

# AVRDUDE

---

A program for download/uploading AVR microcontroller flash and eeprom.  
For AVRDUDE, Version 4.2.0, 4 September 2003.

by Brian S. Dean

---

(Send bugs and comments on AVRDUDE to [avrdude-dev@nongnu.org](mailto:avrdude-dev@nongnu.org).)

Copyright © 2003 Brian S. Dean

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Free Software Foundation.

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	History and Credits .....	1
<b>2</b>	<b>Command Line Options .....</b>	<b>2</b>
2.1	Option Descriptions .....	2
2.2	Example Command Line Invocations .....	5
<b>3</b>	<b>Terminal Mode Operation .....</b>	<b>9</b>
3.1	Terminal Mode Commands .....	9
3.2	Terminal Mode Examples .....	9
<b>4</b>	<b>Configuration File .....</b>	<b>13</b>
4.1	AVRDUDE Defaults .....	13
4.2	Programmer Definitions .....	13
4.3	Part Definitions .....	13
4.3.1	Instruction Format .....	14
4.4	Other Notes .....	15
<b>Appendix A Platform Dependent Information</b>		
	<b>.....</b>	<b>16</b>
A.1	Unix .....	16
A.1.1	Unix Installation .....	16
A.1.1.1	FreeBSD Installation .....	16
A.1.1.2	Linux Installation .....	16
A.1.2	Unix Configuration Files .....	16
A.1.2.1	FreeBSD Configuration Files .....	17
A.1.2.2	Linux Configuration Files .....	17
A.1.3	Unix Port Names .....	17
A.1.4	Unix Documentation .....	17
A.2	Windows .....	17
A.2.1	Installation .....	17
A.2.2	Configuration Files .....	18
A.2.2.1	Configuration file names .....	18
A.2.2.2	How AVRDUDE finds the configuration files .....	18
A.2.3	Port Names .....	18
A.2.3.1	Serial Ports .....	18
A.2.3.2	Parallel Ports .....	18
A.2.4	Using the parallel port .....	19
A.2.4.1	Windows NT/2K/XP .....	19
A.2.4.2	Windows 95/98 .....	19
A.2.5	Documentation .....	19
A.2.6	Credits .....	20

# 1 Introduction

AVRDUDE - AVR Downloader Uploader - is a program for downloading and uploading the on-chip memories of Atmel's AVR microcontrollers. It can program the Flash and EEPROM, and where supported by the serial programming protocol, it can program fuse and lock bits. AVRDUDE also supplies a direct instruction mode allowing one to issue any programming instruction to the AVR chip regardless of whether AVRDUDE implements that specific feature of a particular chip.

AVRDUDE can be used effectively via the command line to read or write all chip memory types (eeprom, flash, fuse bits, lock bits, signature bytes) or via an interactive (terminal) mode. Using AVRDUDE from the command line works well for programming the entire memory of the chip from the contents of a file, while interactive mode is useful for exploring memory contents, modifying individual bytes of eeprom, programming fuse/lock bits, etc.

AVRDUDE supports three basic programmer types: Atmel's STK500, appnote avr910 and the PPI (parallel port interface). PPI represents a class of simple programmers where the programming lines are directly connected to the PC parallel port. Several pin configurations exist for several variations of the PPI programmers, and AVRDUDE can be configured to work with them by either specifying the appropriate programmer on the command line or by creating a new entry in its configuration file. All that's usually required for a new entry is to tell AVRDUDE which pins to use for each programming function.

The STK500 and avr910 use the serial port to communicate with the PC and contains on-board logic to control the programming of the target device. The fundamental difference between the two types lies in the protocol used to control the programmer. The av910 protocol is very simplistic and can easily be used as the basis for a simple, home made programmer since the firmware is available online. On the other hand, the STK500 protocol is more robust and complicated and the firmware is not openly available.

## 1.1 History and Credits

AVRDUDE was written by Brian S. Dean under the name of AVRPROG to run on the FreeBSD Operating System. Brian renamed the software to be called AVRDUDE when interest grew in a Windows port of the software so that the name did not conflict with AVRPROG.EXE which is the name of Atmel's Windows programming software.

The AVRDUDE source now resides in the public CVS repository on savannah.gnu.org (<http://savannah.gnu.org/projects/avrdude/>), where it continues to be enhanced and ported to other systems. In addition to FreeBSD, AVRDUDE now runs on Linux and Windows. The developers behind the porting effort primarily were Ted Roth, Eric Weddington, and Joerg Wunsch.

