

---

## XJIDE 使用说明 V6.0

### 仿真上板型号对应关系:

SL05: 仿真支持 XC8P8612、XC8P9611 (BUG: 使用 SL05 仿真上板时, 开启定时器单步仿真不可用否则会频繁进中断。全速运行仿真不受影响。)

SL06: 仿真支持 XC8P9521、XC8P9520

SL07: 仿真支持 XC8P8600

SL08: 仿真支持 XC8P8613

SL09: 仿真支持 XC8P8508

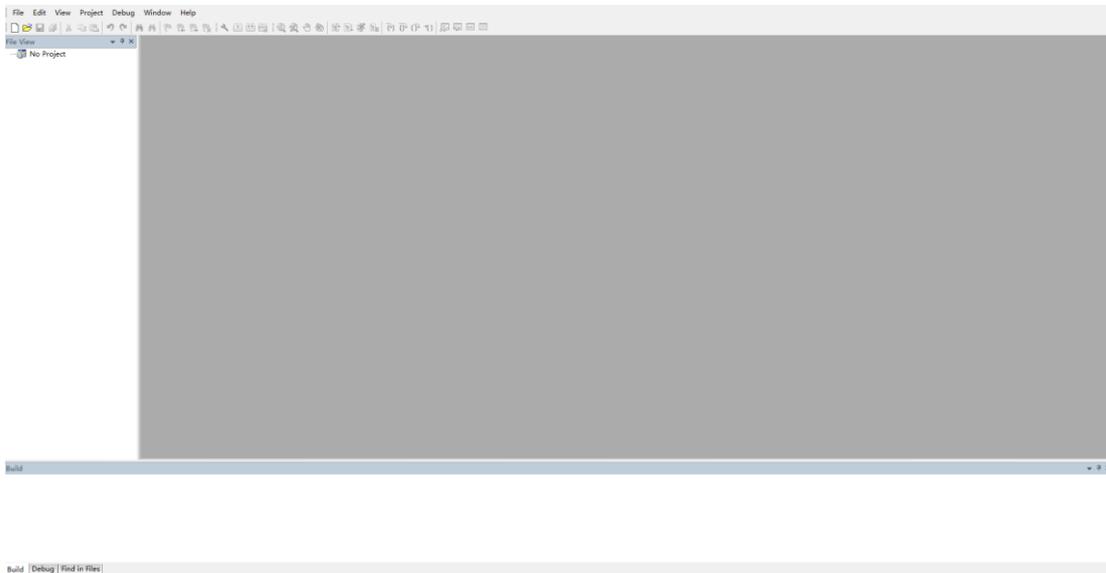
## 1.1 驱动安装:

使用仿真器前，需先安装 USB 驱动。找到  XJ-IDE\_V2\_0 ，双击  InstallDriver.exe 进行驱动安装，只需要安装一次，后续使用不必安装。

