MIC8S 内核系列

**MIC8S_ASM**

文档类型
文档名称
版本号: V1.1
日期: 2015/06/26

# MIC8S_ASM 汇编器使用说明

## 内容概述

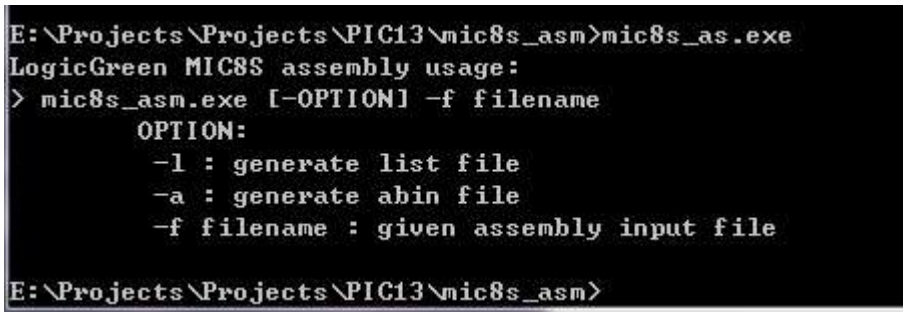
MIC8S_ASM 为 MIC8S 内核系列专用汇编器，工具读入 MIC8S 汇编指令文件，编译生成用于 MIC8S 执行的机器指令码

## MIC8S_ASM 功能概述

- 输入汇编文件的语法检测
- 生成 HEX 以及用于仿真的 ABIN 文件
- 生成中间格式 LIST 文件

## MIC8S_ASM 使用说明

MIC8S_ASM 为命令行工具，目前只支持在 WINDOWS 操作系统；
在命令下输出 mic8s_asm，默认得到如下提示：




如图所示，mic8s_asm 必须至少指定一个汇编源文件，目前 mic8s_asm 暂时只支持一个输入文件；下面是 mic8s_asm 的参数列表：

| 参数 | 功能描述 |
|---|---|
| -l | 产生 list 文件 |
| -a | 产生用于仿真的 ABIN 文件 |
| -f filename | 指定输入汇编文件 |

# *MIC8S_ASM 汇编语法*

MIC8S_ASM 是专用于 MIC8S 内核的汇编器，支持 MIC8S 所有指令。关于 MIC8S 指令集的定义请参考《MIC8S 内核设计文档》中指令集定义部分。

MIC8S_ASM 不区分大小写，建议书写代码统一使用小写方式。
MIC8S_ASM 仅支持以分号(；)开始的单行注释文档

为方便编写汇编程序， MIC8S_ASM 支持其他常用汇编语法，可以实现标识符定义，地址标签，程序空间的数据定义等等。以下列表为 MIC8S_ASM 支持的常用标识符：

| Symbol | *Function* |
|---|---|
| *.org* | *指定代码段的起始地址* |
| *.local* | *定义标识符, 局部有效* |
| *.global* | *定义全局标识符* |
| *.db* | *程序空间的数据字节数据定义* |
| *$* | *当前程序地址* |

MIC8S_AS 支持 8/10/16 进制数据，数据格式定义如下：

| 数据进制 | *格式* |
|---|---|
| 8 进制 | *O 或 Q 结尾，如 08Q, 08O* |
| 16 进制 | *0x 开头，或者 h 结尾，如 0x80, 05h* |
| 10 进制 | *不带任何修饰的数据，如，80， 05* |

## **.org** 使用说明

.org 可以在源文件的任何地方使用，用于重新定义一个程序段的开始地址。汇编代码的地址默认从 0 地址开始，所以 0 地址的段定义可以省略。

*示例代码*

```
        .org    000h            ;设置起始段地址，0 地址可以省略
        goto    start

        .org    080h            ;定义一个新的代码段
Start:
        nop
        goto    main
```