And in the spirit of many open source projects, this manual also draws on the work of others. The initial revision was composed of parts of the original Unix manual page written by Joerg Wunsch, the original web site documentation by Brian Dean, and from the comments describing the fields in the AVRDUDE configuration file by Brian Dean. The texi formatting was modeled after that of the Simulavr documentation by Ted Roth.

## 2 Command Line Options

### 2.1 Option Descriptions

AVRDUDE is a command line tool, used as follows:

```
avrdude -p partno options ...
```

Command line options are used to control AVRDUDE's behaviour. The following options are recognized:

**-p *partno***

This is the only mandatory option and it tells AVRDUDE what type of part (MCU) that is connected to the programmer. The *partno* parameter is the part's id listed in the configuration file. Specify -p ? to list all parts in the configuration file. If a part is unknown to AVRDUDE, it means that there is no config file entry for that part, but it can be added to the configuration file if you have the Atmel datasheet so that you can enter the programming specifications. Currently, the following MCU types are understood:

1200	AT90S1200
2313	AT90S2313
2333	AT90S2333
2343	AT90S2343 (*)
4414	AT90S4414
4433	AT90S4433
4434	AT90S4434
8515	AT90S8515
8535	AT90S8535
m103	ATMEGA103
m128	ATMEGA128
m16	ATMEGA16
m163	ATMEGA163
m169	ATMEGA169
m32	ATMEGA32
m8	ATMEGA8
m8535	ATMEGA8535
t12	ATtiny12
t15	ATtiny15
t26	ATTINY26

(\*) The AT90S2323 uses the same algorithm.

**-c *programmer-id***

Specify the programmer to be used. AVRDUDE knows about several common programmers. Use this option to specify which one to use. The *programmer-id* parameter is the programmer's id listed in the configuration file. Specify -c ? to list all programmers in the configuration file. If you have a programmer that is unknown to AVRDUDE, and the programmer is controlled via the PC parallel

port, there's a good chance that it can be easily added to the configuration file without any code changes to AVRDUDE. Simply copy an existing entry and change the pin definitions to match that of the unknown programmer. Currently, the following programmer ids are understood and supported:

<b>alf</b>	Nightshade ALF-PgmAVR, <a href="http://nightshade.homeip.net/">http://nightshade.homeip.net/</a>
<b>avr910</b>	Atmel Low Cost Serial Programmer
<b>avrisp</b>	Atmel AVR ISP
<b>bascom</b>	Bascom SAMPLE programming cable
<b>bsd</b>	Brian Dean's Programmer, <a href="http://www.bsddhome.com/avrdude/">http://www.bsddhome.com/avrdude/</a>
<b>dt006</b>	Dontronics DT006
<b>pavr</b>	Jason Kyle's pAVR Serial Programmer
<b>picoweb</b>	Picoweb Programming Cable, <a href="http://www.picoweb.net/">http://www.picoweb.net/</a>
<b>pony</b>	stk200 = Pony Prog STK200
<b>sp12</b>	Steve Bolt's Programmer
<b>stk200</b>	STK200
<b>stk500</b>	Atmel STK500

#### **-C *config-file***

Use the specified config file for configuration data. This file contains all programmer and part definitions that AVRDUDE knows about. If you have a programmer or part that AVRDUDE does not know about, you can add it to the config file (be sure and submit a patch back to the author so that it can be incorporated for the next version). If not specified, AVRDUDE reads the configuration file from `/usr/local/etc/avrdude.conf` (FreeBSD and Linux). See Appendix A for the method of searching for the configuration file for Windows.

**-e** Causes a chip erase to be executed. This will reset the contents of the flash ROM and EEPROM to the value '0xff', and is basically a prerequisite command before the flash ROM can be reprogrammed again. The only exception would be if the new contents would exclusively cause bits to be programmed from the value '1' to '0'. Note that in order to reprogram EEPROM cells, no explicit prior chip erase is required since the MCU provides an auto-erase cycle in that case before programming the cell.

