AVR アセンブラ

序文

ようこそMicrochp AVRｱｾﾝﾌﾞﾗへ。

ｱｾﾝﾌﾞﾗは固定ｺｰﾄﾞ位置を生成し、その結果としてﾘﾝｸの必要がありません。

AVRｱｾﾝﾌﾞﾗは以前にAVRｱｾﾝﾌﾞﾗ2(AVRASM2)として知られているｱｾﾝﾌﾞﾗです。AVR Studio® 4で配給されていた以前のAVRASMは今や廃止されてしまい、現在の製品で配給されていません。

AVR系統のﾏｲｸﾛ ｺﾝﾄﾛｰﾗの命令一式の資料については8ﾋﾞｯﾄAVR命令一式手引書を参照してください。

本書は一般の方々の便宜のため有志により作成されたもので、Microchip社とは無関係であることを御承知ください。しおりの[はじめに]での内容にご注意ください。

目次

序文 ・・・

1. AVRアセンブラ既知の問題

問題#4146: マクロ呼び出しで行結合が動かない。

以下のプログラムがこの問題を例証します。

```
.MACRO m
            LDI @0, @1
.ENDM m r16,¥ 0
```

これは前処理部マクロ(#define)に於いて問題ではありません。

ファイルの最後での改行なし

AVRASM2はソース ファイルの最終行が改行なしの場合にいくつかの問題を持ちます。異常がインクルード ファイルの最終行だった場合に誤りメッセージは不正なファイル名/行番号を示すかもしれず、いくつかの場合で構文誤りに終わるかもしれません。Atmel Studioのエディタがソース ファイルの最後に失っている改行を自動的に追加しないことに注意してください。

増加/減少演算子

増加/減少演算子(++/--)はアセンブラによって認識され、驚きを引き起こすかもしれません。例えば、symbol--1は構文誤りを引き起こし、意図が-1の減算の場合はsymbol - -1と書いてください。

この動きはCコンパイラと一致します。++/--演算子は現在のアセンブラでは有用でありませんが、将来の使用のために予約されています。

条件文での前方参照

アセンブラ条件文での前方(進行方向)参照を使うと、驚きを引き起こすかもしれず、いくつかの場合では許されません。例:

```
.ORG LARGEBOOTSTART
; 以下は代表的にELPMで使う、フラッシュ データ対象を指示するようにRAMPZ:Zを構成設定します。

            LDI     ZL, low (cmdtable * 2)
            LDI     ZH, high (cmdtable * 2)
.IF ((cmdtable * 2) > 65535)

            LDI     R16, 1

            STS     RAMPZ, R16
.ENDIF

; より多くのコードがここに続きます。
cmdtable:  .DB      "foo", 0x0
```

これの理由は条件文の結果が前方参照ラベルの値に影響を及ぼし、順番が条件文の結果に影響を及ぼすなどのためです。

以下は許されます。

```
.IFDEF FOO
            NOP                         ; ここでいくつかのコード
.ENDIF
            RJMP    label               ; ここでもっとコード
            .EQU    FOO = 100
label:      nop
```

この例でFOOはそれが条件文で使われる時点で定義されていません。この状況での.IFDEFの使用は許され、条件文は偽です。けれども、上で示される様式は作成者の意図が不明なため推奨されません。この形式はこのような一般的構造を許すことを意図されます。

```
; FOOが未だ定義されていなければそれを定義
.IFNDEF FOO
            .EQU    FOO = 0x100
.ENDIF
```

AVRASM 2.0.30まではこれらの状況が常に正しく検出されず、不可解な誤りメッセージを引き起こします。2.0.31以降は明確なメッセージが与えられます。

前処理部条件文でこの状況は常に上手く定義され、前処理部シンボルは定義が見られるまで常に未定義で、この種の誤りは決して起きません。

誤りﾒｯｾｰｼﾞ

時々誤りﾒｯｾｰｼﾞの理解が難しいかもしれません。一般的にいくつかの事例で単純な入力誤りが次のこのような誤りﾒｯｾｰｼﾞを生じるかもしれません。

```
myfile.asm(30): error: syntax error, unexpected FOO
```

ここのFOOは理解不能な出鱈目を示します。けれども、参照されるﾌｧｲﾙ名/行番号は正しいものです。

ｱｾﾝﾌﾞﾗ ｷｰﾜｰﾄﾞとして扱われる不正なdefined

definedｷｰﾜｰﾄﾞは全ての文脈で認識されます。これは条件文でだけ認識されるべきです。これはdefinedがﾗﾍﾞﾙなどのような使用者ｼﾝﾎﾞﾙとして使われることを妨げます。これに反して、許されるべきでない'.DW foo = defined(bar)'のような構成を許します。前処理部とｱｾﾝﾌﾞﾗが独立したdefined実装を持つことに注意してください。現在(2.1.5での)definedの正確な動きは次のとおりです。

- 前処理部の'defined'ｷｰﾜｰﾄﾞは#defineで定義されたｼﾝﾎﾞﾙにだけ関係し、現在、前処理部の条件文(#if/#elif)でだけこれを行います。
- 残りの全てのｺｰﾄﾞで、ｱｾﾝﾌﾞﾗのdefinedの見解が使われ、正しい動きはｱｾﾝﾌﾞﾗ条件文(.IF/.ELIF)でだけそれを認識します。

前処理部の問題

- 前処理部は偽条件文内側の無効な前処理部疑似命令を検出しません。これは次のこのような入力誤りで驚きをもたらすかもしれません。

```
#if __ATmega8__
// ～
#elseif __ATmega16__ // 誤り、正しい疑似命令は#elif
// これは__ATmega8__が偽の場合に未検出になります。
// ～
#else // __ATmega8__が偽の時に例え__ATmega16__が真でもｱｾﾝﾌﾞﾙされます。
#endif
```

これがﾊﾞｸﾞならば議論の余地があり、この動きはC前処理部と一致します。

- 問題#3361: 前処理部は疑似命令後の付加的な文字を不正に許し、驚きを引き起こし得ます。例えば、#endif #endifはどの誤りや警告のﾒｯｾｰｼﾞもなしに単一の#endifとして解釈されます。
- 問題#4741: 前処理部ﾏｸﾛでのｱｾﾝﾌﾞﾗ条件文は動きません。下で定義されるﾏｸﾛを使うと、条件文のvalの値(真または偽)に依存して異なる構文誤りﾒｯｾｰｼﾞに帰着します。

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

これの理由はｱｾﾝﾌﾞﾗの条件文が独立した行で現れなければならず、上のような前処理部ﾏｸﾛは単一行に連結されるからです。

2. AVRアセンブラ コマンド行任意選択

AVRASM2はコマンド行から独立型プログラムとして使われるかもしれません。AVRASM2コマンド行呼び出し構文は下で示されます。

```
使い方: avrasm2.exe [任意選択] ファイル.asm

      任意選択:
-f [O|M|I|G|-] 出力ファイル形式:
-fO Atmel Studioでのシミュレーション用デバッグ情報 (既定)
-fO1 | -fO2 - 書式1版または2版を強制 (既定:自動)
-fM モトローラ HEX -fI インテル HEX -fG 一般HEX 形式
-f- 出力ファイルなし -o ofile 'ofile'に出力
-d dfile 'dfile'にAtmel Studioでのシミュレーション用デバッグ情報を生成。-f [M|I|G]任意選択でだけ使うことができます。
-l lfile 'lfile'に一覧(リスト)を生成
-m mfile 'mfile'に対応付け(マップ)を生成
-e efile 'efile'にEEPROM内容を配置
-w 相対分岐が4K語までの大きさのプログラムROMに対して丸めを許容 [無視]
-C ver AVRコア版を指定
-c 大文字と小文字を区別
-1/-2 AVRアセンブラ1版互換性をON/OFF [非推奨]
-p1|0 AVRASM1の暗黙の.deviceインクルードを設定/解除 (-1によっても設定) [非推奨]
-I dir 前処理部: インクルード検索パスに'dir'を追加
-i file 前処理部: fileを明示的にインクルード
-D name[=value] 前処理部: シンボルを定義。=valueが省略された場合、それは1に設定されます。
-U name 前処理部: シンボル定義解消
-S file Atmel Studio用インクルード/ラベル情報ファイルを生成
-v 冗長性 [0-9][s]:
-vs 目的対象資源使用統計を含める。
-v1 低位アセンブリ コードを標準出力(stdout)に出力
-v0 静粛、異常メッセージだけを出力
-v1 異常と警告のメッセージを出力
-v2 異常、警告、情報のメッセージを出力 (既定)
-v3-v9 指定外、アセンブラ内部打ち出し量増加
-V  対応付け(マップ)と一覧(リスト)のファイルをVerilog用に形式化
-O i|w|e 重複報告: i(無視)|w(警告)|e(誤り) [誤り]
-W-b|+bo|+bi バイト被演算子出力の範囲警告 -b(禁止)|+bo(溢れ)|+bi(整数)
-W+ie|+iw 未支援命令 +ie(誤り)|+iw(警告)
-W+fw 前方参照によって引き起こされるラベルずれを許容 (危険)
-FD|Tfmt strftime(3)形式の文字列を使って__DATE__ | __TIME__形式
```

-f 出力ファイル形式

支援される形式は一般/インテル/モトローラのHEXとAVRオブジェクト ファイルです。AVRオブジェクト ファイル形式には次の2つの補助変種があります。

- 16ビット行番号領域を持ち、最大65534行を持つソース ファイルを支援する標準(V1)形式
- 24ビット行番号領域を持ち、最大16M行を持つソース ファイルを支援する拡張(V2)形式

既定により、出力形式が未指定または-fOで指定されると、アセンブラは適切な形式を自動的に選び、ファイルが65533行未満を持つ場合にV1、より多くを持つ場合にV2です。-fO1と-fO2の任意選択は行数と無関係にV1またはV2の出力ファイル形式を強制するのに使うことができます。

V1ファイル形式が65534行以上を持つソース ファイルで使われる場合、アセンブラは警告を発行し、65534以上の行はAtmel Studioでデバッグすることができません。

通常の全てのアセンブラ プロジェクトについて、既定任意選択が安全であるべきです。拡張形式は主に機械生成したアセンブリ ファイルに対して意図されます。

-w

相対分岐丸め。AVRASM2がプログラム メモリの大きさに基づいて相対分岐を丸める時を自動的に決めるので、この任意選択は廃止されています。この任意選択は認識されますが無視されます。

-C コア版

AVRコア版指定。コア版は通常、デバイス定義ファイル(型番def.inc)で指定され、この任意選択はアセンブラの試験を意図され、一般的に最終使用者に対して有用ではありません。

-c

ｱｾﾝﾌﾞﾗを全体的に大文字と小文字を区別するようにさせます。前処理部の疑似命令とﾏｸﾛは常に大文字と小文字を区別します。

警告: この任意選択を設定することは多くの既存ﾌﾟﾛｼﾞｪｸﾄを壊します。

-1 -2 [非推奨]

AVRASM1互換動作を許可/禁止します。この動作形態は既定によって禁止(-2)されます。互換動作はそうでなければ誤りと見做される或る構造を許し、既存ﾌﾟﾛｼﾞｪｸﾄを壊す危険を減らします。組み込みｲﾝｸﾙｰﾄﾞ ﾊﾟｽにも影響を及ぼし、ｱｾﾝﾌﾞﾗに新しいｱｾﾝﾌﾞﾗ2でのC:¥Atmel¥AVR Tools¥AvrAssembler2¥Appnotesに代わってｱｾﾝﾌﾞﾗ1でのC:¥Atmel¥AVR Tools¥AvrAssembler¥Appnotesに対してﾃﾞﾊﾞｲｽ定義ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙ(型番def.inc)を探させます。

注: -1任意選択は非推奨で、AVRASM1ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙがもはや配給されないのでAtmel Studio 6で動きません。