## 1.2 软件安装:

双击安装文件  XJIDESetup.msi 进行软件安装。

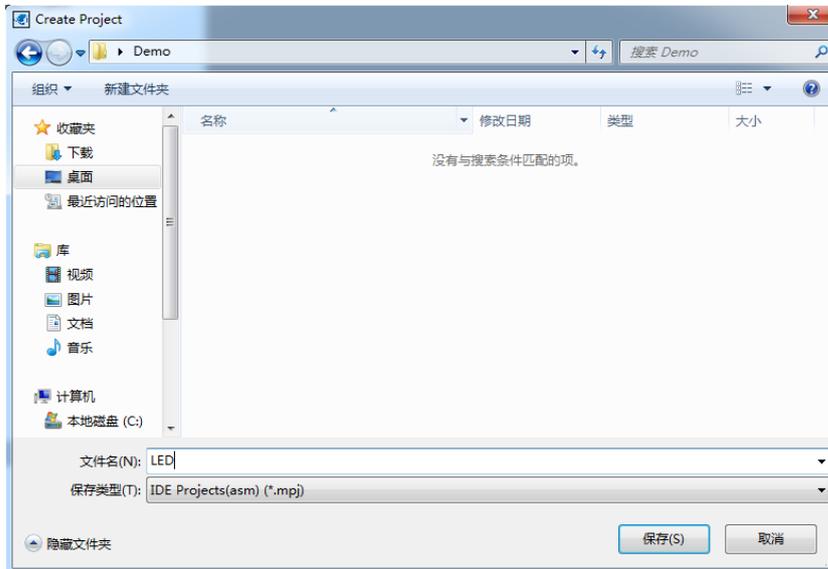
安装成功后，双击图标打开软件。



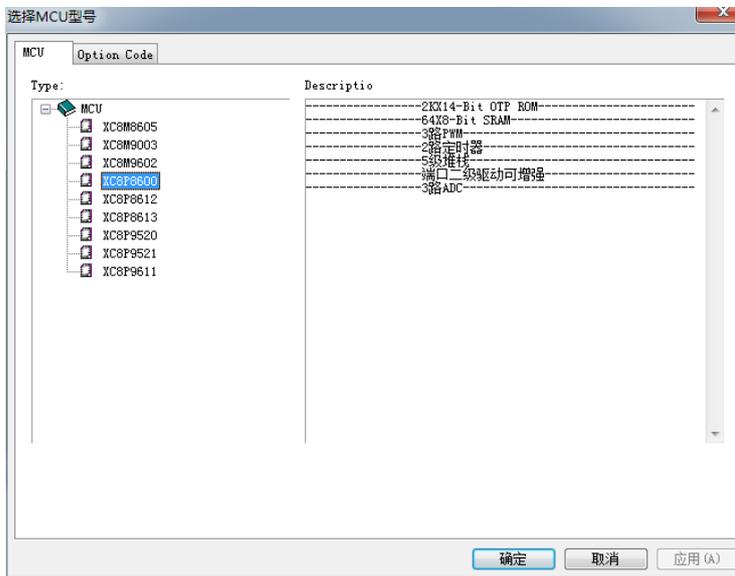
## 2.1 新建与编译工程:

