The Assemby Basics (copy)

Introduction

This Cheat Sheet is part of the Bladabuska's AVR Cheat Sheet Collection. This cheat sheet shows the basic syntax of the Assembly language for AVRs. All information in this cheat sheet is related to the AVR Assembler v2.0 from Microchip Technology Incorporated (formerly Atmel Corporation). The Assembly language for AVR microcontrollers is not case sensitive.

Note: If you use a different assembler software, you must adapt the information to your software.

The Assemby Basics

Comment

Everything between a semicolon (;) and the end of the line is a comment and is ignored by the Assembler software. Comments are primarily used for code documentation, and to disable code sections for debugging purpose.

Line continuation

A backslash character (\) placed at the end of a line is used to inform the preprocessor and the Assembler that the command continues on the next line.

Label Field

A label is a text identifier that starts with a letter (a-z or A-Z), a question mark (?) or an underscore character (_), followed by zero or more letters, numbers, dollar signs ($), question marks (?), or underscore characters (_). Labels cannot contain spaces and must end with a colon character (:).

Integer constant

Integer constants starts by a number or a radix specifier. If an integer starts with a non-zero number character (1-9), than no radix is specified and the integer is considered to be expressed in the decimal number system. Integers in binary notation must start with "-0b", in hexadecimal notation with "0x" and in octal notation with a leading zero (0). No spaces are allowed inside the number, but underscore characters (_) can be placed inside the number to increase readability..

Character constants

Character constants are characters or escape sequences enclosed in single quotes ('). They and can be used anywhere an integer expression is allowed.

sequence	name	symbol	number
\0	Null Character	NUL	0x00
\a	Alert Bell	BEL	0x07
\b	Backspace	BS	0x08
\t	Horizontal Tab	HT	0x09
\n	Line Feed "or newline"	LF	0x0A
\v	Vertical Tab	VT	0x0B
\f	Form Feed	FF	0x0C
\r	Carriage Return	CR	0x0D
\\	Backslash	\	0x5C
\xHH	Hexadecimal Number	Where *H* is the digit in hexadecimal notation (0-9,A-F)	
\000	Octal Number	Where *o* is the digit in octal notation (0-7)	

String constant

A string[7] is any number of characters enclosed in double quotes (") taken literally, no escape sequences are recognized, and it is not NULL-terminated. Quoted strings may be concatenated according to the ANSI C convention (space character) or with the backslash character (line continuation) to form long strings and multiple line-s-panning strings.

[7] Strings can only be used in conjunction with the **DB** directive or with the **MESSAGE**, **WARNING** or **ERROR** directives.

Operators

An operator is a (single or multiple) character(s) that executes an operation in one or more operands. Unary operators takes only one operand, placed after the operator. Binary operators takes two operands and are placed between then. The ternary conditional operator takes three operands. Operators can be grouped with paranthesis to receive maximum precedence.

Operators (cont)

Operator	Operation	Precedence[3]
	Arithmetic Operators	
+	Addition	12
-	Subtraction	12
-	Unary minus	14
%	Modulo[4]	13
*	Multiplication	13
/	Division	13
	Bitwise Operators	
~	Bitwise NOT	14
&	Bitwise AND	8
|	Bitwise OR	6
^	Bitwise XOR	7
<<	Shift Left	11
>>	Shift Right	11
	Logic Operators	
!	Logical NOT	14
&&	Logical AND	5
||	Logical OR	4
	Relational Operators	
<	Less than	10
>=	Less than or equal to	10
>	Greater than	10
>=	Greater than or equal to	10
==	Equal to	
!=	Different from	
	Ternary Conditional Operator	
? :	Ternary Conditional[5]	3

[3] The higher the precedence, the higher the priority.
[4] This operator was introduced in AVR Assembler v2.0.
[5] This operator was introduced in AVR Assembler v2.1.

Assembler directive

Assembler directives starts with a dot (.). They are used to guide the Assembler software on how to understand or implement some part of the code. Assembly directives can be used to define macros, code segments, conditional sections and symbols, to reserve memory space for data, and to implement debug features.[7]

[7] Refer to the **Assembler Directives Table** to a complete list of available Assembler directives.

Function

Functions are built-in macro functions that can be used to evaluate code.

LOW(expression)

> Returns the low byte (bits 7-0) of an expression.

HIGH(expression)

> Returns the second byte (bits 15-8) of an expression.

BYTE2(expression)

> Returns the second byte (bits 15-8) of an expression.

BYTE3(expression)

> Returns the third byte (bits 23-16) of an expression.

BYTE4(expression)

> Returns the fourth byte (bits 31-24) of an expression.

LWRD(expresion)

> Returns the low word (bits 15-0) of an expression.

HWRD(expresion)

> Returns the second word (bits 31-16) of an expression.

PAGE(expression)

> Returns bits 21-16 of an expression.