-I ﾃﾞｨﾚｸﾄﾘ

ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙ検索ﾊﾟｽにﾃﾞｨﾚｸﾄﾘを追加。これは前処理部の#include疑似命令とｱｾﾝﾌﾞﾗの.INCLUDE疑似命令の両方に影響を及ぼします。

複数の-I命令を与えることができます。ﾃﾞｨﾚｸﾄﾘは指定した順で検索されます。

-i ﾌｧｲﾙ

ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙ。#includeﾌｧｲﾙ疑似命令は最初のｿｰｽ ｺｰﾄﾞの行が処理される前に処理されます。複数の-i命令を使うことができ、順に処理されます。

-D 名前[=値] -U 名前

各々、前処理部ﾏｸﾛの定義と定義解消。関数型前処理部ﾏｸﾛがｺﾏﾝﾄﾞ行から定義できないことに注意してください。値なし-Dが与えられた場合、それは1に設定されます。

-S

ｿｰｽ ﾌｧｲﾙに含まれるｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙ、出力ﾌｧｲﾙ、ﾗﾍﾞﾙについての情報を生成します。

-vs

ﾚｼﾞｽﾀ、命令、ﾒﾓﾘに対する使用統計を標準出力に出力します。既定により、ﾒﾓﾘ統計だけが出力されます。

注: 1つが指定されなら、常に完全な統計が一覧(ﾘｽﾄ)ﾌｧｲﾙに出力されます。

-vl

これはｼﾝﾎﾞﾘｯｸ情報が置き換えられた後、標準出力に送られた生の命令を出力します。主にｱｾﾝﾌﾞﾗのﾃﾞﾊﾞｯｸﾞ目的用です。

-v0

誤りﾒｯｾｰｼﾞだけを出力、警告と情報のﾒｯｾｰｼﾞは隠されます。

-v1

誤りと警告のﾒｯｾｰｼﾞだけを出力、情報ﾒｯｾｰｼﾞは隠されます。

-v2

誤り、警告、情報のﾒｯｾｰｼﾞを出力。これは既定の動きです。

-v3～-v9

ｱｾﾝﾌﾞﾗ内部状態打ち出しの量を増して出力。主にｱｾﾝﾌﾞﾗのﾃﾞﾊﾞｯｸﾞに使われます。

-V

一覧(ﾘｽﾄ)と対応付け(ﾏｯﾌﾟ)のﾌｧｲﾙをVerilog用に形式化。Verilog任意選択を設定します。

-O i|w|e

ｺｰﾄﾞの違う部分が.ORG疑似命令を使ったﾒﾓﾘ位置と重なって割り付けられた場合、一般的に誤りﾒｯｾｰｼﾞが発行されます。

この任意選択はこの状況を誤り(-Oe、既定)、警告(-Ow)、または完全に無視(-Oi)にさせる設定を許します。通常のﾌﾟﾛｸﾞﾗﾑに対しては推奨されません。

これは#pragma overlap疑似命令によっても設定されるかもしれません。

-W-b | -W+bo | -W+bi

-b、+bo、+biは各々、警告なし、溢れ時に警告、整数値範囲外の時に警告、に対応します。これは#pragma warning range byteによっても設定されるかもしれません。

-W+ie | -W+iw

+ieと+iwは各々、未支援命令の使用が誤りまたは警告かを選びます。既定は誤りを与えることです。各々、#pragma error instruction /#pragma warning instructionに対応します。

-FDformat -FTformat

各々、_DATE_と_TIME_の予め定義されたﾏｸﾛの形式を指定します。形式文字列はstrftime(3) Cﾗｲﾌﾞﾗﾘ関数へ直接的に渡されます。_DATE_と_TIME_の前処理部ﾏｸﾛは常に文字列通票で、即ち、これらの値は二重引用符内に現れます。

既定の形式は各々、"%b %d %Y"と"%H:%M:%S"です。

例: _DATE_に対してISO形式を指定するには -FD"%Y-%m-%d" を指定してください。

これらの形式はｺﾏﾝﾄﾞ行でだけ指定されるかもしれず、対応する#pragma疑似命令はありません。

注: Windows®の命令解釈部(cmd.exeまたはcommand.com)は例えそれが引用されていても'%'文字で始まって終わる文字の流れを展開するための環境変数として解釈するかもしれません。これは命令解釈部によって変更されることを日付/時間の形式文字列に引き起こし、意図するように動かないかもしれません。多くの場合で動く対策は形式指示を指定するのに二重に'%'文字を使う、例えば、上の例に対しては -FD"%%Y-%%m-%%d" です。命令解釈部の正確な動きは一貫性がなく、いくつかの状況に依存して変わり、その1つに関して、一括処理(ﾊﾞｯﾁ)と相互やり取りの動作形態で違います。形式指示の効果が試験されるべきです。これを試験するためにｿｰｽ ﾌｧｲﾙ内に次の行を置くことが推奨されます。

```
#message "__DATE__ =" __DATE__ "__TIME__ =" __TIME__
```

これはﾌﾟﾛｸﾞﾗﾑがｱｾﾝﾌﾞﾙされる時に日付と時間のﾏｸﾛの値を出力し、確認を容易にします(#error、#warning、#messageの疑似命令の文章をご覧ください)。形式指定用代替構文はこの問題を避けるために将来のAVRASM2版で考慮されるかもしれません。

関連するいくつかのstrftime()形式指定子(完全な詳細についてはstrftime(3)手引書をご覧ください。)

- %Y - 年、4桁
- %y - 年、2桁
- %m - 月番号 (01～12)
- %b - 略した月名
- %B - 完全な月名
- %d - 月内の日番号 (01～31)
- %a - 略した曜日名
- %A - 完全な曜日名
- %H - 時、24時間制 (00～23)
- %I - 時、12時間制 (01～12)
- %p - 12時間制用"AM"または"PM"
- %M - 分 (00～59)
- %S - 秒 (00～59)

3. アセンブラ ソース

アセンブラは命令ニーモニック、ラベル、疑似命令を含むソース ファイルで作業します。命令ニーモニックと疑似命令は度々被演算子(オペランド)を取ります。コード行は(半角英数換算で)120文字に制限されるべきです。

全ての入力行はラベルによって先行することができ、これはフルコロン(:)によって終了されるアルファベット文字列です。ラベルは飛んだり分岐する命令に対する目標としてと、プログラム メモリとRAM内の変数名として使われます。

入力行では以下の4つの書式の1つを取り得ます。

[ラベル:] 命令 [被演算子] [注釈]

[ラベル:] 疑似命令 [被演算子] [注釈]

注釈

空の行

注釈は以下の書式を持ちます。

; [文]

大括弧([ ])内に置かれた項目は任意選択です。注釈区切り記号(;)と行の最後(EOL:End of Line)の間の文はアセンブラによって無視されます。ラベル、命令、疑似命令は後でもっと詳細に記述されます。AVRアセンブラ構文もご覧ください。

例:

```
label:  .EQU    var1=100        ; var1を100に設定 (疑似命令)
        .EQU    var2=200        ; var2を200に設定

test:   RJMP    test            ; 無限繰り返し (命令)
                                ; 純粋な注釈行

                                ; 別の注釈行
```

4. AVRアセンブラ構文

4.1. キーワード

予め定義された識別子(キーワード)が予約され、再定義することはできません。キーワードは全ての命令ニーモニックと関数を含みます。

アセンブラのキーワードは-c任意選択が使われない限り、大文字/小文字と無関係に認識され、-c任意選択の場合、キーワードは小文字(即ち、"add"は予約され、"ADD"は違います。)

4.2. 前処理部疑似命令

AVRASM2は全ての行で最初の非空白文字として'#'を持つ文字列を前処理部疑似命令と見做します。

4.3. 注釈

';'で始まる伝統的なアセンブラ注釈に加えて、AVRASM2はC形式の注釈を認識します。以下の注釈形式が認識されます。

```
; 行の残りが注釈です(伝統的アセンブラ注釈)。
// ';'同様、行の残りが注釈です。
/* 塊注釈; 囲われた文が注釈で、複数行に渡るかもしれません。
この形式の注釈は入れ子にすることができません。 */
```

4.4. 行継続

Cでのように、ソース行は行の最終文字として円記号(¥)を持つ方法によって継続することができます。これは長い前処理部マクロを定義する時と長い.DB疑似命令に対して特に有用です。

```
        .DB     0, 1, "これは長い文字列", '¥n', 0, 2, ¥
                "ここは別の文字列", '¥n', 0, 3, 0
```

4.5. 整数定数

AVRASM2は可読性を増すため、アンダースコア(_)に分離子として使われることを許します。アンダースコアは最初の文字または基数指定子の内側としてを除いて数値内の何処かに配置されるかもしれません。

例: 0b1100_1010 と 0b_11_00_10_10_ は共に正当、一方で _0b11001010 と 0_b11001010 は不正です。

4.6. 文字列と文字の定数

二重引用符(")で囲われた文字列は.DB疑似命令と.MESSAGE/.WARNING/.ERROR疑似命令と併せてだけ使うことができます。文字列は文字どおりに取られ、エスケープ文字列は認識されず、NULL($00)終端ではありません。

引用された文字列はANSI Cの慣例に従って連結されるかもしれず、即ち、"This is a " "longstring"は"This is a longstring"と等価です。これは複数のソース行に渡って長い文字列を形成するために行継続と組み合わされるかもしれません。

文字定数は単一引用符(')で囲われ、整数式が許される何処かに使われるかもしれません。右表のC形式エスケープ文字列はCでと同じ意味で認識されます。

¥ooo (ooo=8進数値)と¥xhh (hh=16進数値)も認識されます。

エスケープ文字列	意味
¥n	改行/ニューライン (ASCII LF $0A)
¥r	行頭復帰/キャリッジ リターン (ASCII CR $0D)
¥a	ベル警報 (ASCII BEL $07)
¥b	後退/バックスペース (ASCII BS $08)
¥f	改頁/フォーム フィード (ASCII FF $0C)
¥t	水平タブ (ASCII HT $09)
¥v	垂直タブ (ASCII VT $0B)
¥¥	逆斜線/バックスラッシュ
¥0	ヌル文字 (ASCII NUL $00)

例:

```
        .DB     "Hello¥n" // これは次行と等価です。
        .DB     'H', 'e', 'l', 'l', 'o', '¥¥', 'n'
        .DB     '¥0', '¥177', '¥xff'
```

C文字列の"Hello, world¥n"等価物を作成するには次のとおりに行ってください。

```
        .DB     "Hello, world", '¥n', 0
```

4.7. 1行複数命令

AVRASM2は行毎に複数の命令と疑似命令を許しますが、これを使うことは推奨されません。これは複数行前処理部マクロの展開を支援することが必要とされます。

4.8. 被演算子

AVRASM2は整数被演算子(オペランド)に対する支援と浮動小数点式に対する限定された支援をもちます。全ての被演算子はこの使用者の手引きで後の被演算子部分で記述されます。

5. アセンブラ疑似命令

5.1. BYTE

変数用にバイトを予約

.BYTE疑似命令はSRAMまたはEEPROM内のメモリ資源を予約します。予約した位置を参照することができるようにするため、.BYTE疑似命令はラベルによって先行されるべきです。この疑似命令は1つのパラメータを取り、それは予約するバイト数です。この疑似命令はコード区部内で使うことができません(.ESEG、.CSEG、.DSEGの疑似命令をご覧ください)。パラメータが与えられなければならないことに注意してください。割り当てられたバイトは初期化されません。

構文

```
ラベル:         .BYTE    式
```

例

```
            .DSEG
var1:       .BYTE 1                 ; var1に対して1バイト予約
table:      .BYTE tab_size          ; tab_sizeバイト予約

            .CSEG
            LDI     R30,low(var1)   ; Zレジスタ下位設定
            LDI     R31,high(var1)  ; Zレジスタ上位設定
            LD      R1,Z            ; R1にVAR1を設定
```

