Microcontroller programming

Atmel AVRASM2

Quick Guide

```
#include <m128def.inc>

forever:    RJMP forever     ; and ever
```

http://onlinedocs.microchip.com AVR Assembler User Guide

AVRAS2 assembler

- Syntax
- Preprocessor
- Keywords

- Operators
- Directives
- Expressions
- Functions

- Known issues

Syntax

```
[label:] instruction [operands] [Comment]

[label:] directive [operands] [Comment]

Comment                    ; [Text]

(empty line)
```

Preprocessor

- Nearly as in C (with necessary differences)
- Directives start with **#**
- Operators **#** a **##**
- Predefined macros **`__SOMETHING__`**

Preprocessor directives

#define #undef

#error #warning #message

#if #else #elif #endif

#ifdef #ifndef

#include

#pragma

(empty directive)

(stringification)

(concatenation)

#pragma - general

#pragma warning range byte *option*

integer / overflow / none

#pragma overlap *option*

ignore / warning / error / default

#pragma error instruction

#pragma warning instruction

#pragma – AVR part related

#pragma AVRPART ADMIN PART_NAME *string*

V0 / V0E / V1 / V2 / V2E

#pragma AVRPART CORE CORE_VERSION *version_string*

#pragma AVRPART CORE INSTRUCTIONS_NOT_SUPPORTED *mnemonic[operand[,operand]][: …]*

#pragma AVRPART CORE NEW_INSTRUCTIONS *mnemonic[operand[,operand]][: …]*

#pragma AVRPART MEMORY PROG_FLASH *size*

#pragma AVRPART MEMORY EEPROM *size*

#pragma AVRPART MEMORY INT_SRAM *size*

#pragma AVRPART MEMORY INT_SRAM START_ADDR *address*

0x60 / 0x100

#pragma partinclude *num*

0 / 1

Operands

- Label
  - value of the location counter at that place
- Variable
  - SET directive
- Constant
  - user defined using EQU directive
  - integer
    decimal
    hexadecimal (0x… $...)
    binary (0b…)
    octal (0□□)
  - floating-point
- PC
  - current value of the Program memory location counter

Operator Precedence OLD

1.
2.
3. **? :** (conditional expression)
4. **||** (logical OR)
5. **&&** (logical AND)
6. **|** (bitwise OR)
7. **^** (bitwise XOR)
8. **&** (bitwise AND)
9. **==** (equal) **!=** (not equal)
10. **<** (less than) **<=** (less or equal) **>** (greater than) **>=** (greater or equal)
11. **<<** (shift left) **>>** (shift right)
12. **!** (unary logical NOT) **~** (unary bitwise NOT) **+** (addition) **-** (substraction)
13. ***** (multiplication) **/** (division) **%** (modulo)
14. **-** (unary minus)

Operator Precedence since AVRAS2.1

1.
2.
3. **? :** (conditional expression)
4. **||** (logical OR)
5. **&&** (logical AND)
6. **|** (bitwise OR)
7. **^** (bitwise XOR)
8. **&** (bitwise AND)
9. **==** (equal) **!=** (not equal)
10. **<** (less than) **<=** (less or equal) **>** (greater than) **>=** (greater or equal)
11. **<<** (shift left) **>>** (shift right)
12. **+** (addition) **-** (substraction)
13. ***** (multiplication) **/** (division) **%** (modulo)
14. **-** (unary minus) **!** (unary logical NOT) **~** (unary bitwise NOT)

Assembler directives

- memory location
- macros
- memory initialization
- conditional compilation
- variables and constants
- output

  - all directives start with .
  - case-insensitive
    (can be switched to sensitive, then keywords are lowercase)

Directives

- BYTE
- CSEG, DSEG, ESEG
- CSEGSIZE
- DB, DW, DD, DQ
- DEF, UNDEF, EQU, SET
- *DEVICE*
- EXIT
- ERROR, WARNING, MESSAGE
- IF, IFDEF, IFNDEF, ELSE, ELIF, ENDIF
- INCLUDE
- LIST, NOLIST, LISTMAC
- MACRO, ENDM, ENDMACRO
- ORG
- OVERLAP, NOOVERLAP

Pre-defined macros

```
__ARVASM_VERSION__
__CORE_VERSION__
__DATE__ __TIME__
__CENTURY__
__YEAR__ __MONTH__ __DATE__
__HOUR__ __MINUTE__ __SECOND__
__FILE__ __LINE__
__PART_NAME__ __partname__
__CORE_coreversion__
```