## .local/.global 使用说明

.local 用于定义局部符号，仅在一个文件的范围内有效；
.global 定义全局符号，可在一个项目的范围内有效，（目前暂不支持.global）
符号的定义可以在文档中的任何位置。

*示例代码*

```
        .org    000h                ;设置其实段地址，0 地址可以省略

.local  FSR     04h                 ;定义值为 04h 的局部符号 FSR
.local  LCR     07h                 ;定义值为 07h 的局部符号 LCR


        goto    start

start:
        movlw   055h                ; W = 0x55
        movwf   FSR                 ; DMEM[04h] = W
        goto    $                   ; death loop
```

## .db 使用说明

.db 用于在程序空间中定义一个字节的数据。我们将在后面的部分详细介绍如何使用这一功能。

*示例代码*

```
        .org    000h                ;设置其实段地址，0 地址可以省略

        .org    010h
.db     55h                         PMEM(10h) = 0x55
.db     aah                         PMEM(11h) = 0xAA
        goto    start               ; address of "goto": 012h

        .org    020h
start:
        movlt   010h                ; {PCH:FSR} = 0x010 point to data segment
        movwp                       ; W = 0x55
        goto    $                   ; death loop
```

# MIC8S 扩展指令应用

MIC8S 实现了 10 条扩展指令，用于高效程序流程控制和数据寻址。善于利用这些扩展指令，能够使用更少的代码，实现更多更复杂的功能，同时也提高软件的运行效率。

## LOOP/MOVLC 指令的使用

传统指令实现循环代码需要一个寄存器用于循环计数，每次循环之前装载循环技术值，然后在每次循环后更新计数值，并判断循环是否达到。对于 MIC8S 的基本指令，由于无法直接将立即数写入到寄存器空间，赋值操作比较繁琐，浪费代码空间且效率不高。而使用扩展的 LOOP 和 MOVLC 指令，可以快速高效的完成循环代码段的任务。

下面是使用 MIC8S 基本指令和使用扩展指令实现同样功能的对照：

*使用基本指令实现的循环操作*

```
            .org    000h                ;设置其实段地址，0 地址可以省略
.local      lcr     07h                 ; 定义 LCR 寄存器

            movlw   10h
            movwf   lcr                 ; 设置 LCR 作为循环计数
loop0:
            call    task                ; 执行循环任务
            decfsz  lcr,    1           ; 更新循环计数
            goto    end                 ; 循环结束
            goto    loop0               ; 下一次循环
task:
end:
            goto    $                   结束后位置
```

*使用扩展指令实现的循环操作*

```
            .org    000h                ;设置其实段地址，0 地址可以省略


            movlc   10h                 ; 设置 LCR 作为循环计数
loop0:
            call    task                ; 执行循环任务
            loop    loop0               ; 执行循环控制
end:
            goto    $                   ; 结束后位置
task:
```

## BRXZ/BRXC 条件跳转指令

MIC8S 基本指令中条件跳转指令仅仅有四条，INCFSZ/DECFSZ，BTFSC/BTFSS；其中 INCFSZ/DECFSZ 一般用于递增/递减的循环控制，而 BTFSC/BTFSS 用于根据寄存器的位状态控制程序执行流程；这两种条件跳转指令当条件满足后，将跳过下一条指令。MIC8S 扩展指令中增加了四条与状态位直接相关的条件跳转指令，可以直接根据状态位(C/Z)直接跳转到程序空间中任意指定的地址。

*使用基本指令实现的条件跳转*

```
            .org    000h                ;设置其实段地址，0 地址可以省略
.local      STATUS  03h                 ; 定义状态寄存器
.local      FLAG    10h                 ; 在 RAM 中定义一个变量

wait:
            movf    FLAG                ; 读 FLAG
            sublw   55h                 ; 55h – FLAG
            btfss   STATUS,  2          ; 判读 Z 标志位
            goto    do_flag             ; FLAG == 0x55
            goto    wait                ; 等待 FLAG
do_flag:
            goto    $
```