5.2. CSEG

コード区部(セグメント)

.CSEG疑似命令はコード区部の開始を定義します。アセンブラ ファイルはいくつかのコード区部から成り、それはアセンブル時に1つのコード区部に連結されます。.BYTE疑似命令はコード区部内で使うことができません。既定の区部型はコードです。コード区部は語計数器であるそれら自身の位置計数器を持ちます。.ORG疑似命令はプログラム メモリ内の特定位置にコードと定数を配置するのに使うことができます。.CSEG疑似命令はどのパラメータも取りません。

構文

```
            .CSEG
```

例

```
            .DSEG                   ; データ区部開始
vartab:     .BYTE 4                 ; SRAMに4バイト予約

            .CSEG                   ; コード区部開始
const:      .DW 2                   ; プログラム メモリに$0002を書き込み
            MOV     R1,R0           ; 何かを行います。
```

5.3. CSEGSIZE

プログラム メモリの大きさ

AT94Kデバイスは AVRプログラム メモリとデータ メモリ間に使用者構成設定可能なメモリ分割を持ちます。プログラムとデータSRAMは、10K×16の専用プログラムSRAM、4K×8の専用データSRAM、6K×16または12K×8に構成設定可能なSRAMの3つ塊に分けられ、これはプログラムとデータのメモリ空間の間で2K×16または4K×8の区画に交換されるかもしれません。この疑似命令はプログラム メモリの塊の大きさを指定するのに使われます。

構文

```
            .CSEGSIZE = 10 | 12 | 14 | 16
```

例

```
            .CSEGSIZE = 12          ; 12K×16としてプログラム メモリの大きさを指定
```

5.4. DB

プログラム メモリとEEPROMでバイト定数を定義

.DB疑似命令はプログラム メモリまたはEEPROM内のメモリ資源を予約します。予約した位置を参照することができるようにするため、.DB疑似命令はラベルによって先行されるべきです。.DB疑似命令は式の列記を取り、最低1つの式を取らなければなりません。.DB疑似命令はコード区部またはEEPROM区部に置かれなければなりません。

式列記はコンマ(,)によって分離された式の連続です。各式は-128と255の間の数値と評価しなければなりません。式が負数と評価する場合、プログラム メモリまたはEEPROMの位置にその数値の8ビットの2の補数が置かれます。

.DB疑似命令がｺｰﾄﾞ区部で与えられ、式列記が複数の式を含む場合、式は各ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ語内に2ﾊﾞｲﾄが置かれるように詰め込まれます。式列記が奇数の式を含む場合、最後の式は例えｱｾﾝﾌﾞﾘ ｺｰﾄﾞで次の行が.DB疑似命令を含んでもそれ自身のﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ語に置かれます。ﾌﾟﾛｸﾞﾗﾑ語の未使用の半分は$00に設定されます。.DB文に余分な$00ﾊﾞｲﾄが追加されたことを使用者に通知するため、警告が与えられます。

構文

```
ﾗﾍﾞﾙ:       .DB       式列記
```

例

```
            .CSEG
consts:     .DB       0,  255,  0b01010101,  -128,  0xAA

            .ESEG
const2:     .DB       1,2,3
```

5.5. DD

ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘとEEPROMで倍語定数を定義

この疑似命令は32ﾋﾞｯﾄ(倍語)に対して使われることを除き、.DW疑似命令と非常に似ています。ﾒﾓﾘ内のﾃﾞｰﾀ配置は厳密にﾘﾄﾙ ｴﾝﾃﾞｨｱﾝ(下位側ｱﾄﾞﾚｽが数値の下位ﾊﾞｲﾄ)です。

構文

```
ﾗﾍﾞﾙ:       .DD       式列記
```

例

```
            .CSEG
varlist:    .DD       0,  0xFADEBABE,  -2147483648,  1 << 30
```

5.6. DEF

ﾚｼﾞｽﾀにｼﾝﾎﾞﾙ名を設定

.DEF疑似命令はﾚｼﾞｽﾀにｼﾝﾎﾞﾙを通して参照されることを許します。定義されたｼﾝﾎﾞﾙはそれが割り当てられたﾚｼﾞｽﾀを参照するためにﾌﾟﾛｸﾞﾗﾑの残りで使うことができます。ﾚｼﾞｽﾀはそれに付随されるいくつかのｼﾝﾎﾞﾙ名を持つことができます。この定義は.UNDEF疑似命令で解消することができます。ｼﾝﾎﾞﾙはﾌﾟﾛｸﾞﾗﾑで後に再定義することができます。

構文

```
            .DEF      ｼﾝﾎﾞﾙ=ﾚｼﾞｽﾀ
```

例

```
            .DEF      temp=R16
            .DEF      ior=R0

            .CSEG
            LDI       temp,0xF0          ; tempﾚｼﾞｽﾀに$F0を設定
            IN        ior,0x3F           ; iorﾚｼﾞｽﾀにSREGを読み込み
            EOR       temp,ior           ; tempとiorを排他的論理和
```

5.7. DQ

ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘとEEPROMで4倍語定数を定義

この疑似命令は64ﾋﾞｯﾄ(4倍語)に対して使われることを除き、.DW疑似命令と非常に似ています。ﾒﾓﾘ内のﾃﾞｰﾀ配置は厳密にﾘﾄﾙ ｴﾝﾃﾞｨｱﾝ(下位側ｱﾄﾞﾚｽが数値の下位ﾊﾞｲﾄ)です。

構文

```
ﾗﾍﾞﾙ:       .DQ       式列記
```

例

```
            .ESEG
eevarlst:   .DQ 0,0xFADEBABEDEADBEEF,  1 << 62
```

5.8. DSEG

ﾃﾞｰﾀ区部(ｾｸﾞﾒﾝﾄ)

.DSEG疑似命令はﾃﾞｰﾀ区部の開始を定義します。ｱｾﾝﾌﾞﾗ ｿｰｽ ﾌｧｲﾙはいくつかのﾃﾞｰﾀ区部から成り、それはｱｾﾝﾌﾞﾙ時に1つのﾃﾞｰﾀ区部に連結されます。ﾃﾞｰﾀ区部は通常、.BYTE疑似命令(とﾗﾍﾞﾙ)だけから成ります。ﾃﾞｰﾀ区部はﾊﾞｲﾄ計数器であるそれら自身の位置計数器を持ちます。.ORG疑似命令はSRAM内の特定位置に変数を配置するのに使うことができます。この疑似命令はどのﾊﾟﾗﾒｰﾀも取りません。

構文

```
            .DSEG
```

例

```
            .DSEG
var1:       .BYTE 1                 ; var1に対して1ﾊﾞｲﾄ予約
table:      .BYTE tab_size          ; tab_sizeﾊﾞｲﾄ予約

            .CSEG
            LDI     R30,low(var1)   ; Zﾚｼﾞｽﾀ下位設定
            LDI     R31,high(var1)  ; Zﾚｼﾞｽﾀ上位設定
            LD      R1,Z            ; R1にVAR1を設定
```

5.9. DW

ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘとEEPROMで語定数を定義

.DW疑似命令はﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘまたはEEPROM内のﾒﾓﾘ資源を予約します。予約した位置を参照することができるようにするため、.DW疑似命令はﾗﾍﾞﾙによって先行されるべきです。.DW疑似命令は式の列記を取り、最低1つの式を取らなければなりません。.DW疑似命令はｺｰﾄﾞ区部またはEEPROM区部に置かれなければなりません。ﾒﾓﾘ内のﾃﾞｰﾀ配置は厳密にﾘﾄﾙ ｴﾝﾃﾞｨｱﾝ(下位側ｱﾄﾞﾚｽが数値の下位ﾊﾞｲﾄ)です。

式列記はｺﾝﾏ(,)によって分離された式の連続です。各式は-32768と65535の間の数値と評価しなければなりません。式が負数と評価する場合、ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘまたはEEPROMの位置にその数値の16ﾋﾞｯﾄの2の補数が置かれます。

構文

```
ﾗﾍﾞﾙ:       .DW     式列記
```

例

```
            .CSEG
varlist:    .DW     0, 0xFFFF, 0b1001110001010101, -32768, 65535

            .ESEG
eevarlst:   .DW     0,0xFFFF,10
```

5.10. ELIFとELSE

条件付きｱｾﾝﾌﾞﾘ

.ELFは式が真で、最初の.IF節と後続する.ELIF節が偽の場合に次の.ELIFの対応する.ENDIFまでのｺｰﾄﾞを含めます。

.ELIFは最初の.IF節と(もしあれば)全ての.ELIF節が偽の場合に対応する.ENDIFまでのｺｰﾄﾞを含めます。

構文

```
.ELIF <式>
.ELSE
.IFDEF <ｼﾝﾎﾞﾙ> |.IFNDEF <ｼﾝﾎﾞﾙ>
～
.ELSE | .ELIF <式>
～
```

例

```
.IFDEF DEBUG
.MESSAGE "ﾃﾞﾊﾞｯｸﾞ中..."
.ELSE
.MESSAGE "開放..."
.ENDIF
```

5.11. ENDIF

条件付きｱｾﾝﾌﾞﾘ

条件付きｱｾﾝﾌﾞﾘはｱｾﾝﾌﾞﾙ時に於ける指令の組を含みます。.ENDIF疑似命令は条件文の.IF、.IFDEF、.IFNDEFの疑似命令に対する終了を定義します。

条件文(.IF～.ELIF～.ELSE～.ENDIFの塊)は入れ子にされるかもしれませんが、条件文はﾌｧｲﾙの最後で終了されなければなりません(条件文は複数ﾌｧｲﾙに渡ることができません)。

構文

```
.ENDIF
.IFDEF <ｼﾝﾎﾞﾙ> |.IFNDEF <ｼﾝﾎﾞﾙ>
～
.ELSE | .ELIF <式>
～
.ENDIF
```

例

```
.IFNDEF DEBUG
.MESSAGE "開放..."
.ELSE
.MESSAGE "ﾃﾞﾊﾞｯｸﾞ中..."
.ENDIF
```

5.12. ENDMとENDMACRO

ﾏｸﾛ終了

.ENDMACRO疑似命令はﾏｸﾛ定義の終わりを定義します。この疑似命令はどのﾊﾟﾗﾒｰﾀも取りません。ﾏｸﾛを定義するより多くの情報については.MACRO疑似命令をご覧ください。.ENDMは代替形で、.ENDMACROと完全に等価です。

構文

```
        .ENDMACRO
        .ENDM
```

例

```
        .MACRO   SUBI16          ; ﾏｸﾛ定義開始
        SUBI     R16,low(@0)     ; 下位ﾊﾞｲﾄ減算
        SBCI     R17,high(@0)    ; 上位ﾊﾞｲﾄ減算
        .ENDMACRO
```

5.13. EQU

式に等しいｼﾝﾎﾞﾙを設定

.EQU疑似命令はﾗﾍﾞﾙに値を割り当てます。このﾗﾍﾞﾙはその後に後の式で使うことができます。.EQU疑似命令によって値を割り当てられたﾗﾍﾞﾙは定数で、変更や再定義をすることができません。

構文

```
        .EQU     ﾗﾍﾞﾙ = 式
```

例

```
        .EQU     io_offset = 0x23
        .EQU     porta = io_offset + 2

        .CSEG                          ; ｺｰﾄﾞ区部開始
        CLR      R2                    ; ﾚｼﾞｽﾀ2を解消
        OUT      porta,R2              ; ﾎﾟｰﾄAに書き込み
```

