```
int array1[CONSTANT1] __attribute__((__space__(xmemory), __aligned__(32)));
/* array with dsPIC30F attributes */

int array5[CONSTANT2]; /* simple array */
int variable1 __attribute__((__space__(xmemory)));

/* variable with attributes */
int variable3; /* simple variable */

int main ( void ) /* start of main application code */
{
/* Application code goes here */
}
void __attribute__((__interrupt__(__save__(variable1,variable2)))) _INT0Interrupt(void)
/* interrupt routine code */
{
/* Interrupt Service Routine code goes here */
}



/* Define constants here */
#define CONSTANT1 10
#define CONSTANT2 20

/* Define macros to simplify attribute declarations */
#define ModBuf_X(k) __attribute__((__space__(xmemory), __aligned__(k)))
#define ModBuf_Y(k) __attribute__((__space__(ymemory), __aligned__(k)))

/********** START OF GLOBAL DEFINITIONS **********/
/* Define arrays: array1[], array2[], etc. */
/* with attributes, as given below */
/* either using the entire attribute */
int array1[CONSTANT1] __attribute__((__space__(xmemory), __aligned__(32)));
int array2[CONSTANT1] __attribute__((__space__(ymemory), __aligned__(32)));
/* or using macros defined above */
int array3[CONSTANT1] ModBuf_X(32);
int array4[CONSTANT1] ModBuf_Y(32);
/* Define arrays without attributes */
int array5[CONSTANT2]; /* array5 is NOT an aligned buffer */
/* Define global variables with attributes */
int variable1 __attribute__((__space__(xmemory)));
int variable2 __attribute__((__space__(ymemory)));
```

```
/* Define global variables without attributes */
int variable3;
/************** END OF GLOBAL DEFINITIONS ***********/
/************** START OF MAIN FUNCTION ***************/
int main ( void )
{
/* Code goes here */
}
/****** START OF INTERRUPT SERVICE ROUTINES *********/
/* Replace the interrupt function names with the */
/* appropriate names depending on interrupt source. */
/* The names of various interrupt functions for */
/* each device are defined in the linker script. */
/* Interrupt Service Routine 1 */
/* No fast context save, and no variables stacked */
void __attribute__((__interrupt__)) _ADCInterrupt(void)
{
/* Interrupt Service Routine code goes here */
}
/* Interrupt Service Routine 2 */
/* Fast context save (using push.s and pop.s) */
void __attribute__((__interrupt__, __shadow__)) _T1Interrupt(void)
{
/* Interrupt Service Routine code goes here */
}
/* Interrupt Service Routine 3: INT0Interrupt */
/* Save and restore variables var1, var2, etc. */
void __attribute__((__interrupt__(__save__(variable1,variable2)))) _INT0Interrupt(void)
{
/* Interrupt Service Routine code goes here */
}
/********* END OF INTERRUPT SERVICE ROUTINES ********/
```

内建函数

__builtin_divsd

描述： 该函数计算 num / den 的商。 如果 den 为 0，则出现数学错误异常。函数参数是有符号的，函数的结果也是有符号的。命令行选项
-Wconversions 可用来检测意外的符号转换。
函数原型： int __builtin_divsd(const long num, const int den);

__builtin_divud

描述： 该函数计算 num / den 的商。 如果 den 为 0，则出现数学错误异常。函数参数是无符号的，函数的结果也是无符号的。命令行选项
-Wconversions 可用来检测意外的符号转换。
函数原型： unsigned int __builtin_divud(const unsigned
long num, const unsigned int den);

__builtin_mulss

描述： 该函数计算乘积 p0 x p1。函数参数是有符号整型，函数的结果是有符号长整型。命令行选项-Wconversions 可用来检测意外的符号转换。
函数原型： signed long __builtin_mulss(const signed int p0,
const signed int p1);

__builtin_mulsu

描述： 该函数计算乘积 p0 x p1 。函数参数是混合符号整型，函数的结果是有符号长整型。命令行选项-Wconversions 可用来检测意外的符号转换。该函数支持全部指令寻址模式，包括对操作数 p1 的立即寻址模式。
函数原型： signed long __builtin_mulsu(const signed int p0,
const unsigned int p1);