*使用扩展指令实现的条件跳转*

```
            .org    000h                ;设置其实段地址，0 地址可以省略
.local      FLAG    10h                 ; 在 RAM 中定义一个变量

wait:
            movf    FLAG                ; 读 FLAG
            sublw   55h                 ; 55h – FLAG
            brsz    do_flag
            goto    wait                ; 等待 FLAG
do_flag:
            goto    $
```

## IMOVW/IMOVF 指令的使用

MIC8S 基本指令集中，可以通过使用 FSR 与 INDF 寄存器实现间接寻址。在扩展指令中，增加了更加灵活的间接寻址方式，包括自动递增/递减，以及带偏移量的自动寻址模式。

当我们需要连续访问多个地址，或者在某一个地址访问内操作时，扩展的寻址指令具有极大的优势。当需要间接访问的数据越多时， 这种优势也会成倍的放大。

*使用基本指令实现的 RAM 访问*

```
            .org    000h                ;设置其实段地址，0 地址可以省略
.local      OP0     40h                 ; 在 RAM 中定义一个变量
.local      OP1     42h                 ;
.local      SUM     07h                 ; 可以直接寻址的寄存器
.local      FSR     05h                 ; 定义 FSR 寄存器
.local      INDF    00h                 ; 定义 INDF 寄存器

            movlw   OP0
            movwf   FSR                 ; 设置目前地址
            movf    INDF                ; 读目标地址数据
            movf    SUM,    1           ; SUM = OP0
            movlw   OP1
            movwf   FSR
            movf    INDF
            addwf   SUM,    1           ; SUM = OP0 + OP1
            goto    $                   ; END
```

*使用扩展指令实现的 RAM 访问*

```
            .org    000h                ;设置其实段地址，0 地址可以省略
.local      OP0     40h                 ; 在 RAM 中定义一个变量
.local      OP1     01h                 ; 这里定义的是偏移量
.local      SUM     07h                 ;

            movli   OP0                 ; 设置基地址
            imovw                       ; 读目标地址 OP0
            movf    SUM,    1           ; SUM = OP0
            imovw   +OP1
            addwf   SUM,    1           ; SUM = OP0 + OP1
            goto    $
```

## IMOVP 指令的使用

MIC8S 基本指令集中没有直接访问程序空间的方法，而是使用了 RETLW 这种间接的方法实现有限的访问功能。 下面我们以一个简单的数据查表为例，可以看出扩展指令的巨大优势。

*使用基本指令实现的程序空间访问*

```
            .org    000h                ;设置其实段地址，0 地址可以省略
.local      pcl     02h                 ; 查表指针
.local      fsr     05h                 ; 偏移地址
.local      lcr     07h                 ; 循环计数
.local      pch     0ah                 ;

main:
            movlw   01h
            movf    fsr,    1           ; 初始化偏移地址
            movlw   04h
            movf    lcr,    1           ; 初始化循环计数
            movlw   01h                 ;
            movf    pch,    1           ; 寻址到 0x100
lookup:
            call    table               ; 查表，数据返回到 W 寄存器
            call    display             ; 处理数据
            incf    fsr,    1           ; 递增地址
            decfsz  lcr,    1           ; 递减计数
            goto    looup               ; 循环查表
end:
            goto    $                   ; Death loop
display:
            nop                         ; 数据处理子程序
            return

            .org    100h
table:
            movf    fsr
            addwf   pcl,    1
            retlw   0x01
            retlw   0x02
            retlw   0x03
            retlw   0x04
```

*使用扩展指令实现程序空间访问*

```
            .org    000h                ;设置其实段地址，0 地址可以省略

main:
            movlt   100h                ; 目标地址
            movlc   04h                 ; 循环计数
lookup:
            movwp++                     ; 查表， 数据返回到 W 寄存器
            call    display             ; 数据处理
            loop    lookup              ;
end:
            goto    $

display:
            nop                         ; 数据处理子程序
            return

            .org    100h
            .db     01h
            .db     02h
            .db     03h
            .db     04h
```

