inc	Increment register	reg	none	op1 = op1 + 1	incPC; sreg: NZ	inc r1
dec	Decrement register	reg	none	op1 = op1 - 1	incPC; sreg: NZ	dec r1
and	And	reg	reg	op1 = op1 & op2	incPC; sreg: NZ	and r1, r2
or	Or	reg	reg	op1 = op1 | op2	incPC; sreg: NZ	or r1, r2
eor	Exclusive-or	reg	reg	op1 = op1 ^ op2	incPC; sreg: NZ	eor r1, r2
com	Complement, Not	reg	none	op1 = ~op1	incPC; sreg: NZ	com r1
neg	Negate	reg	none	op1 = -op1	incPC; sreg: NZC	neg r1
asr	Arithmetic shift right	reg	none	op1 = op1 >> 1 MSB unmodified	incPC; sreg: NZC	asr r1
cp	Compare two registers	reg	reg	none	incPC; sreg: NZC#	cp r1, r2
subi	Subtract immediate	reg 16-31	8-bit value	op1 = op1 - op2	incPC; sreg: NZC	subi r16, 10
andi	And immediate	reg 16-31	8-bit value	op1 = op1 & op2	incPC; sreg: NZ	andi r16, 1
ori	Or immediate	reg 16-31	8-bit value	op1 = op1 | op2	incPC; sreg: NZ	ori r16, 1
adc	Add with carry	reg	reg	op1 = op1 + op2 + C	incPC; sreg: NZC	adc r1, r2
sbc	Subtract with carry	reg	reg	op1 = op1 - op2 - C	incPC; sreg: NZC	sbc r1, r2
Input/Output						
in	Read I/O register	reg	value 0-63	Read op2 into op1	incPC; sreg: none	in r1, 16
out	Write to I/O register	value 0-63	reg	Write op2 to op1	incPC; sreg: none	out 18, r1
Control-Flow						
rjmp	Relative jump	Value: -2048 to 2047	none	none	pc=pc+op1; incPC; sreg: none	rjmp 4 rjmp -3
breq	Branch if equal	Value: -64 to +63	none	none	If Z set, pc=pc+op1; incPC; sreg: none	breq 2 breq -3
brne	Branch if not equal	Value: -64 to +63	none	none	If Z clear, pc=pc+op1; incPC; sreg: none	brne 2 brne -3
brsh	Branch if same or higher	Value: -64 to +63	none	none	If C clear, pc=pc+op1; incPC; sreg: none	brsh 2 brsh -3
brlo	Branch if lower	Value: -64 to +63	none	none	If C set, pc=pc+op1; incPC; sreg: none	brlo 2 brlo -3
rcall	Call relative address	Value: -2048 to 2047	none	push pc+1	pc=pc+op1; incPC; sreg: none	rcall -10
ret	Return from subroutine	none	none		pop pc; sreg: none	ret
Stack						
push	Push register to stack	reg	none	RAM[sp] = op1 sp = sp - 1	incPC; sreg: none	push r1
pop	Pop register off stack	reg	none	sp = sp + 1 op1 = RAM[sp]	incPC; sreg: none	pop r1

Setting sreg
N: set if result is negative
Z: set if result is 0
add, addi, adc C: set if op1 + op2 > 255
sub, subi C: set if op1 < op2 (unsigned)
neg, asr C: set if op1 was odd
sbc C: set if op1 + C < op2 (unsigned)

Setting sreg for cp
if (op1 > op2) set N else clear N
if (op1 == op2) set Z else clear Z
if (op1 < op2) set C else clear C

PIND: IO reg 16
DDRD: IO reg 17
PORTD: IO reg 18
SPL: IO reg 61
SPH: IO reg 62

Asm Directive	Example
labels	label_name:
byte	.byte(label_name) 0,1,1,2,3,5,8
string	.string(label_name) "hello world"

ISA Operations
lo8 ldi r26, lo8(label_name)
hi8 ldi r27, hi8(label_name)

Program layout

Prog memory

```
ldi r31, 91
out 62, r31
ldi r31, 136
out 61, r31      -> set sp
rjmp prog_beg    -> jump to program start
code for functions
code for functions
code for functions
code for functions
code for functions
program_beg:
code for main prog.
code for main prog.
code for main prog.
code for main prog.
code for main prog.
```

```
if (x < y)
  foo
else
  bar
rest of program

;x in r1,y in r2
cp r1, r2
brlo foo_code
bar line 1
bar line 2
bar line 3
rjmp merge_point
foo_code:
foo line 1
foo line 2
foo line 3
merge_point:
rest of prog.
```

```
if (x == y)
;x in r1,y in r2
cp r1, r2
breq foo_code

if (x ==10)
;x in r1,y in r2
ldi r31, 10
cp r1, r31
breq foo_code

if (x >= y)
;x in r1,y in r2
cp r1, r2
brsh foo_code
```

```
write reg 2 to output
ldi r31, 255
out 17, r31
out 18, r2
```