### 2.1.1 新建工程:

工具栏选择 Project->New Project 新建工程，在弹出的对话框中选择工程路径与工程名后，点击保存。



选择 MCU 型号，及芯片 option 配置。

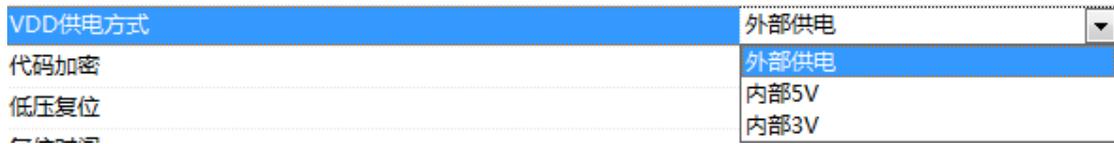


注：配置中 VDD 供电方式是指仿真芯片电源选择。

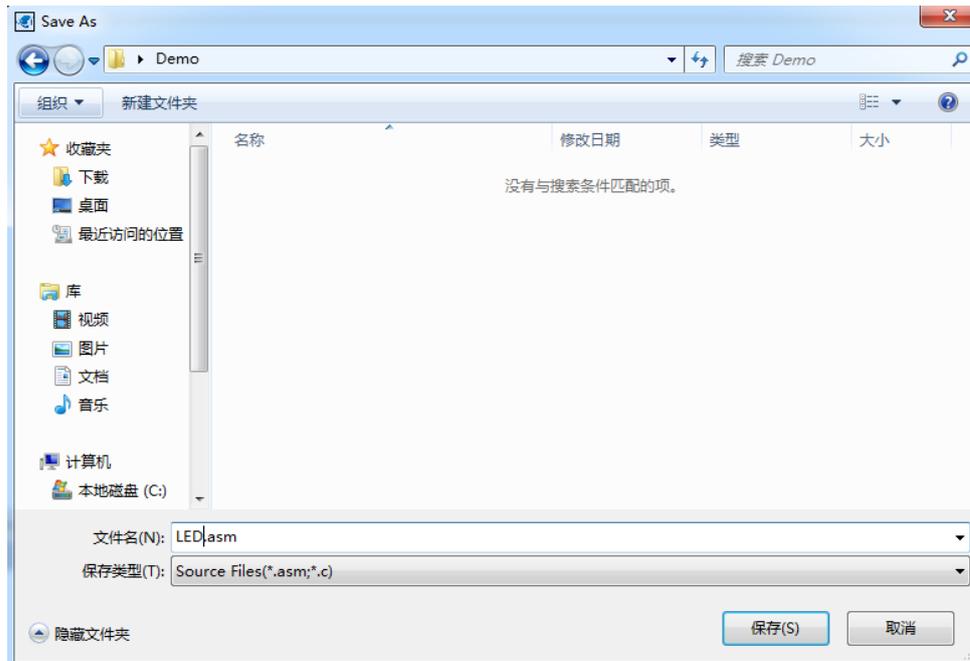
EXTV：仿真芯片外部供电。

5V：仿真芯片内部 5V 供电（由仿真器提供 5V VDD）。

3V：仿真芯片内部 3V 供电（由仿真器提供 3V VDD）。

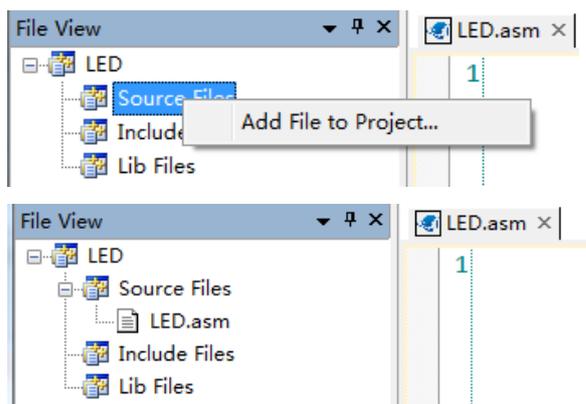


点击 File->New, 新建工程文档, 再点击保存。



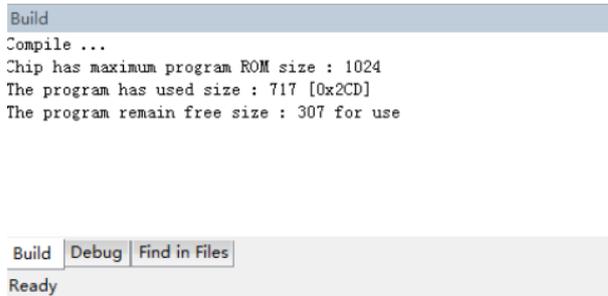
注：若为 .asm 程序文件，则保存文件名后加 .asm 后缀；若为 .ash 头文件，则保存文件后加 .ash 头文件。

右击工程，选择添加程序文件，添加后即可编写程序。

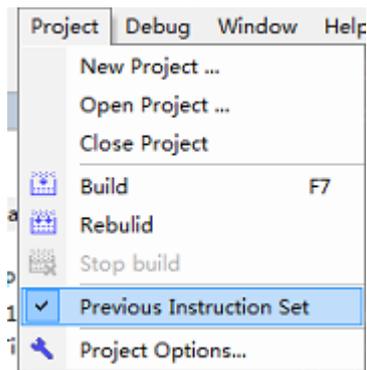


## 2.1.2 编译工程:

点击 ，编译程序，编译成功后如下图所示



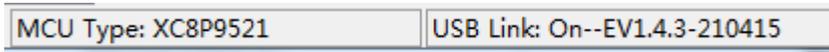
## 3.1 指令集选择:



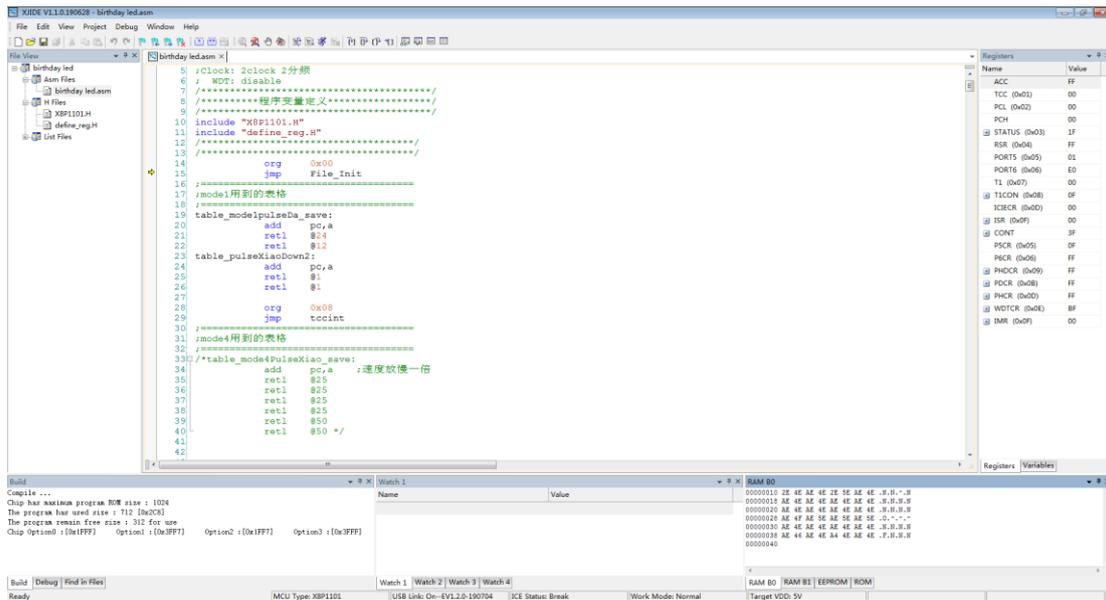
勾选 Previous Instruction Set，则选用兼容指令集，反之则选用原厂指令集。

## 4.1 仿真工程:

连上仿真器下位机后，仿真器上位机任务栏会显示 USB Link: ON--固件版本。



点击 Debug->Start Debugging, 进入仿真界面。



: 编译文件。



: 退出仿真模式->编译文件->进入仿真模式。



: 进入仿真模式。



: 退出仿真模式。



: 插入/移除断点。



: 移除所有断点。



: 软件复位。



: 运行，遇到断点停下。



: 全速运行，忽略断点。

: 软件停止。

: 单步运行。

: 跨越执行。

: 跳出当前子程序。

: 运行至光标处。

## 5.1 程序书写说明:

- 1 支持 include 伪指令, 可包含 .ash 头文件
- 2 支持 macro 伪指令
- 3 支持注释符号 “;”, “//”, “/\*”, “\*/”
- 4 支持 EQU 和 == 伪指令
- 5 支持 ORG 伪指令
- 6 支持 “+”, “-”, “\*”, “/”, “%”, “&”, “|”, “^”, “(”, “)” 混合运算

## C 编部分：

### C 编译器使用说明

#### 1.1 简介

XJ\_C\_IDE 为 XC8P&XC8M 系列 MCU IC 提供 C 编译支持，通过调用 C 语言编译程序 (Compiler), 再结合 ASM 组译器(Assembler)进一步组译及连接目标档来生成 .xbin 文件，烧写器可直接打开 .xbin 文件，烧录至 IC。

