9.4 AVR Dependent Features

9.4.1 Options

`-mmcu=`*mcu*

Specify ATMEL AVR instruction set or MCU type.

Instruction set avr1 is for the minimal AVR core, not supported by the C compiler, only for assembler programs (MCU types: at90s1200, attiny11, attiny12, attiny15, attiny28).

Instruction set avr2 (default) is for the classic AVR core with up to 8K program memory space (MCU types: at90s2313, at90s2323, at90s2333, at90s2343, attiny22, attiny26, at90s4414, at90s4433, at90s4434, at90s8515, at90c8534, at90s8535).

Instruction set avr25 is for the classic AVR core with up to 8K program memory space plus the MOVW instruction (MCU types: attiny13, attiny13a, attiny2313, attiny24, attiny44, attiny84, attiny25, attiny45, attiny85, attiny261, attiny461, attiny861, attiny87, attiny43u, attiny48, attiny88, at86rf401, ata6289).

Instruction set avr3 is for the classic AVR core with up to 128K program memory space (MCU types: at43usb355, at76c711).

Instruction set avr31 is for the classic AVR core with exactly 128K program memory space (MCU types: atmega103, at43usb320).

Instruction set avr35 is for classic AVR core plus MOVW, CALL, and JMP instructions (MCU types: attiny167, attiny327, at90usb82, at90usb162).

Instruction set avr4 is for the enhanced AVR core with up to 8K program memory space (MCU types: atmega48, atmega48p,atmega8, atmega88, atmega88p, atmega8515, atmega8535, atmega8hva, atmega4hvd, atmega8hvd, at90pwm1, at90pwm2, at90pwm2b, at90pwm3, at90pwm3b, at90pwm81).

Instruction set avr5 is for the enhanced AVR core with up to 128K program memory space (MCU types: atmega16, atmega161, atmega162, atmega163, atmega164p, atmega165, atmega165p, atmega168, atmega168p, atmega169, atmega169p, atmega32, atmega323, atmega324p, atmega325, atmega325p, atmega3250, atmega3250p, atmega328p, atmega329, atmega329p, atmega3290, atmega3290p, atmega406, atmega64, atmega640, atmega644, atmega644p, atmega645, atmega6450, atmega649, atmega6490, atmega16hva, atmega16hvb, atmega32hvb, at90can32, at90can64, at90pwm216, at90pwm316, atmega32c1, atmega64c1, atmega16m1, atmega32m1, atmega64m1, atmega16u4, atmega32u4, atmega32u6, at90usb646, at90usb647, at94k, at90scr100).

Instruction set avr51 is for the enhanced AVR core with exactly 128K program memory space (MCU types: atmega128, atmega1280, atmega1281, atmega1284p, atmega128rfa1, at90can128, at90usb1286, at90usb1287, m3000f, m3000s, m3001b).

Instruction set avr6 is for the enhanced AVR core with a 3-byte PC (MCU types: atmega2560, atmega2561).

`-mall-opcodes`
: Accept all AVR opcodes, even if not supported by `-mmcu`.

`-mno-skip-bug`
: This option disable warnings for skipping two-word instructions.

`-mno-wrap`
: This option reject `rjmp/rcall` instructions with 8K wrap-around.

9.4.2 Syntax

9.4.2.1 Special Characters

The presence of a '`;`' on a line indicates the start of a comment that extends to the end of the current line. If a '`#`' appears as the first character of a line, the whole line is treated as a comment.

The '`$`' character can be used instead of a newline to separate statements.

9.4.2.2 Register Names

The AVR has 32 x 8-bit general purpose working registers '`r0`', '`r1`', ... '`r31`'. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing. One of the these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit '`X`', '`Y`' and '`Z`' - registers.

```
X = r26:r27
Y = r28:r29
Z = r30:r31
```

9.4.2.3 Relocatable Expression Modifiers

The assembler supports several modifiers when using relocatable addresses in AVR instruction operands. The general syntax is the following:

```
modifier(relocatable-expression)
```

`lo8`
: This modifier allows you to use bits 0 through 7 of an address expression as 8 bit relocatable expression.

`hi8`
: This modifier allows you to use bits 7 through 15 of an address expression as 8 bit relocatable expression. This is useful with, for example, the AVR '`ldi`' instruction and '`lo8`' modifier.

  For example

```
ldi r26, lo8(sym+10)
ldi r27, hi8(sym+10)
```

`hh8`
: This modifier allows you to use bits 16 through 23 of an address expression as 8 bit relocatable expression. Also, can be useful for loading 32 bit constants.

`hlo8`
: Synonym of '`hh8`'.

`hhi8`

This modifier allows you to use bits 24 through 31 of an expression as 8 bit expression. This is useful with, for example, the AVR '`ldi`' instruction and '`lo8`', '`hi8`', '`hlo8`', '`hhi8`', modifier.

For example

```
ldi r26, lo8(285774925)
ldi r27, hi8(285774925)
ldi r28, hlo8(285774925)
ldi r29, hhi8(285774925)
; r29,r28,r27,r26 = 285774925
```

`pm_lo8`

This modifier allows you to use bits 0 through 7 of an address expression as 8 bit relocatable expression. This modifier useful for addressing data or code from Flash/Program memory. The using of '`pm_lo8`' similar to '`lo8`'.

`pm_hi8`

This modifier allows you to use bits 8 through 15 of an address expression as 8 bit relocatable expression. This modifier useful for addressing data or code from Flash/Program memory.

`pm_hh8`

This modifier allows you to use bits 15 through 23 of an address expression as 8 bit relocatable expression. This modifier useful for addressing data or code from Flash/Program memory.

9.4.3 Opcodes

For detailed information on the AVR machine instruction set, see `www.atmel.com/products/AVR`.