__builtin_mulus

描述： 该函数计算乘积 p0 x p1 。函数参数是混合符号整型，函数的结果是有符号长整型。命令行选项-Wconversions 可用来检测意外的符号转换。该函数支持全部指令寻址模式。
函数原型： signed long __builtin_mulus(const unsigned int p0,
const signed int p1);

__builtin_muluu

描述： 该函数计算乘积 p0 x p1 。函数参数是无符号整型，函数的结果是无符号长整型。命令行选项-Wconversions 可用来检测意外的符号转换。
该函数支持全部指令寻址模式，包括对操作数 p1 的立即寻址模式。
函数原型： unsigned long __builtin_muluu(const unsigned int p0,
const unsigned int p1);

__builtin_tblpage

描述： 该函数返回地址作为参数给定的对象的表页码。参数 p 必须是 EE 数据空间、PSV 或可执行存储空间中的对象的地址；否则，会产生错误消息并导致编译失败。可参阅《MPLAB® C30 C 编译器用户指南》中的 space 属性。
函数原型： unsigned int __builtin_tblpage(const void *p);

__builtin_tbloffset

描述： 该函数返回地址作为参数给定的对象的表页码偏移量。参数 p 必须是 EE 数据空间、PSV 或可执行存储空间中的对象的地址；否则，会产生错误消息并导致编译失败。可参阅《MPLAB® C30 C 编译器用户指南》中的 space 属性。
函数原型： unsigned int __builtin_tbloffset(const void *p);

__builtin_psvpage

描述： 该函数返回地址作为参数给定的对象的 PSV 页码。参数 p 必须是 EE 数据空间、PSV 或可执行存储空间中的对象的地址；否则，会产生错误消息并导致编译失败。可参阅《MPLAB® C30 C 编译器用户指南》中的 space 属性。
函数原型： unsigned int __builtin_psvpage(const void *p);

__builtin_psvoffset

描述： 该函数返回地址作为参数给定的对象的 PSV 页码偏移量。参数 p 必须是 EE 数据空间、PSV 或可执行存储空间中的对象的地址；否则，会产生错误消息并导致编译失败。可参阅《MPLAB® C30 C 编译器用户指南》中的 space 属性。
函数原型： unsigned int __builtin_psvoffset(const void *p);

__builtin_return_address

描述： 该函数返回当前函数或它的一个调用函数的返回地址。对于参数 level ，值 0 产生当前函数的返回地址，值 1 产生当前函数的调用函数的返回地址，等等。当 level 超过当前的堆栈深度时，返回 0。调试时，这个函数必须带有一个非 0 的参数。
函数原型： int __builtin_return_address (const int level);

行内汇编

有两种形式

简单形式：　asm(“assembly　　text”);

复杂形式：　asm(“template”:　“format”(varible),… :　“format”(variable),… :　”clobbers”);

int　my_data[256]　　_attribute_((space(xmemory)));

int　more_data[1024]　_attribute_((space(dma)));

attribute((space(area)));

其中 area 为：

data	一般数据空间
auto_psv	由编译管理的 **PSV**
psv	由用户管理的 **PSV**
dma	**DMA** 存储空间
ymemory	数据存储空间（**Y**）
xmemory	数据存储空间（**X**）
eedata	**EEDATA** 存储空间
prog	程序闪存

数据分配

可扩展数据类型变量：	**near** 数据
构造数据类型变量：	**near** 数据
常量：	自动 **PSV**
函数：	默认为小代码模型

其它属性

aligned(); recerse(); near; far;

address(); persistent; section;

noload

属性指明应该为变量分配空间，但不应为变量装入初值。这一属性对于设计在运行时将变量装入存储器（如从串行 **EEPROM**）的应用程序会有用。**int table1[50] __attribute__ ((noload)) = { 0 };**

persistent

属性指定在启动时变量不应被初始化或清零。具有 **persistent** 属性的变量可用于存储器件复位后仍保持有效的状态信息。

int last_mode __attribute__ ((persistent));

编译器支持 **3** 种 **PSV** 窗口使用模式

支持由用户管理的 **PSV**，将数据放到程序闪存：

attribute space(psv)