#### 1.2 系统需求

- Pentium 1.3GHz 或更高处理器，Win7 及以上操作系统。
- 2G 及以上的 SDRAM
- 2G 及以上的硬盘空间

#### 1.3 新建与编译工程

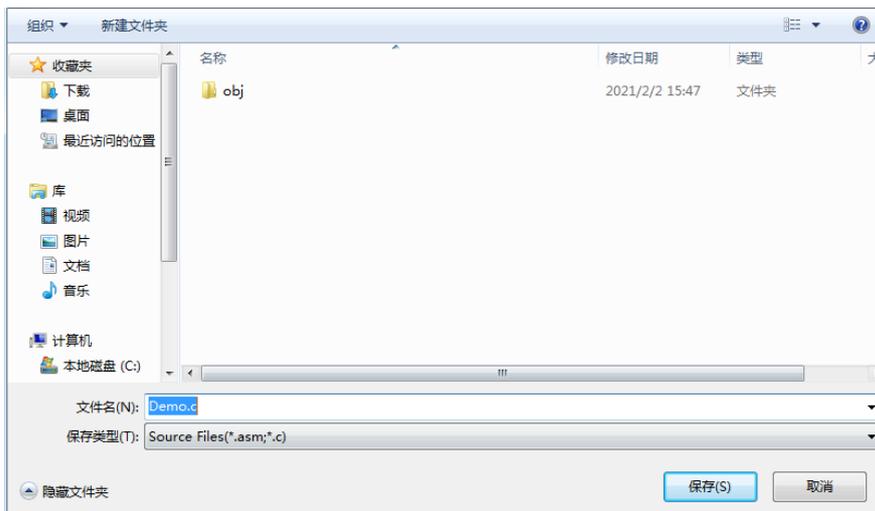
##### 1.3.1 软件安装

双击 IDESetup.msi 安装包进行软件安装，**安装目录中不可包含空格字符**。

##### 1.3.2 新建工程

工具栏选择 Project->New Project 新建工程，在弹出对话框中保存类型(T)选择 IDE Project(c) (\*.mpj)，并设置工程路径与工程名 (**工程路径不可包含空格字符**)，点击保存，然后选择芯片型号。

点击 File->New，新建工程文档，设置文件名并加 .c 后缀，再点击保存。



右击工程树目录点击 Add File to Project，对话框中文件类型选择 (\*.c)，添加对应 .c 程序。右击 Include Files 可添加 .h 头文件。



### 1.3.3 编译工程

点击  编译按钮, 编译成功后, 底部信息输出栏会显示程序空间使用率、RAM 使用率、校验、配置字及错误警告等信息。

```
Build
Compile & Linking ...
ROM Size Total :0x0800, Used :0x0076 [5%], Free :0x078A [95%]
RAM Size Total :0x0050, Used :0x000F [18%], Free :0x0041 [82%]
The file checksum is :0x1C7B, Option Code :0x1FFF 0x3FF7 0x1FFF 0x3FFF
"ALLREG" - 0 Error(s) - 0 Warning(s) -0 Message(s)
```

Build | Debug | Find in Files

### 2.1 C 语法

XJ\_C\_IDE 是 Mini 版的 C 编译器, 所以仅支持一些非常常规的 C 语法, 不支持一些较为复杂的语法包括但不限于指针、递归等操作。

### 2.2 注释

支持两种格式的注释符, 以双斜线起头的单行注释, 以及/\*开始, \*/结束的多行注释。  
举例:

```
//single line comment
/*
Multi line comment
*/
```

### 2.3 数据长度

以下表格为编译器所支持的基本数据类型及有效数据范围

型态	长度	范围
char	1 byte	0~255
unsigned char	1 byte	-128~127
short	2 byte	-32768~32767
unsigned short	2 byte	0~65535
int	2 byte	-32768~32767
unsigned int	2 byte	0~65535
long	4 byte	-2147483648~2147483647
unsigned long	4 byte	0~4294967295