#### **-E *exitspec* [, ...]**

By default, AVRDUDE leaves the parallel port in the same state at exit as it has been found at startup. This option modifies the state of the '/RESET' and 'Vcc' lines the parallel port is left at, according to the *exitspec* arguments provided, as follows:

<b>reset</b>	The '/RESET' signal will be left activated at program exit, that is it will be held low, in order to keep the MCU in reset state afterwards. Note in particular that the programming algorithm for the AT90S1200 device mandates that the '/RESET' signal is active before powering up the MCU, so in case an external power supply is used for this MCU type, a previous invocation of AVRDUDE
--------------	---

	with this option specified is one of the possible ways to guarantee this condition.
<b>noreset</b>	The ‘/RESET’ line will be deactivated at program exit, thus allowing the MCU target program to run while the programming hardware remains connected.
<b>vcc</b>	This option will leave those parallel port pins active (i. e. high) that can be used to supply ‘Vcc’ power to the MCU.
<b>novcc</b>	This option will pull the ‘Vcc’ pins of the parallel port down at program exit.

Multiple *exitspec* arguments can be separated with commas.

#### **-f *format***

This option specifies the file format for the input or output files to be processed. Format can be one of:

<b>i</b>	Intel Hex
<b>s</b>	Motorola S-record
<b>r</b>	raw binary; little-endian byte order, in the case of the flash ROM data
<b>m</b>	immediate mode; actual byte values specified on the command line, separated by commas or spaces in place of the <i>filename</i> field of the ‘-i’, ‘-o’, or ‘-U’ options. This is useful for programming fuse bytes without having to create a single-byte file or enter terminal mode. If the number specified begins with 0x, it is treated as a hex value. If the number otherwise begins with a leading zero (0) it is treated as octal. Otherwise, the value is treated as decimal.
<b>a</b>	auto detect; valid for input only, and only if the input is not provided at stdin.

The default is to use auto detection for input files, and raw binary format for output files.

**-F** Normally, AVRDUDE tries to verify that the device signature read from the part is reasonable before continuing. Since it can happen from time to time that a device has a broken (erased or overwritten) device signature but is otherwise operating normally, this options is provided to override the check.

#### **-i *filename***

Specifies the input file to be programmed into the MCU. Can be specified as ‘-’ to use stdin as the input.

#### **-m *memtype***

Specifies which program area of the MCU to read or write; allowable values depend on the MCU being programmed, but most support at least **eeeprom** for the EEPROM, and **flash** for the flash ROM. Use the ‘-v’ option on the command line or the **part** command from terminal mode to display all the memory types supported by a particular device. The default is **flash**.

**-n** No-write - disables actually writing data to the MCU (useful for debugging AVRDUDE).

- o *filename***  
Specifies the name of the output file to write, and causes the respective memory area to be read from the MCU. Can be specified as '-' to write to stdout.
- P *port***  
Use *port* to identify the device to which the programmer is attached. By default the `/dev/ppi0` port is used, but if the programmer type normally connects to the serial port, the `/dev/cuaa0` port is the default. If you need to use a different parallel or serial port, use this option to specify the alternate port name.
- q**  
Disable (or quell) output of the progress bar while reading or writing to the device.
- t**  
Tells AVRDUDE to enter the interactive “terminal” mode instead of up- or downloading files. See below for a detailed description of the terminal mode.
- U *memtype:op:filename[:format]***  
Perform a memory operation, equivalent to specifying the '-m', '-i' or '-o', and '-f' options, except that multiple '-U' options can be specified in order to operate on multiple memories on the same command-line invocation. The *memtype* field specifies the memory type to operate on. The *op* field specifies what operation to perform:
- |          |  |
|----------|--|
| <b>r</b> | read the specified device memory and write to the specified file                       |
| <b>w</b> | read the specified file and write it to the specified device memory                    |
| <b>v</b> | read the specified device memory and the specified file and perform a verify operation |
- The *filename* field indicates the name of the file to read or write. The *format* field is optional and contains the format of the file to read or write. See the '-f' option for possible values.
- v**  
Enable verbose output.
- V**  
Disable automatic verify check when uploading data.
- y**  
Tells AVRDUDE to use the last four bytes of the connected parts' EEPROM memory to track the number of times the device has been erased. When this option is used and the '-e' flag is specified to generate a chip erase, the previous counter will be saved before the chip erase, it is then incremented, and written back after the erase cycle completes. Presumably, the device would only be erased just before being programmed, and thus, this can be utilized to give an indication of how many erase-rewrite cycles the part has undergone. Since the FLASH memory can only endure a finite number of erase-rewrite cycles, one can use this option to track when a part is nearing the limit. The typical limit for Atmel AVR FLASH is 1000 cycles. Of course, if the application needs the last four bytes of EEPROM memory, this option should not be used.
- Y *cycles***  
Instructs AVRDUDE to initialize the erase-rewrite cycle counter residing at the last four bytes of EEPROM memory to the specified value. If the application needs the last four bytes of EEPROM memory, this option should not be used.