# MIC8S 指令集定义

| 指令名称 | 操作 | 功能描述 | 指令字 | 状态位 | 周期 |
|---|---|---|---|---|---|
| 基本算术运算指令 | | | | | |
| SUBWF | SUBWF F[1], d[2] | [W/F] = (F) – W | 1 | C/DC/Z | 1 |
| DECF | DECF F, d | [W/F] = (F) – 1 | 1 | Z | 1 |
| ADDWF | ADDWF F, d | [W/F] = (F) + W | 1 | C/DC/Z | 1 |
| COMF | COMF F, d | [W/F] = complement F | 1 | Z | 1 |
| INCF | INCF F, d | [W/F] = (F) + 1 | 1 | Z | 1 |
| SUBLW | SUBLW k[4] | W = k – W | 1 | C/DC/Z | 1 |
| ADDLW | ADDLW k | W = k + W | 1 | C/DC/Z | 1 |
| ADCWF | ADCWF F, d | [W/F] = (F) + W + C | 1 | C/DC/Z | 1 |
| SBCWF | SBCWF F, d | [W/F] = (F) – W + C | 1 | C/DC/Z | 1 |
| DAA | DAA | Decimal correct for W after ADDX | 1 | Z | 1 |
| DAS | DAS | Decimal correct for W after SUBX | 1 | - | 1 |
| CLRF | CLRF F | (F) = 0 | 1 | Z | 1 |
| CLRW | CLRW | W = 0 | 1 | Z | 1 |
| 基本逻辑运行指令 | | | | | |
| IORWF | IORWF F, d | [W/F] = (F) \| W | 1 | Z | 1 |
| ANDWF | ANDWF F, d | [W/F] = (F) & W | 1 | Z | 1 |
| XORWF | XORWF F, d | [W/F] = (F) ^ W | 1 | Z | 1 |
| RRF | RRF F, d | [W/F] = {C, F[6:0]} | 1 | C | 1 |
| RLF | RLF F, d | [W/F] = {F[6:0], C} | 1 | C | 1 |
| SWAPF | SWAPF F, d | [W/F] = {F[3:0], F[7:4]} | 1 | - | 1 |
| BCF | BCF F, b[3] | F[b] = 0 | 1 | - | 1 |
| BSF | BSF F, b | F[b] = 1 | 1 | - | 1 |
| ANDLW | ANDLW k | W = k & W | 1 | Z | 1 |
| XORLW | XORLW k | W = k ^ W | 1 | Z | 1 |
| IORLW | IORLW k | W = k \| W | 1 | Z | 1 |
| 基本流程控制指令 | | | | | |
| DECFSZ | DECFSZ F, d | [W/F] = (F) – 1<br>IF(Z) PC = PC + 2 | 1 | - | 1/2 |
| INCFSZ | INCFSZ F, d | [W/F] = (F) + 1<br>IF(Z) PC = PC + 2 | 1 | - | 1/2 |
| BTFSC | BTFSC F, b | IF(!F[b]) PC = PC + 2 | 1 | - | 1/2 |
| BTFSS | BTFSS F, b | IF(F[b]) PC = PC + 2 | 1 | - | 1/2 |
| GOTO | GOTO k | PC = k | 1 | - | 2 |
| CALL | CALL k | PC+1 -> STACK<br>PC = k | 1 | - | 2 |
| RETURN | RETURN | PC <- STACK | 1 | - | 2 |
| RETFIE | RETFIE | GIE = 1<br>PC <- STACK | 1 | - | 2 |