## 2.4 嵌汇编语言

写 C 程序时可以嵌入汇编语言，使用 `__asm__(" ")` 可插入单条语句，使用 `__asm` 开头，`__endasm;` 结束可插入多条语句。

举例：

```
单条语句：
__asm__("NOP");

多条语句：
__asm
NOP
NOP
__endasm;
```

## 2.5 系统寄存器位定义方法

举例

```
typedef union {
    struct {
        unsigned char PORT6_0:1;
        unsigned char PORT6_1:1;
        unsigned char PORT6_2:1;
        unsigned char PORT6_3:1;
        unsigned char PORT6_4:1;
        unsigned char PORT6_5:1;
        unsigned char PORT6_6:1;
        unsigned char PORT6_7:1;
    };
} __PORT6bits_t;
extern volatile __PORT6bits_t __at(PORT6_ADDR) PORT6bits;

#define PORT6_0    PORT6bits.PORT6_0    /* bit 0 */
#define PORT6_1    PORT6bits.PORT6_1    /* bit 1 */
#define PORT6_2    PORT6bits.PORT6_2    /* bit 2 */
#define PORT6_3    PORT6bits.PORT6_3    /* bit 3 */
#define PORT6_4    PORT6bits.PORT6_4    /* bit 4 */
#define PORT6_5    PORT6bits.PORT6_5    /* bit 5 */
#define PORT6_6    PORT6bits.PORT6_6    /* bit 6 */
#define PORT6_7    PORT6bits.PORT6_7    /* bit 7 */
```

## 2.6 RAM 位定义方法

举例

```
typedef union{
    struct {
        unsigned b0      : 1;
        unsigned b1      : 1;
        unsigned b2      : 1;
        unsigned b3      : 1;
        unsigned b4      : 1;
        unsigned b5      : 1;
        unsigned b6      : 1;
        unsigned b7      : 1;
    };
}ob8;
ob8 BIT_FLAG;
#define BIT_1MS BIT_FLAG.b0
#define BIT_OK BIT_FLAG.b1
#define BIT_ERR BIT_FLAG.b2
```

## 2.7 强制指定内存地址

举例

```
__at(0x10) unsigned char IntVecIdx;
```

如果编译报错 multiple sections using address XX, 则可能是地址被其他变量已占用, 建议更换地址。指定地址定义变量时, 地址不能选在系统寄存器区。若需要重新定义系统寄存器, 请使用#define 预处理指令, 不可直接改写 IDE 安装目录 include 文件中的.h 系统头文件。

举例

```
#define KEY PORT5_0
```

## 3.1 8M 系列 IC 中断函数定义方法

C 语言中 `__interrupt {}` 代表硬件中断服务程序。编译程序时会将此段程序安排在指定地址。中断程序中无需添加现场保护程序，芯片硬件自带现场保护。中断服务模块程序参考例中写法，只需在指定地方添加执行代码即可。

举例：

```

void int_isr(void) __interrupt
{
    __asm__("org 0x03");__asm__("mov a,PCL");__asm__("jmp 0x33"); //TC0 中断
    __asm__("org 0x06");__asm__("mov a,PCL");__asm__("jmp 0x33"); //PORT 中断
    __asm__("org 0x09");__asm__("mov a,PCL");__asm__("jmp 0x33"); //INT0 中断
    __asm__("org 0x0c");__asm__("mov a,PCL");__asm__("jmp 0x33"); //INT1 中断
    __asm__("org 0x0f");__asm__("mov a,PCL");__asm__("jmp 0x33"); //ADC 中断
    __asm__("org 0x12");__asm__("mov a,PCL");__asm__("jmp 0x33"); //CMP0 中断
    __asm__("org 0x15");__asm__("mov a,PCL");__asm__("jmp 0x33"); //CMP1 中断
    __asm__("org 0x18");__asm__("mov a,PCL");__asm__("jmp 0x33"); //TC1 中断
    __asm__("org 0x1b");__asm__("mov a,PCL");__asm__("jmp 0x33"); //PRD0 中断
    __asm__("org 0x1e");__asm__("mov a,PCL");__asm__("jmp 0x33"); //PRD1 中断
    __asm__("org 0x21");__asm__("mov a,PCL");__asm__("jmp 0x33"); //PRD2 中断
    __asm__("org 0x24");__asm__("mov a,PCL");__asm__("jmp 0x33"); //PRD3 中断
    __asm__("org 0x27");__asm__("mov a,PCL");__asm__("jmp 0x33"); //DT0 中断
    __asm__("org 0x2a");__asm__("mov a,PCL");__asm__("jmp 0x33"); //DT1 中断
    __asm__("org 0x2d");__asm__("mov a,PCL");__asm__("jmp 0x33"); //DT2 中断
    __asm__("org 0x30");__asm__("mov a,PCL");__asm__("jmp 0x33"); //DT3 中断
    __asm__("org 0x33");__asm__("mov 0x10,a");
    switch(IntVectIdx)
    {
        case 0x04: //TC0 中断
        {
            ...
            break;
        }
        //case 0x07: //PORT 中断
        //case 0x0A: //INT0 中断
        //case 0x0D: //INT1 中断
        //case 0x10: //ADC 中断
        //case 0x13: //CMP0 中断
        //case 0x16: //CMP1 中断
        //case 0x19: //TC1 中断
        //case 0x1C: //PRD0 中断
        //case 0x1F: //PRD1 中断
        //case 0x22: //PRD2 中断
        //case 0x25: //PRD3 中断
        //case 0x28: //DT0 中断
        //case 0x2B: //DT1 中断
        //case 0x2E: //DT2 中断
        //case 0x31: //DT3 中断
    }
}