## 2.2 Example Command Line Invocations

Download the file `diag.hex` to the ATmega128 chip using the STK500 programmer connected to the default serial port:

```
% avrdude -p m128 -c stk500 -e -U flash:w:diag.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.03s

avrdude: Device signature = 0x1e9702
avrdude: erasing chip
avrdude: done.
avrdude: performing op: 1, flash, 0, diag.hex
avrdude: reading input file "diag.hex"
avrdude: input file diag.hex auto detected as Intel Hex
avrdude: writing flash (19278 bytes):

Writing | ##### | 100% 7.60s

avrdude: 19456 bytes of flash written
avrdude: verifying flash memory against diag.hex:
avrdude: load data flash data from input file diag.hex:
avrdude: input file diag.hex auto detected as Intel Hex
avrdude: input file diag.hex contains 19278 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 6.83s

avrdude: verifying ...
avrdude: 19278 bytes of flash verified

avrdude done. Thank you.

%
```

Upload the flash memory from the ATmega128 connected to the STK500 programmer and save it in raw binary format in the file named `diag.flash`:

```
% avrdude -p m128 -c stk500 -U flash:r:diag.flash:r

avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.03s█

avrdude: Device signature = 0x1e9702
avrdude: reading flash memory:

Reading | ##### | 100% 46.10s█

avrdude: writing output file "diag.flash"

avrdude done.  Thank you.

%
```

Using the default programmer, download the file `diag.hex` to flash, `eeeprom.hex` to EEPROM, and set the Extended, High, and Low fuse bytes to `0xff`, `0x89`, and `0x2e` respectively:

```
% avrdude -p m128 -U flash:w:diag.hex \  
>             -U eeprom:w:eeprom.hex \  
>             -U efuse:w:0xff:m      \  
>             -U hfuse:w:0x89:m      \  
>             -U lfuse:w:0x2e:m  
  
avrdude: AVR device initialized and ready to accept instructions  
  
Reading | ##### | 100% 0.03s  
  
avrdude: Device signature = 0x1e9702  
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed  
        To disable this feature, specify the -D option.  
avrdude: erasing chip  
avrdude: reading input file "diag.hex"  
avrdude: input file diag.hex auto detected as Intel Hex  
avrdude: writing flash (19278 bytes):  
  
Writing | ##### | 100% 7.60s  
  
avrdude: 19456 bytes of flash written  
avrdude: verifying flash memory against diag.hex:  
avrdude: load data flash data from input file diag.hex:  
avrdude: input file diag.hex auto detected as Intel Hex  
avrdude: input file diag.hex contains 19278 bytes  
avrdude: reading on-chip flash data:  
  
Reading | ##### | 100% 6.84s  
  
avrdude: verifying ...  
avrdude: 19278 bytes of flash verified  
  
[ ... other memory status output skipped for brevity ... ]  
  
avrdude done.  Thank you.  
  
%
```

## 3 Terminal Mode Operation

AVRDUDE has an interactive mode called *terminal mode* that is enabled by the ‘-t’ option. This mode allows one to enter interactive commands to display and modify the various device memories, perform a chip erase, display the device signature bytes and part parameters, and to send raw programming commands. Commands and parameters may be abbreviated to their shortest unambiguous form. Terminal mode also supports a command history so that previously entered commands can be recalled and edited.

### 3.1 Terminal Mode Commands

The following commands are implemented:

<code>dump memtype addr nbytes</code>	Read <i>nbytes</i> from the specified memory area, and display them in the usual hexadecimal and ASCII form.
<code>dump</code>	Continue dumping the memory contents for another <i>nbytes</i> where the previous dump command left off.
<code>write memtype addr byte1 ... byteN</code>	Manually program the respective memory cells, starting at address <i>addr</i> , using the values <i>byte1</i> through <i>byteN</i> . This feature is not implemented for bank-addressed memories such as the flash memory of ATMega devices.
<code>erase</code>	Perform a chip erase.
<code>send b1 b2 b3 b4</code>	Send raw instruction codes to the AVR device. If you need access to a feature of an AVR part that is not directly supported by AVRDUDE, this command allows you to use it, even though AVRDUDE does not implement the command.
<code>sig</code>	Display the device signature bytes.
<code>part</code>	Display the current part settings.
<code>?</code>	
<code>help</code>	Give a short on-line summary of the available commands.
<code>quit</code>	Leave terminal mode and thus AVRDUDE.