EXP2(expression)

> Returns 2 to the power of expression.

LOG2(expression)

> Returns the integer part of a log2(expression).

INT(expression)

> Truncates a floating point expression to integer.

FRAC(expression)

> Extracts fractional part of a floating point expression.

Function (cont)

Q7(expression)

Converts a fractional floating point expression to a form suitable for the multiplication instructions. (Sign + 7-bit fraction)

Q15(expression)

Converts a fractional floating point expression to a form suitable for the multiplication instructions. (Sign + 15-bit fraction)

ABS(expression)

Returns the absolute value of a constant expression.

DEFINED(symbol)

Returns 1 if symbol was previously defined (using .SET, .DEF, or .EQU directives), 0 otherwise. Normally used in conjunction with .IF and .ELIF directives. It does not require parentheses around its argument.

STRLEN(string)

Returns the length of a string constant, in bytes.

Assembler Directives Table

Source Segments

.CSEG[A]

Defines the start of a CODE segment. CODE segments have their own location counter (in words), starting at 0.

.DSEG[A]

Defines the start of a DATA segment. DATA segments have their own location counter (in bytes), starting at the first address after the I/O registers (0x60 or 0x100, depending on the number of peripherals).

.ESEG[A]

Defines the start of an EEPROM segment. EEPROM segments have their own location counter (in bytes), starting at 0.

.ORG <address>

Sets the location counter (of the segment were it was placed) to an absolute value.

Reserve Memory Space

.BYTE[LC] <number>

Reserves a number of BYTES in SRAM or EEPROM memories. The directive takes one parameter, which is the number of bytes to reserve.

Assembler Directives Table (cont)

.DB[LD] <list>

Reserves a number of BYTES in PROGRAM or EEPROM memory.

.DW[LD] <list>

Reserves a number of WORDS (2 BYTES) in PROGRAM or EEPROM memory.

.DD[LD] <list>

Reserves a number of DOUBLE-WORDS (4 BYTES) in PROGRAM or EEPROM memory.

.DQ[LD] <list>

Reserves a number of QUAD-WORDS (8 BYTES) in PROGRAM or EEPROM memory.

User Defined Symbols

.EQU <symbol>=<expression>

Assigns a user defined symbol to a value or expression. A symbol assigned to a value by the EQU directive is a constant and **cannot be** changed or redefined.

.SET <symbol>=<expression>

Assigns a user defined symbol to a value or expression. A symbol assigned to a value by the SET directive **can be** changed or redefined later in the program.

.DEF <symbol>=<register>

Assigns a user defined symbol to a register. A register can have several symbolic names attached to it. A symbol **can be** redefined later in the program.

.UNDEF <symbol>

Undefine a symbol previously defined with the DEF directive. This provides a way to obtain a simple scoping of register definitions, avoiding warnings about register reuse.

Macro Definition

.MACRO <name>

Defines the start of the macro. It takes the macro name as a parameter. A macro is marked with a + in the opcode field of the listfile.

.ENDMACRO/.ENDM

Defines the end of a macro definition. ENDM is a synonym to ENDMACRO directive.

Assembler Directives Table (cont)

File Management

.INCLUDE <file>

Tells the Assembler to start reading from a specified file. The Assembler then assembles the specified file until end of file (EOF) or an EXIT directive is encountered.

.EXIT

Tells the Assembler to stop assembling the file. If an EXIT directive appears in an included file, the Assembler continues from the line following the INCLUDE directive in the file containing the INCLUDE directive.

Conditional Assembly

.IFDEF <symbol>

If the symbol is defined, it will include code untill the corresponding ELSE directive.

.IFNDEF <symbol>

If the symbol is not defined, it will include code untill the corresponding ELSE directive.

.IF <expression>

If expression is evaluated different from 0, it will include code untill the corresponding ELSE, ELIF or ENDIF directive.

.ELIF <expression>

It will include code until the corresponding ENDIF (or the next ELIF at the same level), if the expression is true, and the initial IF clause and its following ELIF clauses (if any) are also false.

.ELSE

It will include code until the corresponding ENDIF, if the initial IF clause and its following ELIF clauses (if any) are also false.

.ENDIF

Defines the end for the conditional IF, IFDEF, or IFNDEF directives.

Assembler Program Output

.MESSAGE[I] <string>

Output a message string.

.WARNING[I] <string>

Outputs a warning string, but does not halt assembling.

Assembler Directives Table (cont)

.ERROR[I] <string>

Outputs an error message string and halts the assembling.

Listfile Generation Control

.LIST

Turn the listfile generation ON. The listfile is a combination of assembly source code, addresses, and opcodes. Listfile generation is turned on by default.

.NOLIST

Turn listfile generation OFF.