5.14. ERROR

誤りﾒｯｾｰｼﾞ文字列を出力

.ERROR疑似命令は文字列を出力してｱｾﾝﾌﾞﾙを停止します。条件付きｱｾﾝﾌﾞﾘで使われるかもしれません。

構文

```
        .ERROR   "<文字列>"
```

例

```
.IFDEF TOBEDONE
.ERROR "行われるべきものが未だあります。"
.ENDIF
```

5.15. ESEG

EEPROM区部(ｾｸﾞﾒﾝﾄ)

.ESEG疑似命令はEEPROM区部の開始を定義します。ｱｾﾝﾌﾞﾗ ｿｰｽ ﾌｧｲﾙはいくつかのEEPROM区部から成り、それはｱｾﾝﾌﾞﾙ時に単一のEEPROM区部に連結されます。EEPROM区部は通常、.DB、.DW、.DD、.DQの疑似命令(ﾄﾗﾍﾞﾙ)だけから成ります。EEPROM区部はﾊﾞｲﾄ計数器であるそれら自身の位置計数器を持ちます。.ORG疑似命令はEEPROM内の特定位置に変数を配置するのに使うことができます。この疑似命令はどのﾊﾟﾗﾒｰﾀも取りません。

構文

```
            .ESEG
```

例

```
            .DSEG
var1:       .BYTE   1               ; var1に対して1ﾊﾞｲﾄ予約
table:      .BYTE   tab_size        ; tab_sizeﾊﾞｲﾄ予約

            .ESEG
eevar1:     .DW     0xFFFF          ; EEPROMの1語を初期化
```

5.16. EXIT

このﾌｧｲﾙを終了

.EXIT疑似命令はﾌｧｲﾙのｱｾﾝﾌﾞﾙ停止をｱｾﾝﾌﾞﾗに告げます。通常、ｱｾﾝﾌﾞﾗはﾌｧｲﾙの最後(EOF)まで走行します。ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙで.EXIT疑似命令が現れた場合、ｱｾﾝﾌﾞﾗは.INCLUDE疑似命令を含んでいるﾌｧｲﾙに於いて.INCLUDE疑似命令に後続する行から続けます。

構文

```
            .EXIT
```

例

```
            .EXIT                   ; このﾌｧｲﾙを終了
```

5.17. IF、IFDEF、IFNDEF

条件付きｱｾﾝﾌﾞﾘ

条件付きｱｾﾝﾌﾞﾘはｱｾﾝﾌﾞﾙ時に於ける指令の組を含みます。.IFDEF疑似命令は<ｼﾝﾎﾞﾙ>が定義されているなら、対応する.ELSE疑似命令までのｺｰﾄﾞを含めます。ｼﾝﾎﾞﾙは.EQUまたは.SETの疑似命令で定義されなければなりません(.DEF疑似命令では動きません)。.IF疑似命令は<式>が0以外に評価される場合にｺｰﾄﾞを含みます。対応する.ELSEまたは.ENDIFの疑似命令までが有効です。

5段階までの入れ子が可能です。

構文

```
.IFDEF <ｼﾝﾎﾞﾙ>
.IFNDEF <ｼﾝﾎﾞﾙ>
.IF <式>
.IFDEF <ｼﾝﾎﾞﾙ> |.IFNDEF <ｼﾝﾎﾞﾙ>
～
.ELSE | .ELIF <式>
～
.ENDIF
```

例

```
            .MACRO SET_BAT
.IF @0>0x3F
            .MESSAGE "0x3Fを超えるｱﾄﾞﾚｽ"
            LDS     @2, @0
            SBR     @2, (1<<@1)
            STS     @0, @2
.ELSE
            .MESSAGE "0x3F以下のｱﾄﾞﾚｽ"
.ENDIF
            .ENDMACRO
```

5.18. INCLUDE

別のﾌｧｲﾙを組み込み(ｲﾝｸﾙｰﾄﾞし)ます。

.INCLUDE疑似命令は指定したﾌｧｲﾙから読み込みを開始することをｱｾﾝﾌﾞﾗに告げます。ｱｾﾝﾌﾞﾗはその後に指定したﾌｧｲﾙをﾌｧｲﾙの最後、または.EXIT疑似命令に出会うまでｱｾﾝﾌﾞﾙを行います。ｲﾝｸﾙｰﾄﾞされたﾌｧｲﾙが.INCLUDE疑似命令自体を含むかもしれません。(構文での)2つの形式の違いは1つ目のものが最初に現在のﾃﾞｨﾚｸﾄﾘを検索し、2つ目のものがそれをしないことです。

構文

```
.INCLUDE "ﾌｧｲﾙ名"
.INCLUDE <ﾌｧｲﾙ名>
```

例

```
; iodefs.asm:
            .EQU    sreg    = 0x3F  ; ｽﾃｰﾀｽ ﾚｼﾞｽﾀ
            .EQU    sphigh  = 0x3E  ; ｽﾀｯｸ ﾎﾟｲﾝﾀ上位
            .EQU    splow   = 0x3D  ; ｽﾀｯｸ ﾎﾟｲﾝﾀ下位

; incdemo.asm
.INCLUDE iodefs.asm                 ; I/O定義をｲﾝｸﾙｰﾄﾞ
            IN      R0,sreg         ; ｽﾃｰﾀｽ ﾚｼﾞｽﾀ読み込み
```

5.19. LIST

一覧(ﾘｽﾄ)ﾌｧｲﾙ生成をONに切り替え

.LIST疑似命令は一覧(ﾘｽﾄ)ﾌｧｲﾙ生成をONに切り替えることをｱｾﾝﾌﾞﾗに告げます。ｱｾﾝﾌﾞﾗはｱｾﾝﾌﾞﾘ ｿｰｽ ｺｰﾄﾞ、ｱﾄﾞﾚｽ、命令符号の組み合わせである一覧ﾌｧｲﾙを生成します。一覧ﾌｧｲﾙは既定でONです。この疑似命令はｱｾﾝﾌﾞﾘ ｿｰｽ ﾌｧｲﾙの選んだ部分の一覧ﾌｧｲﾙだけを生成するために.NOLIST疑似命令と共に使うこともできます。

構文

```
.LIST
```

例

```
.NOLIST                             ; 一覧ﾌｧｲﾙ生成禁止
.INCLUDE "macro.inc"                ; ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙは一覧ﾌｧｲﾙで見えるようにされません。
.INCLUDE "const.def"                ;
.LIST                               ; 一覧ﾌｧｲﾙ生成を再許可
```

5.20. LISTMAC

ﾏｸﾛ展開をONに切り替え

.LISTMAC疑似命令はﾏｸﾛが呼ばれた時にｱｾﾝﾌﾞﾗによって生成される一覧(ﾘｽﾄ)ﾌｧｲﾙでﾏｸﾛの展開が見えるようにｱｾﾝﾌﾞﾗに告げます。既定はﾊﾟﾗﾒｰﾀを持つﾏｸﾛ呼び出しだけが一覧ﾌｧｲﾙで示されます。

構文

```
.LISTMAC
```

例

```
            .MACRO MACX                 ; 例ﾏｸﾛを定義
            ADD     R0,@0               ; 何か行います。
            EOR     R1,@1               ; 何か行います。
            .ENDMACRO                   ; ﾏｸﾛ定義終了

.LISTMAC                                ; ﾏｸﾛ展開許可

            MACX    R2,R1               ; ﾏｸﾛ呼び出し、展開表示
```

5.21. MACRO

ﾏｸﾛ開始

.MACRO疑似命令はこれがﾏｸﾛ定義の始めであることをｱｾﾝﾌﾞﾗに告げます。.MACRO疑似命令はﾊﾟﾗﾒｰﾀとしてﾏｸﾛ名を取ります。ﾌﾟﾛｸﾞﾗﾑ内に於いて後でﾏｸﾛの名前が書かれると、それが使われた所でﾏｸﾛ定義が展開されます。ﾏｸﾛは10個までのﾊﾟﾗﾒｰﾀを取ることができます。これらのﾊﾟﾗﾒｰﾀはﾏｸﾛ定義内で@0～@9として参照されます。ﾏｸﾛ呼び出しを発行する時にﾊﾟﾗﾒｰﾀはｺﾝﾏ(,)で分離された列記として与えられます。ﾏｸﾛ定義は.ENDMACRO疑似命令によって終了されます。

既定により、ｱｾﾝﾌﾞﾗによって生成された一覧(ﾘｽﾄ)ﾌｧｲﾙで、ﾏｸﾛ呼び出しだけが示されます。一覧ﾌｧｲﾙでﾏｸﾛ展開を含めるためには.LISTMAC疑似命令が使われなければなりません。ﾏｸﾛは一覧ﾌｧｲﾙの命令符号領域に於いて+で記されます。

構文

```
            .MACRO   ﾏｸﾛ名
```

例

```
            .MACRO   SUBI16             ; ﾏｸﾛ定義開始
            SUBI     @1,low(@0)         ; 下位ﾊﾞｲﾄ減算
            SBCI     @2,high(@0)        ; 上位ﾊﾞｲﾄ減算
            .ENDMACRO                   ; ﾏｸﾛ定義終了

            .CSEG                       ; ｺｰﾄﾞ区部開始
            SUBI16   0x1234,R16,R17     ; R17:R6から0x1234を減算
```

5.22. MESSAGE

ﾒｯｾｰｼﾞ文字列を出力

.MESSAGE疑似命令は文字列を出力します。条件付きｱｾﾝﾌﾞﾘで有用です。

構文

```
.MESSAGE    "<文字列>"
```

例

```
.IFDEF DEBUG
.MESSAGE "ﾃﾞﾊﾞｯｸﾞ動作"
.ENDIF
```

5.23. NOLIST

一覧(ﾘｽﾄ)ﾌｧｲﾙ生成をOFFに切り替え

.NOLIST疑似命令は一覧(ﾘｽﾄ)ﾌｧｲﾙ生成をOFFに切り替えることをｱｾﾝﾌﾞﾗに告げます。ｱｾﾝﾌﾞﾗは通常、ｱｾﾝﾌﾞﾘ ｿｰｽ ｺｰﾄﾞ、ｱﾄﾞﾚｽ、命令符号の組み合わせである一覧ﾌｧｲﾙを生成します。一覧ﾌｧｲﾙは既定でONですが、この疑似命令を使うことによって禁止することができます。この疑似命令はｱｾﾝﾌﾞﾘ ｿｰｽ ﾌｧｲﾙの選んだ部分の一覧ﾌｧｲﾙだけを生成するために.LIST疑似命令と共に使うこともできます。

構文

```
.NOLIST
```

例

```
.NOLIST                                 ; 一覧ﾌｧｲﾙ生成禁止
.INCLUDE "macro.inc"                    ; ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙは一覧ﾌｧｲﾙで見えるようにされません。
.INCLUDE "const.def"                    ;
.LIST                                   ; 一覧ﾌｧｲﾙ生成を再許可
```

5.24. ORG

ﾌﾟﾛｸﾞﾗﾑ原点を設定

.ORG疑似命令は位置計数器に絶対値を設定します。設定する値はﾊﾟﾗﾒｰﾀとして与えられます。.ORG疑似命令がﾃﾞｰﾀ区部で与えられる場合に設定されるのはSRAM位置計数器、この疑似命令がｺｰﾄﾞ区部で与えられる場合に設定されるのはﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ計数器、そしてこの疑似命令がEEPROM区部で与えられる場合に設定されるのはEEPROM位置計数器です。

ｺｰﾄﾞとEEPROMの位置計数器の既定値は0で、SRAM位置計数器の既定値はｱｾﾝﾌﾞﾙが開始される時にI/Oｱﾄﾞﾚｽ空間の終了直後のｱﾄﾞﾚｽ(代表的に、拡張I/Oなしのﾃﾞﾊﾞｲｽは$0060、拡張I/O付きﾃﾞﾊﾞｲｽは$0100)です。SRAMとEEPROMの位置計数器はﾊﾞｲﾄで計数し、一方でﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ位置計数器が語で計数することに注意してください。いくつかのﾃﾞﾊﾞｲｽでSRAMやEEPROMがないことに注意してください。