In addition, the following commands are supported on the STK500 programmer:

<code>vtarg voltage</code>	Set the target’s supply voltage to <i>voltage</i> Volts.
<code>varef voltage</code>	Set the adjustable voltage source to <i>voltage</i> Volts. This voltage is normally used to drive the target’s <i>Aref</i> input on the STK500.
<code>fosc freq[M k]</code>	Set the master oscillator to <i>freq</i> Hz. An optional trailing letter <i>M</i> multiplies by 1E6, a trailing letter <i>k</i> by 1E3.
<code>fosc off</code>	Turn the master oscillator off.
<code>parms</code>	Display the current voltage and master oscillator parameters.

## 3.2 Terminal Mode Examples

Display part parameters, modify eeprom cells, perform a chip erase:

```
% avrdude -p m128 -c stk500 -t
```

```
avrdude: AVR device initialized and ready to accept instructions
avrdude: Device signature = 0x1e9702
avrdude: current erase-rewrite cycle count is 52 (if being tracked)
avrdude> part
>>> part
```

```
AVR Part           : ATMEGA128
Chip Erase delay   : 9000 us
PAGEL              : PD7
BS2                : PA0
RESET disposition  : dedicated
RETRY pulse        : SCK
serial program mode : yes
parallel program mode : yes
Memory Detail      :
```

Memory	Type	Paged	Size	Page Size	#Pages	MinW	MaxW	Polled ReadBack
-----	-----	-----	-----	-----	-----	-----	-----	-----
eeeprom	no		4096	8	0	9000	9000	0xff 0xff
flash	yes		131072	256	512	4500	9000	0xff 0x00
lfuse	no		1	0	0	0	0	0x00 0x00
hfuse	no		1	0	0	0	0	0x00 0x00
efuse	no		1	0	0	0	0	0x00 0x00
lock	no		1	0	0	0	0	0x00 0x00
calibration	no		1	0	0	0	0	0x00 0x00
signature	no		3	0	0	0	0	0x00 0x00

```
avrdude> dump eeprom 0 16
```

```
>>> dump eeprom 0 16
```

```
0000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.....|■
```

```
avrdude> write eeprom 0 1 2 3 4
```

```
>>> write eeprom 0 1 2 3 4
```

```
avrdude> dump eeprom 0 16
```

```
>>> dump eeprom 0 16
```

```
0000  01 02 03 04 ff ff ff ff  ff ff ff ff ff ff ff ff  |.....|■
```

```
avrdude> erase
```

```
>>> erase
```

```
avrdude: erasing chip
```

```
avrdude> dump eeprom 0 16
```

```
>>> dump eeprom 0 16
```

```
0000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |.....|■
```

```
avrdude>
```

Program the fuse bits of an ATmega128 (disable M103 compatibility, enable high speed external crystal, enable brown-out detection, slowly rising power). First display the factory defaults, then reprogram:

```
% avrdude -p m128 -c stk500 -t

avrdude: AVR device initialized and ready to accept instructions
avrdude: Device signature = 0x1e9702
avrdude: current erase-rewrite cycle count is 52 (if being tracked)
avrdude> d efuse
>>> d efuse
0000 fd |. |■

avrdude> d hfuse
>>> d hfuse
0000 99 |. |■

avrdude> d lfuse
>>> d lfuse
0000 e1 |. |■

avrdude> w efuse 0 0xff
>>> w efuse 0 0xff

avrdude> w hfuse 0 0x89
>>> w hfuse 0 0x89

avrdude> w lfuse 0 0x2f
>>> w lfuse 0 0x2f

avrdude>
```

## 4 Configuration File

AVRDUDE reads a configuration file upon startup which describes all of the parts and programmers that it knows about. The advantage of this is that if you have a chip that is not currently supported by AVRDUDE, you can add it to the configuration file without waiting for a new release of AVRDUDE. Likewise, if you have a parallel port programmer that is not supported by AVRDUDE, chances are good that you can copy an existing programmer definition, and with only a few changes, make your programmer work with AVRDUDE.