.LISTMAC

Turn macro expansion on. It tells the Assembler that when a macro is called, the expansion of the macro is to be shown on the listfile generated by the Assembler. The default is that only the macro-call with parameters is shown in the listfile.

Special Funcitons

.CSEGSIZE = <value>

This directive is used to specify the size of the program memory block at the SRAM memory. This directive can only be used with *AT94K Series Field Programmable System Level Integrated Circuit Devices*.

.OVERLAP[V]

This directive is for projects with special needs and should normally not be used.

.NOOVERLAP[V]

This directive is for projects with special needs and should normally not be used.

[A] All segments of the same type will be concatenated into one single segment of that type when assembled.
[C] Cannot be used inside CODE segments.
[D] Cannot be used inside DATA segments.
[I] My be used in conditional assembly,
[L] In order to be able to refer to the reserved location, the directive should be preceded by a LABEL.
[V] Introduced in AVR Assembler v2.1.

Preprocessor Directives

Preprocessor directives are lines whose first non-empty character is a hash symbol (#). They are used to issue commands to the preprocessor.

Macro Definition

#define <name> [<value>]

Define a preprocessor object-like macros that basically define a constant. If value is not specified, it is set to 1.

#define <name([<arg>, <...>])> [<value>]

Define a preprocessor function-like macros that do parameter substitution. This macro must be called with the same number of arguments it is defined with. The left parenthesis must appear immediately after name (no spaces between), otherwise it will be interpreted as part of the value of a object-like macro. If value is not specified, it is set to 1.

#undef <name>

Undefine macro name previously defined with a #define directive. If name is not previously defined, the .undef directive is silently ignored.

Conditional Assembly

#ifdef <name>

All the following lines until the corresponding #endif, #else, or #elif are conditionally assembled if name is previously #defined. Shorthand for #if defined (name).

#ifndef <name>

All the following lines until the corresponding #endif, #else, or #elif are conditionally assembled if name is not #defined. Shorthand for #if !defined (name).

#if <expression>

All the following lines until the corresponding #endif, #else, or #elif are conditionally assembled if expression evaluates to true (not equal to 0). Any undefined symbols used in expression are silently evaluated to 0.

Preprocessor Directives (cont)

#elif <expression>

All the following lines until the corresponding #endif, #else, or #elif are conditionally assembled if expression evaluates to true (not equal to 0),and all previous #if or #elif expressions were evaluated to false. Any undefined symbols used in expression are silently evaluated to 0.

#else

All the following lines until the corresponding #endif are conditionally assembled if all previous #if or #elif expressions were evaluated to false.

#endif

Terminates a conditional block initiated with an #if, #ifdef, or #ifndef directive.

Preprocessor Program Output

#message <string>

Outputs string to standard output, but does not affect assembler error or warning counters.

#warning <string>

Outputs string to standard error, and increments the assembler warning counter.

#error <string>

Outputs string to standard error, increments the assembler error counter, and prevents the program of being successfully assembled.

File Management

#include "file"

Include a file, searching in the current working directory first, then searching in the built-in known place.

#include <file>

Include a file, searching in the built-in known place only, unless the current working directory is explicitly specified with a dot (.) entry in the include path.

Empty Directive

#

Does nothing. The line is removed by the preprocessor.

Note: #pragma preprocessor directives will be treated in a separate topic.

Preprocessor Pre-defined Macros

__AVRASM_VERSION__

INTEGER. AVR Assembler version, encoded as (1000*major + minor)

__CORE_VERSION__

STRING. AVR core version.

__CORE_coreversion__

INTEGER. AVR core version value of the coreversion. For example: __CORE_V2__

__DATE__

STRING. Build date in the format "Jun 28 2004".

__TIME__

STRING. Build time in the format: "HH:MM:SS".

__CENTURY__

INTEGER. Build time century (tipically 20).

__YEAR__

INTEGER. Build time year, less century (0-99).

__MONTH__

INTEGER. Build time month (1-12).

__DAY__

INTEGER. Build time day (1-31).

__HOUR__

INTEGER. Build time hour (0-23).

__MINUTE__

INTEGER. Build time minute (0-59).

__SECOND__

INTEGER. Build time second (0-59).

__FILE__

STRING. Sorce file name.

__LINE__

INTEGER. Current line number in source file.

__PART_NAME__

STRING. AVR part name.

__partname__

INTEGER. AVR part name value of the partname For example: __ATmega8__

Preprocessor directive

Any line whose first non-empty character is a hash symbol (#). It is used to issue command to the preprocessor, a software that runs before calling the Assembler program.[2,3,4]

[2] Refer to the **Preprocessor Directives Table** for a complete list of available preprocessor directives.

[3] Refer to the **Preprocessor Pre-Defined Macro Table** for a complete list of available preprocessor pre-defined macros.

[4] Refer to the **Preprocessor Pragma Directives** for a complete list of available preprocessor pragma directives.