`as` implements all the standard AVR opcodes. The following table summarizes the AVR opcodes, and their arguments.

Legend:

`r`	any register
`d`	'ldi' register (r16-r31)
`v`	'movw' even register (r0, r2, ..., r28, r30)
`a`	'fmul' register (r16-r23)
`w`	'adiw' register (r24,r26,r28,r30)
`e`	pointer registers (X,Y,Z)
`b`	base pointer register and displacement ([YZ]+disp)
`z`	Z pointer register (for [e]lpm Rd,Z[+])
`M`	immediate value from 0 to 255
`n`	immediate value from 0 to 255 (n = ~M). Relocation impossible
`s`	immediate value from 0 to 7
`P`	Port address value from 0 to 63. (in, out)
`p`	Port address value from 0 to 31. (cbi, sbi, sbic, sbis)
`K`	immediate value from 0 to 63 (used in 'adiw', 'sbiw')
`i`	immediate value
`l`	signed pc relative offset from -64 to 63
`L`	signed pc relative offset from -2048 to 2047
`h`	absolute code address (call, jmp)
`S`	immediate value from 0 to 7 (S = s << 4)
`?`	use this opcode entry if no parameters, else use next opcode entry

```
1001010010001000   clc
1001010011011000   clh
1001010011111000   cli
1001010010101000   cln
1001010011001000   cls
1001010011101000   clt
1001010010111000   clv
1001010010011000   clz
1001010000001000   sec
1001010001011000   seh
1001010001111000   sei
1001010000101000   sen
1001010001001000   ses
1001010001101000   set
1001010000111000   sev
1001010000011000   sez
100101001SSS1000   bclr    S
100101000SSS1000   bset    S
1001010100001001   icall
1001010000001001   ijmp
1001010111001000   lpm     ?
1001000ddddd010+   lpm     r,z
1001010111011000   elpm    ?
1001000ddddd011+   elpm    r,z
0000000000000000   nop
1001010100001000   ret
1001010100011000   reti
1001010110001000   sleep
1001010110011000   break
1001010110101000   wdr
1001010111101000   spm
000111rdddddrrrr   adc     r,r
000011rdddddrrrr   add     r,r
001000rdddddrrrr   and     r,r
000101rdddddrrrr   cp      r,r
000001rdddddrrrr   cpc     r,r
000100rdddddrrrr   cpse    r,r
001001rdddddrrrr   eor     r,r
001011rdddddrrrr   mov     r,r
100111rdddddrrrr   mul     r,r
001010rdddddrrrr   or      r,r
000010rdddddrrrr   sbc     r,r
000110rdddddrrrr   sub     r,r
001001rdddddrrrr   clr     r
000011rdddddrrrr   lsl     r
000111rdddddrrrr   rol     r
001000rdddddrrrr   tst     r
0111KKKKddddKKKK   andi    d,M
0111KKKKddddKKKK   cbr     d,n
1110KKKKddddKKKK   ldi     d,M
11101111dddd1111   ser     d
0110KKKKddddKKKK   ori     d,M
0110KKKKddddKKKK   sbr     d,M
0011KKKKddddKKKK   cpi     d,M
0100KKKKddddKKKK   sbci    d,M
0101KKKKddddKKKK   subi    d,M
1111110rrrrr0sss   sbrc    r,s
1111111rrrrr0sss   sbrs    r,s
```

```
1111100ddddd0sss   bld     r,s
1111101ddddd0sss   bst     r,s
10110PPdddddPPPP   in      r,P
10111PPrrrrrPPPP   out     P,r
10010110KKddKKKK   adiw    w,K
10010111KKddKKKK   sbiw    w,K
10011000pppppsss   cbi     p,s
10011010pppppsss   sbi     p,s
10011001pppppsss   sbic    p,s
10011011pppppsss   sbis    p,s
1111011lllllll000   brcc    l
1111001lllllll000   brcs    l
1111001lllllll001   breq    l
1111011lllllll100   brge    l
1111011lllllll101   brhc    l
1111001lllllll101   brhs    l
1111011lllllll111   brid    l
1111001lllllll111   brie    l
1111001lllllll000   brlo    l
1111001lllllll100   brlt    l
1111001lllllll010   brmi    l
1111011lllllll001   brne    l
1111011lllllll010   brpl    l
1111011lllllll000   brsh    l
1111011lllllll110   brtc    l
1111001lllllll110   brts    l
1111011lllllll011   brvc    l
1111001lllllll011   brvs    l
1111011lllllllsss   brbc    s,l
1111001lllllllsss   brbs    s,l
1101LLLLLLLLLLLL   rcall   L
1100LLLLLLLLLLLL   rjmp    L
1001010hhhhh111h   call    h
1001010hhhhh110h   jmp     h
1001010rrrrr0101   asr     r
1001010rrrrr0000   com     r
1001010rrrrr1010   dec     r
1001010rrrrr0011   inc     r
1001010rrrrr0110   lsr     r
1001010rrrrr0001   neg     r
1001000rrrrr1111   pop     r
1001001rrrrr1111   push    r
1001010rrrrr0111   ror     r
1001010rrrrr0010   swap    r
00000001ddddrrrr   movw    v,v
00000010ddddrrrr   muls    d,d
000000110ddd0rrr   mulsu   a,a
000000110ddd1rrr   fmul    a,a
000000111ddd0rrr   fmuls   a,a
000000111ddd1rrr   fmulsu  a,a
1001001ddddd0000   sts     i,r
1001000ddddd0000   lds     r,i
10o0oo0ddddbooo   ldd     r,b
100!000ddddbee-+   ld      r,e
10o0oo1rrrrrbooo   std     b,r
100!001rrrrree-+   st      e,r
1001010100011001   eicall
1001010000011001   eijmp
```