AVRDUDE first looks for a system wide configuration file in a platform dependent location. On Unix, this is usually `/usr/local/etc/avrdude.conf`, while on Windows it is usually in the same location as the executable file. The name of this file can be changed using the `‘-C’` command line option. After the system wide configuration file is parsed, AVRDUDE looks for a per-user configuration file to augment or override the system wide defaults. On Unix, the per-user file is `.avrduderc` within the user’s home directory. On Windows, this file is the `avrdude.rc` file located in the same directory as the executable.

### 4.1 AVRDUDE Defaults

```
default_parallel = "default-parallel-device";
```

Assign the default parallel port device. Can be overridden using the `‘-P’` option.

```
default_serial = "default-serial-device";
```

Assign the default serial port device. Can be overridden using the `‘-P’` option.

```
default_programmer = "default-programmer-id";
```

Assign the default programmer id. Can be overridden using the `‘-c’` option.

### 4.2 Programmer Definitions

The format of the programmer definition is as follows:

```
programmer
  id      = <id1> [, <id2> [, <id3>] ...] ;    # <idN> are quoted strings
  desc    = <description> ;                      # quoted string
  type    = par | stk500 ;                       # programmer type
  vcc     = <num1> [, <num2> ... ] ;             # pin number(s)
  reset   = <num> ;                               # pin number
  sck     = <num> ;                               # pin number
  mosi    = <num> ;                               # pin number
  miso    = <num> ;                               # pin number
  errled  = <num> ;                               # pin number
  rdyled  = <num> ;                               # pin number
  pgmled  = <num> ;                               # pin number
  vfyld   = <num> ;                               # pin number
;
```



## 4.3 Part Definitions

```

part
    id                = <id> ;                # quoted string
    desc              = <description> ;        # quoted string
    devicecode        = <num> ;                # numeric
    chip_erase_delay  = <num> ;                # micro-seconds
    page1             = <num> ;                # pin name in hex, i.e., 0xD7
    bs2               = <num> ;                # pin name in hex, i.e., 0xA0
    reset             = dedicated | io;
    retry_pulse       = reset | sck;
    pgm_enable        = <instruction format> ;
    chip_erase        = <instruction format> ;
    memory <memtype>
        paged         = <yes/no> ;            # yes / no
        size          = <num> ;                # bytes
        page_size     = <num> ;                # bytes
        num_pages     = <num> ;                # numeric
        min_write_delay = <num> ;            # micro-seconds
        max_write_delay = <num> ;            # micro-seconds
        readback_p1   = <num> ;                # byte value
        readback_p2   = <num> ;                # byte value
        pwroff_after_write = <yes/no> ;      # yes / no
        read          = <instruction format> ;
        write         = <instruction format> ;
        read_lo       = <instruction format> ;
        read_hi       = <instruction format> ;
        write_lo      = <instruction format> ;
        write_hi      = <instruction format> ;
        loadpage_lo   = <instruction format> ;
        loadpage_hi   = <instruction format> ;
        writepage     = <instruction format> ;
    ;
;

```

### 4.3.1 Instruction Format

Instruction formats are specified as a comma separated list of string values containing information (bit specifiers) about each of the 32 bits of the instruction. Bit specifiers may be one of the following formats:

- 1           The bit is always set on input as well as output
- 0           the bit is always clear on input as well as output
- x           the bit is ignored on input and output
- a           the bit is an address bit, the bit-number matches this bit specifier's position within the current instruction byte

<b>a</b> <i>N</i>	the bit is the <i>N</i> th address bit, bit-number = <i>N</i> , i.e., <b>a12</b> is address bit 12 on input, <b>a0</b> is address bit 0.
<b>i</b>	the bit is an input data bit
<b>o</b>	the bit is an output data bit

Each instruction must be composed of 32 bit specifiers. The instruction specification closely follows the instruction data provided in Atmel's data sheets for their parts. For example, the EEPROM read and write instruction for an AT90S2313 AVR part could be encoded as:

```

read  = "1  0  1  0    0  0  0  0    x x x x  x x x x",
        "x a6 a5 a4  a3 a2 a1 a0    o o o o  o o o o";

write = "1  1  0  0    0  0  0  0    x x x x  x x x x",
        "x a6 a5 a4  a3 a2 a1 a0    i i i i  i i i i";