自动 **PSV** 模式 ,，将数据放到程序闪存,支持一个 **32K PSV** 页:

attribute space(auto_psv) 或者 **const**

由编译器管理的 **PSV**，将数据放到程序闪存，支持多个 **32K PSV** 页：

attribute space(psv) 或者 **space(prog)**

示例：**_psv_ int data[256] _attribtue_((space(psv)));**

psv 对象不能跨越 **PSV** 页

prog 对象可以跨越 **PSV** 页

中断

为中断服务程序编写代码

下面的原型声明了函数 isr0 为中断服务程序：

void __attribute__((__interrupt__)) isr0(void);

由原型可以看出，中断函数必须不带参数，没有返回值。如果需要的话，编译器将保护所有工作寄存器，以及 status 寄存器和重复计数寄存器。将其他变量指定为 interrupt 属性的参数，可以保护这些变量。例如，要使编译器自动保护和恢复变量 var1 和 var2，使用下面的原型：

void __attribute__((__interrupt__(__save__(var1,var2)))) isr0(void);

为请求编译器使用快速现场保护（使用 push.s 和 pop.s 指令），指定函数的 shadow 属性（参阅第 2.3.2 节“指定函数的属性”）。例如：

void __attribute__((__interrupt__, __shadow__)) isr0(void);

使用宏声明简单的中断服务程序

如果一个中断服务程序不需要 interrupt 属性的任何可选参数，则可使用简单的语法。在针对器件的头文件中定义了下面的宏：

#define _ISR __attribute__((interrupt))

#define _ISRFAST __attribute__((interrupt, shadow))

例如，声明 timer0 中断的中断服务程序：

#include <p30fxxxx.h>

void _ISR _INT0Interrupt(void);

用快速现场保护声明 SPI1 中断的中断服务程序：

#include <p30fxxxx.h>

void _ISRFAST _SPI1Interrupt(void);

禁止中断

```
DISICNT = 0x3FFF; /* disable interrupts */
/* ... protected C code ... */
DISICNT = 0x0000; /* enable interrupts */

Disable interrupts while the KEY sequence is written
      PUSH SR
      MOV #0x00E0,W0
      IOR SR

Re-enable interrupts
      POP SR

unsigned  int ipl=    SRbits.IPL;
SRbits.IPL = 7;
/*    Protected code here       */
SRbits.IPL=ipl;
```

Use volatile keyword for shared variables

```
int volatile gnTicks = 0;
void _attribute_((_interrrupt_))  _T2Interrupt(void){
gnTicks++;
IFS0bits.T2IF =0;
}
```

C 和汇编混合编程

```
/*
** file: ex1.c
*/
extern unsigned int asmVariable;
extern void asmFunction(void);
unsigned int cVariable;
void foo(void)
{
asmFunction();
asmVariable = 0x1234;
}
```

文件 ex2.s 定义了链接应用程序需要使用的 asmFunction 和 asmVariable。汇编文件还说明了如何调用 C 函数 foo，以及如何访问 C 定义的变量 cVariable。

```
;
; file: ex2.s
;
.text
.global _asmFunction
_asmFunction:
mov #0,w0
mov w0,_cVariable
return
.global _begin
_main:
call _foo
return
.bss
.global _asmVariable
.align 2
_asmVariable: .space 2
.end
```

Calling Asm from C

```
Extern void write_to_EEdata(int EEpace, int EEaddr, int value);
int _EEDATA(2)     config_bits[16];
void main(void){
      write_to_EEdata(0xFF,&config_bits[4],32);
}
.global _write_to_EEdata
_write_to_EEdata:
      Mov      w0         ,     TBLPAG       ;          EEPAGE
      Tblwtl   w2         ,     [w1]         ;          EEdata<-value
      Mov      #0x4004 ,     w0         ;
      Mov      w0         ,     NVMCON     ;
      Push     SR
      Mov      #0x00E0 ,     w0
      Ior      SR
      Mov      #0x55      ,     w0
      Mov      w0         ,     NVMKEY
      Mov      #0xAA      ,     w0
      Mov      w0         ,     NVMKEY
      Best     NVMCON,     #WR
      Pop      SR
      Return
```

Accessing C variables

```
Unsigned int divf(unsigned int num, unsigned int den){
      Register unsigned int quo asm("w0");
      Register unsigned int Wn asm("w2")=den;

      Asm ("repeat #17");
      Asm("drivf     %[Wm], %[Wn]" :
            "=r"    (quo):
            [Wm]    "r"    (num) ,    [Wn]    "r"    (Wn):
            "w1" )     ;
      Return(quo);
}
```