構文

```
            .ORG    式
```

例

```
            .DSEG                   ; ﾃﾞｰﾀ区部開始
            .ORG    0x0120          ; SRAMｱﾄﾞﾚｽを16進の120に設定
variable:   .BYTE   1               ; SRAMｱﾄﾞﾚｽ$0120でﾊﾞｲﾄを予約

            .CSEG                   ; ｺｰﾄﾞ区部開始
            .ORG    0x0010          ; ﾌﾟﾛｸﾞﾗﾑ ｶｳﾝﾀを16進の10に設定
            MOV     R0, R1          ; 何かを行います。
```

5.25. OVERLAPとNOOVERLAP

重複する区部(ｾｸｼｮﾝ)を構成設定

AVRASM2.1で導入。これらの疑似命令は特別な要求を持つﾌﾟﾛｼﾞｪｸﾄ用で、通常は使われるべきではありません。

これらの疑似命令は現在活性の区部(.CSEG、.DSEG、.ESEG)にだけ影響を及ぼします。

.OVERLAP/.NOOVERLAP疑似命令はどの誤りや警告のﾒｯｾｰｼﾞも生成されることなく、ｺｰﾄﾞ/ﾃﾞｰﾀを他の場所で定義されたｺｰﾄﾞ/ﾃﾞｰﾀと重複することを許すことを区部(ｾｸｼｮﾝ)に記します。これは#pragma汎用疑似命令を使って設定するものから全く独立しています。重複許容属性は.ORG疑似命令の向こう側へも引き続き有効ですが、.CSEG/.DSEG/.ESEGの疑似命令の向こう側には引き継がれません(各区部は別々に記されます)。

これの代表的な使い方は、ｱｾﾝﾌﾞﾗの重複検出を完全に禁止することなく、重複するｺｰﾄﾞまたはﾃﾞｰﾀによって後で変更されるかどうか分からないｺｰﾄﾞまたはﾃﾞｰﾀの或る既定の形式を構成設定することです。

構文

```
.OVERLAP
.NOOVERLAP
```

例

```
.overlap
            .ORG    0               ; 区部#1
            RJMP    default
.nooverlap
            .ORG    0               ; 区部#2
            RJMP    RESET           ; ここで誤りは与えられません。
            .ORG    0               ; 区部#3
            RJMP    RESET           ; #2と重複するためここで誤り
```

5.26. SET

式に等しいｼﾝﾎﾞﾙを設定

.SET疑似命令はﾗﾍﾞﾙに値を割り当てます。このﾗﾍﾞﾙはその後に後の式で使うことができます。.EQU疑似命令と違い、.SET疑似命令によって値を割り当てられたﾗﾍﾞﾙはﾌﾟﾛｸﾞﾗﾑに於いて後で変更(再定義)することができます。

構文

```
            .SET    ﾗﾍﾞﾙ = 式
```

例

```
            .SET    FOO = 0x114     ; FOOにSRAMの位置を設定
            LDS     R0, FOO         ; 位置をR0に設定
            .SET    FOO = FOO+1     ; FOO増加(再定義)。.EQUを使う場合これは不正です。
            LDS     R1, FOO         ; 次の位置をR1に設定
```

5.27. UNDEF

ﾚｼﾞｽﾀ ｼﾝﾎﾞﾙ名の定義を解消

.UNDEF疑似命令は以前に.DEF疑似命令で定義されたｼﾝﾎﾞﾙの定義を解消します。これはﾚｼﾞｽﾀ再使用についての警告を避けるため、ﾚｼﾞｽﾀ定義の簡単な有効範囲を達成する方法を提供します。

構文

```
        .UNDEF   ｼﾝﾎﾞﾙ
```

例

```
        .DEF     var1 = R16
        LDI      var1,0x20
        ～                          ; var1でもっと何かを行います。
        .UNDEF   var1

        .DEF     var2 = R16         ; R16は今や警告なしで再使用することができます。
```

5.28. WARNING

警告ﾒｯｾｰｼﾞ文字列を出力

.WARNING疑似命令は警告文字列を出力しますが、.ERROR疑似命令と異なり、ｱｾﾝﾌﾞﾙを停止しません。条件付きｱｾﾝﾌﾞﾙで使われるかもしれません。

構文

```
.WARNING "<文字列>"
```

例

```
.IFDEF EXPERIMENTAL_FEATURE
.WARNING "これは正しく試験されておらず、自己責任でお使いください。"
.ENDIF
```

6. 前処理部

AVRASM2前処理部(ﾌﾟﾘﾌﾟﾛｾｯｻ)は以下のいくつかの例外付きでC前処理部にならって作られています。

- AVRASM2によって使われる全ての整数形式、即ち、$ABCDと0b011001は前処理部によって有効な整数として認識され、#if疑似命令で式に使うことができます。
- '.'と'@'は識別子で許されます。'.'は前処理部ﾏｸﾛで使われる'.DW'のような前処理部疑似命令を許すのに必要とされ、'@'はｱｾﾝﾌﾞﾗ ﾏｸﾛ引数を正しく処理するのに必要とされます。
- ｱｾﾝﾌﾞﾗ形式の注釈区切り文字(;)だけでなくC形式の注釈も認識します。注釈区切り文字として';'を使うことは';'のCの使い方と衝突し、従って前処理部疑似命令と共にｱｾﾝﾌﾞﾗ形式の注釈に使うことは推奨されません。
- #line疑似命令は実装されていません。
- 可変引数ﾏｸﾛ(即ち、可変個の引数を持つﾏｸﾛ)は実装されていません。
- #warningと#messageの疑似命令はANSI C規格で指定されていません。

6.1. #define

構文

```
① #define 名前 [値]
② #define 名前(引数, ～) [値]
```

説明

前処理部ﾏｸﾛを定義。基本的に定数を定義するｵﾌﾞｼﾞｪｸﾄ様ﾏｸﾛ(①)とﾊﾟﾗﾒｰﾀ代入を行う関数様ﾏｸﾛ(②)の2つのﾏｸﾛ形式があります。

値は何れかの文字列かもしれず、これはﾏｸﾛが展開される(使われる)まで評価されません。値が指定されない場合、これは1に設定されます。

形式①のﾏｸﾛは-D引数を使ってｺﾏﾝﾄﾞ行から定義されるかもしれません。

形式②使用時、ﾏｸﾛはそれが定義されたのと同じ引数で呼ばれなければなりません。値での引数のどの存在もﾏｸﾛが呼ばれる時に対応する引数で置き換えられます。左括弧が名前の直後に現れなければならず(間に空白なし)、さもなければそれは形式①のﾏｸﾛの値の一部として解釈されることに注意してください。

例

最初の'('の配置が下の例で非常に重要なことに注意してください。

```
① #define  EIGHT  (1 << 3)
② #define  SQR(X)  ((X)*(X))
```

6.2. #undef

構文

```
#undef 名前
```

説明

以前に#define疑似命令で定義されたﾏｸﾛ名の定義を解消します。名前が以前に定義されていない場合、.undef疑似命令は黙って無視されます。この動きはANSI C規格に従っています。ﾏｸﾛは-U引数を使ってｺﾏﾝﾄﾞ行から定義を解消されることもあるかもしれません。

6.3. #ifdef

構文

```
#ifdef 名前
```

説明

名前が以前に#defined(#define疑似命令で定義)されていれば、対応する#endif、#else、#elifまで後続する全ての行が条件付きｱｾﾝﾌﾞﾙされます。#if defined (名前)に対する略称。

例

```
#ifdef FOO
            // 何かを行います。
#endif
```

6.4. #ifndef

構文

```
#ifndef 名前
```

説明

#ifdefの逆。名前が以前に#defined(#define疑似命令で定義)されていなければ、対応する#endif、#else、#elifまで後続する全ての行が条件付きｱｾﾝﾌﾞﾙされます。#if !defined (名前)に対する略称。

6.5. #ifと#elif

構文

```
#if 条件
#elif 条件
```

説明

条件が真ならば(0と等しくなければ)、後続する全ての行が対応する#endif、#else、#elifまで条件付きｱｾﾝﾌﾞﾙされます。条件は多分、展開された前処理部ﾏｸﾛを含む何れかの整数式です。前処理部は名前が定義されている場合に1を、そうでなければ0を返す特別な演算子の#defined (名前)を認識します。条件で使われるどの未定義ｼﾝﾎﾞﾙも黙って0に評価されます。

条件文は任意の深さに入れ子にされるかもしれません。

#elifは前の#if～#elifの合成列の分岐が真と評価されない場合にだけ評価されることを除き、#ifと同じ規則で条件を評価します。

例

```
#if 0
            // ここのｺｰﾄﾞは決して含まれません。
#endif

#if defined(__ATmega48__) || defined(__ATmega88__)
            // これらのﾃﾞﾊﾞｲｽ用特有ｺｰﾄﾞ
#elif defined (__ATmega169__)
            // ATmega169用特有ｺｰﾄﾞ
#endif      // ﾃﾞﾊﾞｲｽ特有ｺｰﾄﾞ
```

6.6. #else

構文

```
#else
```

説明

前の#if～#elifの合成列での分岐が真と評価されない場合に後続する全ての行が対応する#endifまで条件付きｱｾﾝﾌﾞﾙされます。

例

```
#if defined(__ATmega48__) || defined(__ATmega88__)
            // これらのﾃﾞﾊﾞｲｽ用特有ｺｰﾄﾞ
#elif defined (__ATmega169__)
            // ATmega169用特有ｺｰﾄﾞ
#else
            #error "未支援ﾃﾞﾊﾞｲｽ:" __PART_NAME__
#endif      // ﾃﾞﾊﾞｲｽ特有ｺｰﾄﾞ
```

6.7. #endif

構文

```
#endif
```

説明

#if、#ifdef、#ifndefの疑似命令で始まる条件文の塊を終了

6.8. #error、#warning、#message

構文

```
#error 通票
#warning 通票
#message 通票
```

説明

#errorは標準誤り出力に通票を発行してｱｾﾝﾌﾞﾗ誤り計数器を増し、これによってﾌﾟﾛｸﾞﾗﾑが成功裏にｱｾﾝﾌﾞﾙされることを阻みます。#errorはANSI C規格で指定されています。

#warningは標準誤り出力に通票を発行してｱｾﾝﾌﾞﾗ警告計数器を増します。#warningはANSI C規格で指定されませんが、GNU C前処理部のような前処理部で一般的に実装されています。

#messageは標準出力に通票を発行し、ｱｾﾝﾌﾞﾗの誤りと警告の計数器に影響を及ぼしません。#messageはANSI C規格で指定されていません。

これら全ての疑似命令について、出力は通常の誤りと警告のﾒｯｾｰｼﾞのようにｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙ名と行番号を含みます。

通票は前処理部通票の列です。前処理部ﾏｸﾛは(")で引用された文字列の内側に現れる場合を除いて展開されます。

例

```
#error "未支援ﾃﾞﾊﾞｲｽ:" __PART_NAME__
```

6.9. #include

構文

```
① #include "ﾌｧｲﾙ名"
② #include <ﾌｧｲﾙ名>
```

説明

ﾌｧｲﾙを組み込み(ｲﾝｸﾙｰﾄﾞ)。2つの形式は①が現在の作業ﾃﾞｨﾚｸﾄﾘを最初に検索し、機能的にｱｾﾝﾌﾞﾗの.INCLUDE疑似命令と等価であることで異なり、②はｲﾝｸﾙｰﾄﾞ ﾊﾟｽに於いて"."登録で明示的に指定していない限り、現在の作業ﾃﾞｨﾚｸﾄﾘを検索しません。両形式は明示的に指定されたどれものﾊﾟｽの後に既知の組み込み場所を検索します。これはｱｾﾝﾌﾞﾗで供給された型番def.incｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙ用位置です。