```

## 3.2 8M 系列 IC 新增特殊语法

地址	直接寻址	间接寻址			
0x00	通用RAM	通用RAM			
...					
...					
0x7F					
0x80	系统寄存器	通用RAM		芯片物理地址	C编映射地址
...			0x80	0x180	
...			...	...	
0xFF			0xFF	0x1FF	

8M 系列 IC 共有 256 个 RAM，地址 0x00~0x7F 的 RAM 可直接寻址操作，亦可间接寻址操作；在地址 0x80~0xFF 中，若要设置系统寄存器必须直接寻址操作，若要使用 RAM 必须间接寻址操作。为了 C 编译器有效区分地址 0x80~0xFF 的系统寄存器和通用 RAM，故将该地址 RAM 映射至 0x180~0x1FF，用户实际操作地址 0x180~0x1FF 时，编译器后台将进行间接寻址操作。

操作地址 0x180~0x1FF 的 RAM 时，需加 `__xdata` 前缀，且必须指定有效地址，使 C 编译器后台可识别并进行间接寻址操作。指定有效地址时，需注意变量类型，留好空间，防止地址重复定义报错。

举例

```
__xdata __at(0x180) unsigned char i;
__xdata __at(0x181) unsigned int j;
__xdata __at(0x183) unsigned long m;
__xdata __at(0x187) unsigned char n;
```

使用地址 0x180~0x1FF 的 RAM 时，请在以下格式内使用，若超出以下格式操作 RAM，可能出现译码问题，请联系软件开发人员。

算术运算	加法	<code>A = B + C;</code>
		<code>A++;</code>
		<code>++A;</code>
		<code>A = B++;</code>
		<code>A = ++B;</code>
		<code>A += B;</code>
	减法	<code>A = B - C;</code>
		<code>A--;</code>
		<code>--A;</code>
		<code>A = B--;</code>
		<code>A = --B;</code>
		<code>A -= B;</code>
	乘法	<code>A = B * C;</code> <code>A *= B;</code>
	除法	<code>A = B / C;</code> <code>A /= B;</code>
求余	<code>A = B % C;</code> <code>A %= B;</code>	
关系运算	大于	<code>if(A &gt; B);</code>
	小于	<code>if(A &lt; B);</code>
	双等于	<code>if(A == B);</code>
	大于等于	<code>if(A &gt;= B);</code>
	小于等于	<code>if(A &lt;= B);</code>
	不等于	<code>if(A != B);</code>
位操作运算	位与	<code>A = B &amp; C;</code>
	位或	<code>A = B   C;</code>
	位异或	<code>A = B ^ C;</code>
	位非	<code>A = ~B;</code>
	左移	<code>A = B &lt;&lt; 常数;</code>
	右移	<code>A = B &gt;&gt; 常数;</code>
逻辑运算	逻辑与	<code>if((表达式A) &amp;&amp; (表达式B));</code>
	逻辑或	<code>if((表达式A)    (表达式B));</code>
	逻辑非	<code>if(!表达式A);</code>
条件运算	三目运算符	<code>(表达式A) ? (表达式B) : (表达式C);</code>