```

## 4.4 Other Notes

- The **devicecode** parameter is the device code used by the STK500 and are obtained from the software section (**avr061.zip** of Atmel's AVR061 application note available from <http://www.atmel.com/atmel/acrobat/doc2525.pdf>).
- Not all memory types will implement all instructions.
- AVR Fuse bits and Lock bits are implemented as a type of memory.
- Example memory types are: **flash**, **eeprom**, **fuse**, **lfuse** (low fuse), **hfuse** (high fuse), **efuse** (extended fuse), **signature**, **calibration**, **lock**.
- The memory type specified on the AVRDUDE command line must match one of the memory types defined for the specified chip.
- The **pwroff\_after\_write** flag causes AVRDUDE to attempt to power the device off and back on after an unsuccessful write to the affected memory area if VCC programmer pins are defined. If VCC pins are not defined for the programmer, a message indicating that the device needs a power-cycle is printed out. This flag was added to work around a problem with the at90s4433/2333's; see the at90s4433 errata at: <http://www.atmel.com/atmel/acrobat/doc1280.pdf>

## Appendix A Platform Dependent Information

### A.1 Unix

#### A.1.1 Unix Installation

To build and install from the source tarball on Unix like systems:

```
$ gunzip -c avrdude-4.2.0.tar.gz | tar xf -  
$ cd avrdude-4.2.0  
$ ./configure  
$ make  
$ su root -c 'make install'
```

The default location of the install is into `/usr/local` so you will need to be sure that `/usr/local/bin` is in your `PATH` environment variable.

If you do not have root access to your system, you can do the the following instead:

```
$ gunzip -c avrdude-4.2.0.tar.gz | tar xf -  
$ cd avrdude-4.2.0  
$ ./configure --prefix=$HOME/local  
$ make  
$ make install
```

##### A.1.1.1 FreeBSD Installation

AVRDUDE is installed via the FreeBSD Ports Tree as follows:

```
% su - root  
# cd /usr/ports/devel/avrdude  
# make install
```

If you wish to install from a pre-built package instead of the source, you can use the following instead:

```
% su - root  
# pkg_add -r avrdude
```

Of course, you must be connected to the Internet for these methods to work, since that is where the source as well as the pre-built package is obtained.

##### A.1.1.2 Linux Installation

On rpm based linux systems (such as RedHat, SUSE, Mandrake, etc), you can build and install the rpm binaries directly from the tarball:

```
$ su - root  
# rpmbuild -tb avrdude-4.2.0.tar.gz  
# rpm -Uvh /usr/src/redhat/RPMS/i386/avrdude-4.2.0-1.i386.rpm
```

Note that the path to the resulting rpm package, differs from system to system. The above example is specific to RedHat.

### A.1.2 Unix Configuration Files

When AVRDUDE is build using the default ‘`--prefix`’ configure option, the default configuration file for a Unix system is located at `/usr/local/etc/avrdude.conf`. This can be overridden by using the ‘`-C`’ command line option. Additionally, the user’s home directory is searched for a file named `.avrduderc`, and if found, is used to augment the system default configuration file.

#### A.1.2.1 FreeBSD Configuration Files

When AVRDUDE is installed using the FreeBSD ports system, the system configuration file is always `/usr/local/etc/avrdude.conf`.

#### A.1.2.2 Linux Configuration Files

When AVRDUDE is installed using from an rpm package, the system configuration file will be always be `/etc/avrdude.conf`.

### A.1.3 Unix Port Names

The parallel and serial port device file names are system specific. The following table lists the default names for a given system.

System	Default Parallel Port	Default Serial Port
FreeBSD	<code>/dev/ppi0</code>	<code>/dev/cuaa0</code>
Linux	<code>/dev/parport0</code>	<code>/dev/ttyS0</code>

On FreeBSD systems, AVRDUDE uses the `ppi(4)` interface for accessing the parallel port and the `sio(4)` driver for serial port access.

On Linux systems, AVRDUDE uses the `ppdev` interface for accessing the parallel port and the `tty` driver for serial port access.

### A.1.4 Unix Documentation

AVRDUDE installs a manual page as well as info, HTML and PDF documentation. The manual page is installed in `/usr/local/man/man1` area, while the HTML and PDF documentation is installed in `/usr/local/share/doc/avrdude` directory. The info manual is installed in `/usr/local/info/avrdude.info`.