これが異なるﾃﾞｨﾚｸﾄﾘ/ｺﾝﾋﾟｭｰﾀ間でﾌﾟﾛｼﾞｪｸﾄを移動することを難しくするため、#include疑似命令で絶対ﾊﾟｽを使うことは強く反対されます。ｲﾝｸﾙｰﾄﾞ ﾊﾟｽを指定するのに-I引数を使うか、またはAtmel Studioのﾌﾟﾛｼﾞｪｸﾄ(Project)⇒ｱｾﾝﾌﾞﾗ任選択(Assembler Optioons)でそれを構成設定してください。

例

```
#include <m48def.inc>
#include "mydefs.inc"
```

6.10. #pragma 汎用

構文

```
① #pragma warning range byte 任意選択
② #pragma overlap 任意選択
③ #pragma error instruction
④ #pragma warning instruction
```

説明

① ｱｾﾝﾌﾞﾗは定数整数式を内部的に64ﾋﾞｯﾄ符号付き整数として評価します。このような式が即値被演算子(ｵﾍﾟﾗﾝﾄﾞ)として使われる場合、それらは命令によって必要とされるﾋﾞｯﾄ数に切り詰められなければなりません。殆どの被演算子に関し、範囲外の値は"被演算子範囲外"誤りﾒｯｾｰｼﾞを引き起こします。けれども、LDI、CPI、ORI、ANDI、SUBI、SBCIの命令(http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdfもご覧ください。)に対する即値ﾊﾞｲﾄ被演算子はこの任意選択によって影響を及ぼされる可能性があるいくつかの解釈を持ちます。

- 任意選択 = integer : 即値被演算子は整数として評価され、それの値が-128～255の範囲外の場合に警告が与えられます。ｱｾﾝﾌﾞﾗは使用者が符号付きまたは符号なしのどちらの整数被演算子を意図しているかを知らず、従ってﾊﾞｲﾄに合う符号付きと符号なしのどの値も許します。
- 任意選択 = overflow (既定) : 即値被演算子は基本的に符号なしﾊﾞｲﾄとして評価され、どの符号拡張ﾋﾞｯﾄも無視されます。この任意選択は特に整数解釈がLDI R16, ~((1<<7)|(1<<3))のように多くの警告を引き起こすﾋﾞｯﾄ遮蔽を扱う時に適します。
- 任意選択 = none : ﾊﾞｲﾄ被演算子に対する全ての範囲外警告を禁止。推奨されません。

② ｺｰﾄﾞの違う区部(ｾｸｼｮﾝ)が、.ORG疑似命令を使って重なっているﾒﾓﾘ位置に割り当てられる場合、通常、誤りﾒｯｾｰｼﾞが発行されます。この任意選択はその動きを次のように変更します。

- **任意選択** = ignore ：重複状況を完全に無視、誤りなし、警告なし。推奨されません。
- **任意選択** = warning ：重複検出時に警告を生成
- **任意選択** = error ：重複を誤りと見做します。これが既定で推奨設定です。
- **任意選択** = default ：既定の処理を(上の)errorか、または-0ｺﾏﾝﾄﾞ行任意選択で指定されたものに戻します。

ｱｾﾝﾌﾞﾗは重複処理に対して、-0ｺﾏﾝﾄﾞ行任意選択によって構成設定される既定設定と、この#pragmaでだけ変更することができる有効な設定の2つの設定を維持します。2つの設定はｱｾﾝﾌﾞﾗの実施に於いて等価です。この#pragmaはそれが実施される行からこの#pragmaの別の実施によってそれが変更される行まで有効な設定を変更します。従って、この#pragmaはｱﾄﾞﾚｽ範囲ではなく、ｿｰｽ行範囲を網羅します。「OVERLAPとNOOVERLAP」もご覧ください。

③ 指定したﾃﾞﾊﾞｲｽで未支援の命令を使うことがｱｾﾝﾌﾞﾗの誤り(既定の動き)を引き起こさせます。

④ 指定したﾃﾞﾊﾞｲｽで未支援の命令を使うことがｱｾﾝﾌﾞﾗの警告を引き起こさせます。

6.11. #pragma AVRﾃﾞﾊﾞｲｽ関連

構文

```
① #pragma AVRPART ADMIN PART_NAME 文字列
② #pragma AVRPART CORE CORE_VERSION 版文字列
③ #pragma AVRPART CORE INSTRUCTIONS_NOT_SUPPORTED ﾆｰﾓﾆｯｸ[ 被演算子[,被演算子] ][:～]
④ #pragma AVRPART CORE NEW_INSTRUCTIONS ﾆｰﾓﾆｯｸ[ 被演算子[,被演算子] ][:～]
⑤ #pragma AVRPART MEMORY PROG_FLASH 大きさ
⑥ #pragma AVRPART MEMORY EEPROM 大きさ
⑦ #pragma AVRPART MEMORY INT_SRAM SIZE 大きさ
⑧ #pragma AVRPART MEMORY INT_SRAM START_ADDR ｱﾄﾞﾚｽ
⑨ #pragma ﾃﾞﾊﾞｲｽ ｲﾝｸﾙｰﾄﾞ(数値)
```

説明

これらの疑似命令は様々なﾃﾞﾊﾞｲｽ特有特性を指定するのに使われ、通常はﾃﾞﾊﾞｲｽ定義ｲﾝｸﾙｰﾄﾞ(型番def.inc)ﾌｧｲﾙで使われます。通常、これらの#pragmaを使用者ﾌﾟﾛｸﾞﾗﾑで直接使う理由はありません。

前処理部ﾏｸﾛは#pragmaで許されません。式は数値引数に対して許されず、10進、16進、8進、または2進形式での純粋な数値でなければなりません。文字列引数は引用符で囲まれてはなりません。#pragmaは以下のﾃﾞﾊﾞｲｽ特有特性を指定します。

① ﾃﾞﾊﾞｲｽ名、例えば、ATmega8

② AVRｺｱ版。これは支援される基本命令一式を定義します。許されるｺｱ版は現在、V0、V0E、V1、V2、V2Eです。

③ ｺｱ版に対して、このﾃﾞﾊﾞｲｽによって支援されない、ﾌﾙｺﾛﾝ(:)で分離された命令の一覧

④ ｺｱ版に対して、このﾃﾞﾊﾞｲｽによって支援される、ﾌﾙｺﾛﾝ(:)で分離された追加命令の一覧

⑤ ﾊﾞｲﾄでのﾌﾗｯｼｭ ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘの大きさ

⑥ ﾊﾞｲﾄでのEEPROMの大きさ

⑦ ﾊﾞｲﾄでのSRAMの大きさ

⑧ SRAM開始ｱﾄﾞﾚｽ。基本的な旧AVRﾃﾞﾊﾞｲｽに対して0x60、拡張入出力を持つﾃﾞｨﾊﾞｽに対して0x100またはそれ以上

⑨ AVRASM1互換用。既定値:0。1に設定される場合、上で記述された#pragmaを含むことが期待されるdevice.hと呼ばれるﾌｧｲﾙをｲﾝｸﾙｰﾄﾞする.DEVICE疑似命令を引き起こします。これは(.DEVICE疑似命令を含むけれどもﾃﾞﾊﾞｲｽ記述の#pragmaを含まない)AVRASM1用に設計された型番def.incﾌｧｲﾙをAVRASM2で使うことを許します。この属性は「AVRｱｾﾝﾌﾞﾗ ｺﾏﾝﾄﾞ行任意選択」ｺﾏﾝﾄﾞ行任意選択を使って設定することもできます。

例

これらの例の組み合わせが実際のAVRﾃﾞﾊﾞｲｽを記述していないことに注意してください。

```
① #pragma AVRPART ADMIN PART_NAME ATmega32
② #pragma AVRPART CORE CORE_VERSION V2
③ #pragma AVRPART CORE INSTRUCTIONS_NOT_SUPPORTED movw:break:lpm rd,z
④ #pragma AVRPART CORE NEW_INSTRUCTIONS lpm rd,z+
⑤ #pragma AVRPART MEMORY PROG_FLASH 131072
⑥ #pragma AVRPART MEMORY EEPROM 4096
⑦ #pragma AVRPART MEMORY INT_SRAM SIZE 4096
⑧ #pragma AVRPART MEMORY INT_SRAM START_ADDR 0x60
```

6.12. # (空疑似命令)

構文

```
#
```

説明

当然、この疑似命令は全く何もしません。これが存在する理由はANSI C規格によって必要とされることだけです。

6.13. 演算子

6.13.1. 文字列化 (#)

文字列化演算子はパラメータの引用符付き文字列通票を関数型マクロにします。

例

```
#define MY_IDENT(X) .db #X, '¥n', 0
```

次のように呼ばれると、

```
MY_IDENT(FooFirmwareRev1)
```

は以下のように展開されます。

```
.db "FooFirmwareRev1", '¥n', 0
```

注: 1. 文字列化は関数型マクロへのパラメータでだけ使うことができます。

2. パラメータの値は文字どおりに使われ、即ち、文字列化の前に展開されません。

6.13.2. 連結 (##)

連結演算子は2つの前処理部通票を連結し、新しい通票を形成します。これは少なくても通票の1つが関数型マクロへのパラメータの時に最も有用です。

例

```
   #define FOOBAR subi
① #define IMMED(X) X##i
② #define SUBI(X,Y) X ## Y
```

IMMEDとSUBIのマクロが次のように呼ばれると、

```
① IMMED(ld) r16,1
② SUBI(FOO,BAR) r16,1
```

これらは以下のように展開されます。

```
① ldi r16,0x1
② subi r16,0x1
```

注: 1. 連結演算子はマクロ展開で最初または最後に現れることができません。

2. 関数型マクロ引数で使われると、その引数は文字どおりに使われ、即ち、連結前に展開されません。

3. 連結によって形成された通票は更なる展開を仮定します。上の例②ではパラメータのFOOとBARが最初にFOOBARに連結され、その後にsubiへ展開されます。

6.14. 予め定義されたﾏｸﾛ

前処理部は予め定義された多数のﾏｸﾛを持ちます。全てが2つのｱﾝﾀﾞｰｽｺｱ(__)で始まって終わる名前を持ちます。競合を避けるため、使用者定義ﾏｸﾛはこの命名則を使うべきではありません。

予め定義されたﾏｸﾛは下表で示されるように、組み込み、またはそれらが#pragma汎用疑似命令によって設定されるかのどちらかです。

名前	型	以下によって設定	説明
__AVRASM_VERSION__	整数	組み込み	ｱｾﾝﾌﾞﾗ版、(主×1000+副)として符号化
__CORE_VERSION__	文字列	#pragma汎用	AVRｺｱ版
__DATE__	文字列	組み込み	構築日付。形式: "Jun 28 2004"。 「AVRｱｾﾝﾌﾞﾗ ｺﾏﾝﾄﾞ行任意選択」をご覧ください。
__TIME__	文字列	組み込み	構築時間。形式: "HH:MM:SS"。 「AVRｱｾﾝﾌﾞﾗ ｺﾏﾝﾄﾞ行任意選択」をご覧ください。
__CENTURY__	整数	組み込み	構築時間の世紀 (代表的に20)
__YEAR__	整数	組み込み	構築時間の年、世紀の下 (0～99)
__MONTH__	整数	組み込み	構築時間の月 (1～12)
__DAY__	整数	組み込み	構築時間の日 (1～31)
__HOUR__	整数	組み込み	構築時間の時 (0～23)
__MINUTE__	整数	組み込み	構築時間の分 (0～59)
__SECOND__	整数	組み込み	構築時間の秒 (0～59)
__FILE__	文字列	組み込み	ソース ファイル名
__LINE__	整数	組み込み	ソース ファイル内の現在の行番号
__PART_NAME__	文字列	#pragma汎用	AVRﾃﾞﾊﾞｲｽ名
__partname__	整数	#pragma汎用	__PART_NAME__の値に対応するﾃﾞﾊﾞｲｽ名。 例: `#ifdef __ATmega8__`
__CORE_coreversion__	整数	#pragma汎用	__CORE_VERSION__の値に対応する版。 例: `#ifdef __CORE_V2__`