## 4.1 8P 系列 IC 中断函数定义方法

C 语言中 `__interrupt {}` 代表硬件中断服务程序。编译程序时会将此段程序安排在指定地址。由于芯片硬件不自带现场保护，故参考示例中通过宏定义方式定义 PUSH & POP，用户只需要调用该宏，即可实现软件的现场保护。中断服务模块程序参考例中写法，只需在指定地方添加执行代码即可。

举例：

```
volatile __at(0x10) unsigned char A_BUFF;      //中断 ACC 保护 RAM
volatile __at(0x11) unsigned char R3_BUFF;    //中断 STATUS 保护 RAM
#define PUSH(A_REG,R3_REG)
__asm__("mov "#A_REG",a\n swap "#A_REG"\n swapa STATUS\n mov "#R3_REG",a") //中断入栈保护
#define POP(A_REG,R3_REG)
__asm__("swapa "#R3_REG"\n mov STATUS,a\n swapa "#A_REG") //中断 STATUS 保护 RAM

void int_isr(void) __interrupt
{
    __asm__("org 0x08");
    PUSH(_A_BUFF,_R3_BUFF); //中断入栈保护
    //=====中断程序=====//
    ...
    ...
    //=====中断程序=====//
    POP(_A_BUFF,_R3_BUFF); //中断出栈保护恢复
}
```

## 4.2 8P 系列 IC IO 页面寄存器操作方法

8P 系列 IC 特殊功能寄存器分为 R 页面寄存器和 IO 页面寄存器，R 页面寄存器为常规操作方式，IO 页面寄存器操作是通过宏定义实现，具体见示例。

举例：

```
volatile unsigned char Temp;
#define CONTW(VAL) __asm__("mov a,@#VAL\n ctw") //CTW = VAL: CONT 寄存器赋值
#define IOCP_W(REG,VAL) __asm__("mov a,@#VAL\n iw #REG") //REG = VAL: IO 寄存器赋值
#define IOCP_R(RAM,REG) __asm__("ir #REG\n mov #RAM,a") //RAM = REG: IO 寄存器读值
void main()
{
    //===== R 页面寄存器=====//
    PORT6= 0xAA; //PORT6 赋值 0xAA
    Temp = PORT6; //读 PORT6 到 Temp
    //===== R 页面寄存器=====//

    //===== IO 页面寄存器=====//
    CONTW(0x08); //CONT 赋值 0x08

    IOCP_W(ADCVS,0xAA); //ADCVS 赋值 0xAA
    IOCP_R(Temp, ADCVS); //读 ADCVS 到 Temp
    //===== IO 页面寄存器=====//
}
```

注：8P 1K 系列芯片为节省 RAM 空间，暂不支持对变量的乘法/除法/求余运算。

### 5.1 使用建议及注意:

#### 5.1.1 建议:

- 1、尽量使用无符号(unsigned)变量, 在部分运算中不判断正负号会提高译码效率。
- 2、表达式之中不要交叉使用常数与变量, 将常数集中可以提高优化效率。
- 3、不要将程序拆分为过多的.c 文件, 集中程序可减少 RAM 使用量, 提高优化效率。
- 4、为减少系统对 RAM 的占用, 程序中建议优先使用移位代替乘除法, 如果需要用到“\*”, “/”, “%” 运算, 应尽量使用 RAM 的单字节长度。
- 5、定义中断服务程序与正常流程共享的变量时, 建议在定义字符前加 volatile 关键字, 防止被异常优化。

#### 5.1.2 注意:

- 1、软件安装目录与程序工程目录中不可包含空格字符, 否则程序将编译报错。
- 2、不要修改系统内建的.h 头文件, 否则程序将编译报错。