Note that these locations can be altered by various configure options such as ‘`--prefix`’.

## A.2 Windows

### A.2.1 Installation

A Windows executable of avrdude is included in WinAVR which can be found at <http://sourceforge.net/projects/winavr>. WinAVR is a suite of executable, open source software development tools for the AVR for the Windows platform.

To build avrdude from the source You must have Cygwin (<http://www.cygwin.com/>).

To build and install from the source tarball for Windows (using Cygwin):

```
$ set PREFIX=<your install directory path>
$ export PREFIX
$ gunzip -c avrdude-4.2.0.tar.gz | tar xf -
$ cd avrdude-4.2.0
$ ./configure LDFLAGS="-static" --prefix=$PREFIX --datadir=$PREFIX
--sysconfdir=$PREFIX/bin --enable-versioned-doc=no
$ make
$ make install
```

### A.2.2 Configuration Files

#### A.2.2.1 Configuration file names

AVRDUDE on Windows looks for a system configuration file name of `avrdude.conf` and looks for a user override configuration file of `avrdude.rc`.

#### A.2.2.2 How AVRDUDE finds the configuration files.

AVRDUDE on Windows has a different way of searching for the system and user configuration files. Below is the search method for locating the configuration files:

1. The directory from which the application loaded.
2. The current directory.
3. The Windows system directory. On Windows NT, the name of this directory is `SYSTEM32`.
4. Windows NT: The 16-bit Windows system directory. The name of this directory is `SYSTEM`.
5. The Windows directory.
6. The directories that are listed in the `PATH` environment variable.

### A.2.3 Port Names

#### A.2.3.1 Serial Ports

When you select a serial port (i.e. when using an STK500) use the Windows serial port device names such as: `com1`, `com2`, etc.

### A.2.3.2 Parallel Ports

AVRDUDE will only accept 3 Windows parallel port names: lpt1, lpt2, or lpt3. Each of these names corresponds to a fixed parallel port base address:

<code>lpt1</code>	<code>0x378</code>
<code>lpt2</code>	<code>0x278</code>
<code>lpt3</code>	<code>0x3BC</code>

On your desktop PC, lpt1 will be the most common choice. If you are using a laptop, you might have to use lpt3 instead of lpt1. Select the name of the port the corresponds to the base address of the parallel port that you want.

### A.2.4 Using the parallel port

#### A.2.4.1 Windows NT/2K/XP

On Windows NT, 2000, and XP user applications cannot directly access the parallel port. However, kernel mode drivers can access the parallel port. giveio.sys is a driver that can allow user applications to set the state of the parallel port pins.

Before using AVRDUDE, the giveio.sys driver must be loaded. The accompanying command-line program, loaddrv.exe, can do just that.

To make things even easier there are 3 batch files that are also included:

1. `install_giveio.bat` Install and start the giveio driver.
2. `status_giveio.bat` Check on the status of the giveio driver.
3. `remove_giveio.bat` Stop and remove the giveio driver from memory.

These 3 batch files calls the loaddrv program with various options to install, start, stop, and remove the driver.

When you first execute `install_giveio.bat`, `loaddrv.exe` and `giveio.sys` must be in the current directory. When `install_giveio.bat` is executed it will copy `giveio.sys` from your current directory to your Windows directory. It will then load the driver from the Windows directory. This means that after the first time `install_giveio` is executed, you should be able to subsequently execute the batch file from any directory and have it successfully start the driver.

Note that you must have administrator privilege to load the giveio driver.

#### A.2.4.2 Windows 95/98

On Windows 95 and 98 the giveio.sys driver is not needed.

### A.2.5 Documentation

AVRDUDE installs a manual page as well as info, HTML and PDF documentation. The manual page is installed in `/usr/local/man/man1` area, while the HTML and PDF documentation is installed in `/usr/local/share/doc/avrdude` directory. The info manual is installed in `/usr/local/info/avrdude.info`.

Note that these locations can be altered by various configure options such as ‘`--prefix`’ and ‘`--datadir`’.

### A.2.6 Credits.

Thanks to:

- Dale Roberts for the giveio driver.
- Paula Tomlinson for the loaddrv sources.
- Chris Liechti <cliechti@gmx.net> for modifying loaddrv to be command line driven and for writing the batch files.