7. 式

ｱｾﾝﾌﾞﾗは定数式を利用します。

式は非演算子(ｵﾍﾟﾗﾝﾄﾞ)、演算子(ｵﾍﾟﾚｰﾀ)、それと関数から成ることができます。全ての式は内部的に64ﾋﾞｯﾄです。

7.1. 関数

ｱｾﾝﾌﾞﾗで定義された関数

- LOW(式)は式の下位ﾊﾞｲﾄを返します。
- HIGH(式)は式の第2ﾊﾞｲﾄを返します。
- BYTE2(式)は式のHIGHと同じ関数です。
- BYTE3(式)は式の第3ﾊﾞｲﾄを返します。
- BYTE4(式)は式の第4ﾊﾞｲﾄを返します。
- LWRD(式)は式の15～0ﾋﾞｯﾄを返します。
- HWRD(式)は式の31～16ﾋﾞｯﾄを返します。
- PAGE(式)は式の21～16ﾋﾞｯﾄを返します。
- EXP2(式)は式の2のべき乗を返します。
- LOG2(式)はlog2(式)の整数部を返します。
- INT(式)は浮動小数点式を整数に切り詰めます(即ち、小数部分を破棄)。
- FRAC(式)は浮動小数点式の小数部を抽出します(即ち、整数部を破棄)。
- Q7(式)は浮動小数点式をFMUL/FMULU/FMULSUの命令に適合する形式に変換します(http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdfもご覧ください)。(符号+7ﾋﾞｯﾄ小数)
- Q15(式)は浮動小数点式をFMUL/FMULU/FMULSUの命令に適合する形式に変換します(http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdfもご覧ください)。(符号+15ﾋﾞｯﾄ小数)
- ABS(式)は定数式の絶対値を返します。
- DEFINED(ｼﾝﾎﾞﾙ)はｼﾝﾎﾞﾙが.EQU/.SET/.DEFの疑似命令を使って以前に定義されている場合に真を返します。通常、.IF疑似命令(.IF defined(～))と共に使われますが、どの脈絡でも使われ得ます。これはそれの引数の周りの括弧が必要とされないことと、引数として単一のｼﾝﾎﾞﾙを使うことだけに意味があることで、他の関数と異なります。
- STRLEN(文字列)は文字列定数の長さをﾊﾞｲﾄで返します。

7.2. 被演算子

以下の被演算子(ｵﾍﾟﾗﾝﾄﾞ)を使うことができます。

- 使用者定義ﾗﾍﾞﾙ、これはそれらが現れた場所の位置計数器の値が与えられます。
- .SET疑似命令によって定義された使用者定義変数
- .EQU疑似命令によって定義された使用者定義定数
- 整数定数: 定数は以下を含むいくつかの形式で与えることができます。
  - 10進数 (既定) : 10, 255
  - 16進数 (2つの表記法) : 0x0A, $0A, 0xFF, $FF
  - 2進数 : 0b00001010, 0b11111111
  - 8進数 (0先行) : 010, 077
- PC - ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ位置計数器の現在値
- 浮動小数点定数

7.3. 演算子

ｱｾﾝﾌﾞﾗはここで記述される多数の演算子(ｵﾍﾟﾚｰﾀ)を支援します。より高い優先順位はより高い優先権です。式は括弧で囲まれるかもしれず、そのような式は常に括弧の外側の何かと結合される前に評価されます。二項演算子の結合規則は連結された演算子の評価順を示し、左結合は左から右、即ち、2-3-4は(2-3)-4に評価されることを意味し、一方で右結合は2-3-4が2-(3-4)なことを意味します。いくつかの演算子は結合がなく、連結が意味を持たないことを意味します。

次の演算子が定義されています。

シンボル	説明	シンボル	説明
!	論理否定(NOT)	<=	以下
~	ビット単位否定(NOT)	>	超える
-	単項負(マイナス)	>=	以上
*	乗算	==	等しい(イコール)
/	除算	!=	等しくない(ノット イコール)
%	剰余 (AVRASM2のみ)	&	ビット単位論理積(AND)
+	加算	^	ビット単位排他的論理和(XOR)
-	減算	|	ビット単位論理和(OR)
<<	左移動	&&	論理積(AND)
>>	右移動	||	論理和(OR)
<	未満	?	条件付き演算子

論理否定(NOT)

シンボル　：!
説明　　：単項演算子、式が0だった場合に1を返し、式が0以外だった場合に0に返します。
優先順位：14
結合規則：なし
例　　：
```
LDI R16,!0xF0                    ; R16に$00を設定
```

ビット単位否定(NOT)

シンボル　：~
説明　　：単項演算子、全ビットが反転された入力式を返します。
優先順位：14
結合規則：なし
例　　：
```
LDI R16,~0xF0                    ; R16に$0Fを設定
```

単項負(マイナス)

シンボル　：-
説明　　：単項演算子、式の算術否定を返します。
優先順位：14
結合規則：なし
例　　：
```
LDI R16,-2                       ; R16に-2($FE)を設定
```

乗算

シンボル　：*
説明　　：二項演算子、2つの式の積を返します。
優先順位：13
結合規則：左
例　　：
```
LDI R30,label*2                  ; R30にlabel×2を設定
```

除算

シンボル　：/
説明　　：二項演算子、左式を右式で割った整数の商を返します。
優先順位：13
結合規則：左
例　　：
```
LDI R30,label/2                  ; R30にlabel/2を設定
```

剰余

シンボル　：%
説明　　：二項演算子、左式を右式で割った整数の余りを返します。
優先順位：13
結合規則：左
例　　：
```
LDI R30,label%2                 ; R30にlabel%2を設定
```

加算

シンボル　：+
説明　　：二項演算子、2つの式の和(合計)を返します。
優先順位：12
結合規則：左
例　　：
```
 LDI R30,c1+c2                  ; R30にc1+c2を設定
```

減算

シンボル　：-
説明　　：二項演算子、左式から右式を引いた値を返します。
優先順位：12
結合規則：左
例　　：
```
 LDI R17,c1-c2                  ; R17にc1-c2を設定
```

左移動

シンボル　：<<
説明　　：二項演算子、右式によって与えられた数だけ左移動した左式を返します。
優先順位：11
結合規則：左
例　　：
```
 LDI R17,1<<bitmask             ; R17に1をbitmask回左移動して設定
```

右移動

シンボル　：>>
説明　　：二項演算子、右式によって与えられた数だけ右移動した左式を返します。
優先順位：11
結合規則：左
例　　：
```
 LDI R17,c1>>c2                 ; R17にc2回右移動したc1を設定
```

未満

シンボル　：<
説明　　：二項演算子、左の符号付きの式が右の符号付きの式未満の場合に1、そうでなければ0を返します。
優先順位：10
結合規則：なし
例　　：
```
 ORI R18,bitmask*(c1<c2)+1      ; R18を式と論理和
```

以下

シンボル　：<=
説明　　：二項演算子、左の符号付きの式が右の符号付きの式以下の場合に1、そうでなければ0を返します。
優先順位：11
結合規則：左
例　　：
```
 ORI R18,bitmask*(c1<=c2)+1     ; R18を式と論理和
```

超える

ｼﾝﾎﾞﾙ　：>
説明　　：二項演算子、左の符号付きの式が右の符号付きの式を超える場合に1、そうでなければ0を返します。
優先順位：10
結合規則：なし
例　　：
```
ORI R18,bitmask*(c1>c2)+1       ; R18を式と論理和
```

以上

ｼﾝﾎﾞﾙ　：>=
説明　　：二項演算子、左の符号付きの式が右の符号付きの式以上の場合に1、そうでなければ0を返します。
優先順位：10
結合規則：なし
例　　：
```
ORI R18,bitmask*(c1>=c2)+1      ; R18を式と論理和
```

等しい(ｲｺｰﾙ)

ｼﾝﾎﾞﾙ　：==
説明　　：二項演算子、左の符号付きの式が右の符号付きの式と等しければ1、そうでなければ0を返します。
優先順位：9
結合規則：なし
例　　：
```
ANDI R19,bitmask*(c1==c2)+1     ; R19を式と論理積
```

等しくない(ﾉｯﾄ ｲｺｰﾙ)

ｼﾝﾎﾞﾙ　：!=
説明　　：二項演算子、左の符号付きの式が右の符号付きの式と等しくなければ1、それ以外は0を返します。。
優先順位：9
結合規則：なし
例　　：
```
.SET flag=(c1!=c2)              ; flagに1または0を設定(定義)
```

ﾋﾞｯﾄ単位論理積(AND)

ｼﾝﾎﾞﾙ　：&
説明　　：二項演算子、2つの式のﾋﾞｯﾄ単位論理積(AND)を返します。
優先順位：8
結合規則：左
例　　：
```
LDI R18,high(c1&c2)             ; R18に式を設定
```

ﾋﾞｯﾄ単位排他的論理和(XOR)

ｼﾝﾎﾞﾙ　：^
説明　　：二項演算子、2つの式のﾋﾞｯﾄ単位排他的論理和(XOR)を返します。
優先順位：7
結合規則：左
例　　：
```
LDI R18,low(c1^c2)              ; R18に式を設定
```

ﾋﾞｯﾄ単位論理和(OR)

ｼﾝﾎﾞﾙ　：|
説明　　：二項演算子、2つの式のﾋﾞｯﾄ単位論理和(OR)を返します。
優先順位：6
結合規則：左
例　　：
```
LDI R18,low(c1|c2)              ; R18に式を設定
```

論理積(AND)

シンボル　：&&
説明　　：二項演算子、左が両方共に0以外ならば1、そうでなければ0を返します。
優先順位：5
結合規則：左
例　　：
```
LDI R18,low(c1&&c2)             ; R18に式を設定
```

論理和(OR)

シンボル　：||
説明　　：二項演算子、1つまたは両方の式が0以外ならば1、そうでなければ0を返します。
優先順位：4
結合規則：左
例　　：
```
LDI R18,low(c1||c2)             ; R18に式を設定
```

条件付き演算子

シンボル　：?:
構文　　：
```
条件? 式1 : 式2
```
説明　　：三項演算子、条件が真ならば式1を、そうでなければ式2を返します。
優先順位：3
結合規則：なし
例　　：
```
LDI R18, a > b? a : b           ; R18にaとbの最大数値を設定
```
注: この機能はAVRASM2.1で導入され、2.0またはそれ以前の版では利用不可です。

8. AVR命令一式

AVR命令一式についての情報に関しては8ビットAVR命令一式手引書を参照してください。

9. 改訂履歴

資料改訂	日付	注釈
1022A	1998年1月	初版資料公開
A	2017年6月	資料の完全更新 Microchip形式に変換してAtmel資料番号1022を置き換え

Microchipウェブ サイト

Microchipはhttp://www.microchip.com/で当社のウェブ サイト経由でのオンライン支援を提供します。このウェブ サイトはお客様がファイルや情報を容易に利用可能にする手段として使われます。お気に入りのインターネット ブラウザを用いてアクセスすることができ、ウェブ サイトは以下の情報を含みます。

- **製品支援** – データシートと障害情報、応用記述と試供プログラム、設計資源、使用者の手引きとハードウェア支援資料、最新ソフトウェア配布と保管されたソフトウェア
- **全般的な技術支援** – 良くある質問(FAQ)、技術支援要求、オンライン検討グループ、Microchip相談役プログラム員一覧
- **Microchipの事業** – 製品選択器と注文の手引き、最新Microchip報道発表、セミナーとイベントの一覧、Microchip営業所の一覧、代理店と代表する工場