Expressions

- constant expressions
  - internally 64bit
  - operands
    - labels, variables, constants; PC, int, float
  - operators
  - functions

Functions

- LOW, HIGH
- BYTE2, BYTE3, BYTE4
- LWRD, HWRD
- PAGE
- EXP2, LOG2
- INT, FRAC
- Q7, Q15
- ABS
- DEFINED
- STRLEN

Minimalistic design

```
.include ”m128def.inc”          ; optional – only if needed

.CSEG
.ORG Ø
        CLI
MAIN:
        RJMP MAIN
```

Typical design

```
.include "m128def.inc"
.def TEMP = R19

.CSEG
.ORG Ø
          RJMP RESET                    ; Reset Handler
          NOP                           ; (or only JMP RESET if too far)
          JMP EXT_INTØ
          ...                           ; + all other irq handlers
          JMP SPM_RDY                   ; (this is the last one)

.ORG Øx46
RESET:
          LDI TEMP,LOW(RAMEND)          ; Initialize Stack Pointer
          OUT SPL,TEMP
          LDI TEMP,HIGH(RAMEND)
          OUT SPH,TEMP
          ...                           ; Whatever else needs to be initialized

MAIN:
          ...                           ; Do whatever you want
          RJMP MAIN                     ; repeat

~ or ~
HANG:
          RJMP HANG                     ; Hang up when finished
```

Common design („C-like“)

```
.include "m128def.inc"
.def TEMP = R19

.CSEG
.ORG Ø
          RJMP RESET                    ; Reset Handler
          NOP                           ; (or only JMP RESET if too far)
          JMP __bad_irq
          ...                           ; + all other irq handlers
          JMP __bad_irq                 ; (this is the last one)
.ORG Øx46

RESET:
          EOR r1,r1
          OUT SREG,r1
          LDI TEMP,LOW(RAMEND)          ; Initialize Stack Pointer
          OUT SPL,TEMP
          LDI TEMP,HIGH(RAMEND)
          OUT SPH,TEMP
          ...                           ; Whatever else needs to be initialized
          RCALL MAIN
          RJMP exit

__bad_irq:
          JMP Ø

MAIN:
          ...                           ; Do whatever you want
          RET

exit:
          RJMP exit                     ; Hang
```

Known Issues

- **Issue #4146: Line continuation doesn't work in macro calls**
- **Missing newline at end of file**
  - AVRASM2 has some issues if the last line in a source file is missing a newline: Error messages may refer to wrong filename/line number if the error is in the last line of a included files, and in some cases syntax errors may result. Beware that the Atmel Studio editor will not append a missing newline at the end of a source file automatically.
- **Increment/decrement operators**
  - Increment/decrement operators (++/--) are recognized by the assembler and may cause surprises, e.g. `symbol--1` will cause a syntax error, write `symbol - -1` if that is what is intended. This behaviour is consistent with C compilers. The ++/-- operators are not useful in the current assembler, but are reserved for future use.
- **Forward references in conditionals**
  - Using a forward reference in an assembler conditional may cause surprises, and in some cases is not allowed.
- **Error messages**
  - Sometimes error messages may be hard to understand.
  - Typically, a simple typo in some instances may produce error messages like this:
    `myfile.asm(30): error: syntax error, unexpected FOO`
    where FOO represents some incomprehensible gibberish. The referenced filename/line number is correct, however.
- ***defined* incorrectly treated as an assembler keyword**
  - The keyword DEFINED(symbol) is recognized in all contexts, it should only be recognized in conditionals. This prevents DEFINED(symbol) to be used as a user symbol like a label, etc. On the other hand, it allows for constructs like '.dw foo = defined(bar)' which it shouldn't. Note that the preprocessor and assembler have separate implementations of DEFINED(symbol).
- **Preprocessor issues**
  - The preprocessor will not detect invalid preprocessor directives inside a false conditional. This may lead to surprises with typos. (nested #if)

```
#define TEST \
.IF val
\ .DW 0
\ .ELSE
\ .DW 1
\ .ENDIF
```

Issue #3361: The preprocessor incorrectly allows additional text after directives, which may cause surprises, e.g., #endif #endif will be interpreted as a single #endif directive, without any error or warning message.

Issue #4741: Assembler conditionals in preprocessor macros don't work. Use of the macro defined below will result in different syntax error messages, depending on the value of the conditional val (true or false)
The reason for this is that assembler conditionals must appear on a separate line, and a preprocessor macro like the above is concatenated into a single line.