| | | | | | |
|---|---|---|---|---|---|
| RETLW | RETLW k | W = k<br>PC <- STACK | 1 | - | 2 |
| 基本数据传输指令 | | | | | |
| MOVWF | MOVWF F | (F) = W | 1 | - | 1 |
| MOVF | MOVF F, d | [W/F] = (F) | 1 | Z | 1 |
| MOVLW | MOVLW k | W = k | 1 | - | 1 |
| 其他辅助指令 | | | | | |
| NOP | NOP | NO operation | 1 | - | 1 |
| OPTION | OPTION | OPTION = W | 1 | - | 1 |
| TRIS | TRIS F | IOSTA/B = W<br>(F=5/6) | 1 | - | 1 |
| SLEEP | SLEEP | Power/down MIC8S | 1 | TO/PD | 1 |
| CLRWDT | CLRWDT | Clear WDT | 1 | TO/PD | 1 |
| INT | INT | PC+1 -> STACK<br>PC = 0x2<br>GIE = 0 | 1 | - | 2 |
| 扩展数据传输指令 | | | | | |
| MOVLT | MOVLT k | PCH = k[9:8]<br>FSR = k[7:0] | 2 | - | 2 |
| MOVLC | MOVLC k | LCR = K[7:0] | 2 | - | 2 |
| MOVLI | MOVLI k | FSR = k[7:0] | 2 | - | 2 |
| MOVWP | MOVWP | W = PMEM[PCH:FSR] | 1 | - | 2 |
| | MOVWP++ | W=PMEM[PCH:FSR]<br>FSR++ | 1 | - | 2 |
| | MOVWP-- | FSR—<br>W = PMEM[PCH:FSR] | 1 | - | 2 |
| IMOVW | IMOVW | W = DMEM[FSR] | 1 | - | 1/2 |
| | IMOVW++ | W=DMEM[FSR]<br>FSR++ | 1 | - | 1/2 |
| | IMOVW-- | FSR—<br>W=DMEM[FSR] | 1 | - | 1/2 |
| | IMOVW +q[5] | W=DMEM[FSR+q] | 1 | - | 1/2 |
| | IMOVW –q | W=DMEM[FSR-q] | 1 | - | 1/2 |
| IMOVF | IMOVF | DMEM[FSR] = W | 1 | - | 1 |
| | IMOVF++ | DMEM[SFR] = W<br>SFR++ | 1 | - | 1 |
| | IMOVF-- | SFR—<br>DMEM[FSR] = W | 1 | - | 1 |
| | IMOVF +q | DMEM[FSR+q] = W | 1 | - | 1 |
| | IMOVF -q | DMEM[FSR-q] = W | 1 | - | 1 |
| 扩展流程控制指令 | | | | | |
| LOOP | LOOP k | IF(LCR!=0) { LCR--, PC = k} | 2 | - | 2 |
| BRSZ | BRSZ k | IF(Z==1) PC = k | 2 | - | 2 |

| | | | | | |
|---|---|---|---|---|---|
| BRCZ | BRCZ k | IF(Z==0) PC = k | 2 | - | 2 |
| BRSC | BRSC k | IF(C==1) PC = k | 2 | - | 2 |
| BRCC | BRCC k | IF(C==0) PC = k | 2 | - | 2 |

说明：

1. 指令集中的 F 为一个 6 位宽度的立即数，用于指定 IO/RAM 的地址；
2. 指令集中的 d 为一个 1 位宽度的立即数，当 d=0（默认）时，指令执行的结果回写到 W 工作寄存器，否则写入到 F 指定地址；
3. 指令集中的 b 为一个 3 位的立即数， 用于指定访问数据的位地址；
4. 指令集中的 k 为一个立即数，根据指令不同，k 的宽度分为 8/10 位不等，8 位宽度一般是数据，10 位为程序目标地址；
5. 指令集中的 q 为一个 4 位的立即数，指定间接寻址模式的地址偏移量

# 版本历史

| 版本 | 作者 | 日期 | 版本日志 |
|---|---|---|---|
| 1.0.0 | LGT | 2014/10/15 | The first edition |
| | | | |
| | | | |
| | | | |