お客様への変更通知サービス

Microchipのお客様通知サービスはMicrochip製品を最新に保つのに役立ちます。加入者は指定した製品系統や興味のある開発ツールに関連する変更、更新、改訂、障害情報がある場合に必ず電子メール通知を受け取ります。

登録するにはhttp://www.microchip.com/でMicrochipのウェブ サイトをアクセスしてください。”Support”下で”Customer Change Notification”をクリックして登録指示に従ってください。

お客様支援

Microchip製品の使用者は以下のいくつかのチャネルを通して支援を受け取ることができます。

- 代理店または販売会社
- 最寄りの営業所
- 現場応用技術者(FAE:Field Aplication Engineer)
- 技術支援

お客様は支援に関してこれらの代理店、販売会社、または現場応用技術者(FAE)に連絡を取るべきです。最寄りの営業所もお客様の手助けに利用できます。営業所と位置の一覧はこの資料の後ろに含まれます。

技術支援はhttp://www.microchip.com/supportでのウェブ サイトを通して利用できます。

Microchipデバイス コード保護機能

Microchipデバイスでの以下のコード保護機能の詳細に注意してください。

- Microchip製品はそれら特定のMicrochipデータシートに含まれる仕様に合致します。
- Microchipは意図した方法と通常条件下で使われる時に、その製品系統が今日の市場でその種類の最も安全な系統の1つであると考えます。
- コード保護機能を破るのに使われる不正でおそらく違法な方法があります。当社の知る限りこれらの方法の全てはMicrochipのデータシートに含まれた動作仕様外の方法でMicrochip製品を使うことが必要です。おそらく、それを行う人は知的財産の窃盗に関与しています。
- Microchipはそれらのコードの完全性について心配されているお客様と共に働きたいと思います。
- Microchipや他のどの半導体製造業者もそれらのコードの安全を保証することはできません。コード保護は当社が製品を”破ることができない”として保証すると言うことを意味しません。

コード保護は常に進化しています。Microchipは当社製品のコード保護機能を継続的に改善することを約束します。Microchipのコード保護機能を破る試みはデジタル ミレニアム著作権法に違反するかもしれません。そのような行為があなたのソフトウェアや他の著作物に不正なアクセスを許す場合、その法律下の救済のために訴権を持つかもしれません。

法的通知

デバイス応用などに関してこの刊行物に含まれる情報は皆さまの便宜のためにだけ提供され、更新によって取り換えられるかもしれません。皆さまの応用が皆さまの仕様に合致するのを保証するのは皆さまの責任です。**Microchipはその条件、品質、性能、商品性、目的適合性を含め、明示的にも黙示的にもその情報に関連して書面または表記された書面または黙示の如何なる表明や保証もしません。**Microchipはこの情報とそれの使用から生じる全責任を否認します。生命維持や安全応用でのMicrochipデバイスの使用は完全に購入者の危険性で、購入者はそのような使用に起因する全ての損害、請求、訴訟、費用からMicrochipを擁護し、補償し、免責にすることに同意します。他に言及されない限り、Microchipのどの知的財産権下でも暗黙的または違う方法で許認可は譲渡されません。

商標

DNVによって認証された品質管理ｼｽﾃﾑ

ISO/TS 16949

Microchipはその世界的な本社、ｱﾘｿﾞﾅ州のﾁｬﾝﾄﾞﾗｰとﾃﾝﾍﾟ、ｵﾚｺﾞﾝ州ｸﾞﾗｼｬﾑの設計とｳｪﾊｰ製造設備とｶﾘﾌｫﾙﾆｱとｲﾝﾄﾞの設計ｾﾝﾀｰに対してISO/TS-16949:2009認証を取得しました。当社の品質ｼｽﾃﾑの処理と手続きはPIC® MCUとdsPIC® DSC、KEELOQ符号飛び回りﾃﾞﾊﾞｲｽ、直列EEPROM、ﾏｲｸﾛ周辺機能、不揮発性ﾒﾓﾘ、ｱﾅﾛｸﾞ製品用です。加えて、開発ｼｽﾃﾑの設計と製造のためのMicrochipの品質ｼｽﾃﾑはISO 9001:2000認証取得です。

日本語© *HERO* 2020.

本応用記述はMicrochipのAVRｱｾﾝﾌﾞﾗ使用者の手引き(DS40001917A-2017年6月)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。

世界的な販売とサービス

米国	亜細亜/太平洋	亜細亜/太平洋	欧州
本社 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 技術支援: http://www.microchip.com/support ｳｪﾌﾞｱﾄﾞﾚｽ: www.microchip.com **ｱﾄﾗﾝﾀ** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **ｵｰｽﾁﾝ** TX Tel: 512-257-3370 **ﾎﾞｽﾄﾝ** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **ｼｶｺﾞ** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **ﾀﾞﾗｽ** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **ﾃﾞﾄﾛｲﾄ** Novi, MI Tel: 248-848-4000 **ﾋｭｰｽﾄﾝ** TX Tel: 281-894-5983 **ｲﾝﾃﾞｱﾅﾎﾟﾘｽ** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **ﾛｻﾝｾﾞﾙｽ** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **ﾛｰﾘｰ** NC Tel: 919-844-7510 **ﾆｭｰﾖｰｸ** NY Tel: 631-435-6000 **ｻﾝﾎｾ** CA Tel: 408-735-9110 Tel: 408-436-4270 **ｶﾅﾀﾞ - ﾄﾛﾝﾄ** Tel: 905-695-1980 Fax: 905-695-2078	**亜細亜太平洋支社** Suites 3707-14, 37th Floor Tower 6, The Gateway Harbour City, Kowloon **香港** Tel: 852-2943-5100 Fax: 852-2401-3431 **ｵｰｽﾄﾗﾘｱ - ｼﾄﾞﾆｰ** Tel: 61-2-9868-6733 Fax: 61-2-9868-6755 **中国 - 北京** Tel: 86-10-8569-7000 Fax: 86-10-8528-2104 **中国 - 成都** Tel: 86-28-8665-5511 Fax: 86-28-8665-7889 **中国 - 重慶** Tel: 86-23-8980-9588 Fax: 86-23-8980-9500 **中国 - 東莞** Tel: 86-769-8702-9880 **中国 - 広州** Tel: 86-20-8755-8029 **中国 - 杭州** Tel: 86-571-8792-8115 Fax: 86-571-8792-8116 **中国 - 香港特別行政区** Tel: 852-2943-5100 Fax: 852-2401-3431 **中国 - 南京** Tel: 86-25-8473-2460 Fax: 86-25-8473-2470 **中国 - 青島** Tel: 86-532-8502-7355 Fax: 86-532-8502-7205 **中国 - 上海** Tel: 86-21-3326-8000 Fax: 86-21-3326-8021 **中国 - 瀋陽** Tel: 86-24-2334-2829 Fax: 86-24-2334-2393 **中国 - 深圳** Tel: 86-755-8864-2200 Fax: 86-755-8203-1760 **中国 - 武漢** Tel: 86-27-5980-5300 Fax: 86-27-5980-5118 **中国 - 西安** Tel: 86-29-8833-7252 Fax: 86-29-8833-7256	**中国 - 廈門** Tel: 86-592-2388138 Fax: 86-592-2388130 **中国 - 珠海** Tel: 86-756-3210040 Fax: 86-756-3210049 **ｲﾝﾄﾞ - ﾊﾝｶﾞﾛｰﾙ** Tel: 91-80-3090-4444 Fax: 91-80-3090-4123 **ｲﾝﾄﾞ - ﾆｭｰﾃﾞﾘｰ** Tel: 91-11-4160-8631 Fax: 91-11-4160-8632 **ｲﾝﾄﾞ - ﾌﾟﾈｰ** Tel: 91-20-3019-1500 **日本 - 大阪** Tel: 81-6-6152-7160 Fax: 81-6-6152-9310 **日本 - 東京** Tel: 81-3-6880-3770 Fax: 81-3-6880-3771 **韓国 - 大邱** Tel: 82-53-744-4301 Fax: 82-53-744-4302 **韓国 - ｿｳﾙ** Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934 **ﾏﾚｰｼｱ - ｸｱﾗﾙﾝﾌﾟｰﾙ** Tel: 60-3-6201-9857 Fax: 60-3-6201-9859 **ﾏﾚｰｼｱ - ﾍﾟﾅﾝ** Tel: 60-4-227-8870 Fax: 60-4-227-4068 **ﾌｨﾘﾋﾟﾝ - ﾏﾆﾗ** Tel: 63-2-634-9065 Fax: 63-2-634-9069 **ｼﾝｶﾞﾎﾟｰﾙ** Tel: 65-6334-8870 Fax: 65-6334-8850 **台湾 - 新竹** Tel: 886-3-5778-366 Fax: 886-3-5770-955 **台湾 - 高雄** Tel: 886-7-213-7830 **台湾 - 台北** Tel: 886-2-2508-8600 Fax: 886-2-2508-0102 **ﾀｲ - ﾊﾞﾝｺｸ** Tel: 66-2-694-1351 Fax: 66-2-694-1350	**ｵｰｽﾄﾘｱ - ｳﾞｪﾙｽ** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **ﾃﾞﾝﾏｰｸ - ｺﾍﾟﾝﾊｰｹﾞﾝ** Tel: 45-4450-2828 Fax: 45-4485-2829 **ﾌｨﾝﾗﾝﾄﾞ - ｴｽﾎﾟｰ** Tel: 358-9-4520-820 **ﾌﾗﾝｽ - ﾊﾟﾘ** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **ﾌﾗﾝｽ - ｻﾝ ｸﾙｰ** Tel: 33-1-30-60-70-00 **ﾄﾞｲﾂ - ｶﾞﾙﾋﾝｸﾞ** Tel: 49-8931-9700 **ﾄﾞｲﾂ - ﾊｰﾝ** Tel: 49-2129-3766400 **ﾄﾞｲﾂ - ﾊｲﾙﾌﾞﾛﾝ** Tel: 49-7131-67-3636 **ﾄﾞｲﾂ - ｶｰﾙｽﾙｰｴ** Tel: 49-721-625370 **ﾄﾞｲﾂ - ﾐｭﾝﾍﾝ** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **ﾄﾞｲﾂ - ﾛｰｾﾞﾝﾊｲﾑ** Tel: 49-8031-354-560 **ｲｽﾗｴﾙ - ﾗｰﾅﾅ** Tel: 972-9-744-7705 **ｲﾀﾘｱ - ﾐﾗﾉ** Tel: 39-0331-742611 Fax: 39-0331-466781 **ｲﾀﾘｱ - ﾊﾟﾄﾞﾊﾞ** Tel: 39-049-7625286 **ｵﾗﾝﾀﾞ - ﾃﾞﾙｰﾈﾝ** Tel: 31-416-690399 Fax: 31-416-690340 **ﾉﾙｳｪｰ - ﾄﾛﾝﾊｲﾑ** Tel: 47-7289-7561 **ﾎﾟｰﾗﾝﾄﾞ - ﾜﾙｼｬﾜ** Tel: 48-22-3325737 **ﾙｰﾏﾆｱ - ﾌﾞｶﾚｽﾄ** Tel: 40-21-407-87-50 **ｽﾍﾟｲﾝ - ﾏﾄﾞﾘｰﾄﾞ** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **ｽｳｪｰﾃﾞﾝ - ｲｪｰﾃﾎﾞﾘ** Tel: 46-31-704-60-40 **ｽｳｪｰﾃﾞﾝ - ｽﾄｯｸﾎﾙﾑ** Tel: 46-8-5090-4654 **ｲｷﾞﾘｽ - ｳｫｰｷﾝｶﾞﾑ** Tel: 44-118-921-5800 Fax: 44-118-921-5820