CHINA MASTERs

第十二届中国技术精英年会

C12H02 CF8

8位单片机的C语言优化技巧

课程目标

- 了解优化工具
- 学习一些有用的优化技术
- 通过提供的实验，练习所学知识

课程安排

- C语言简介
- 优化工具介绍
- 实验1
- 一些有用的优化技术
- 实验2
- 了解指针优化
- 实验3
- 总结

C语言简介

- 实现所需功能比较容易
- 较汇编语言，移植性强
- 可读性较好

进行优化的时间和原因

- 进行优化前，请切记：
  - 完善的代码好于任何优化编译器
  - 编译器生成的是您所编写的C语言代码的最优化机器码
  - 编译完成后，优化程序会做一些清洁工作

课程安排

- C语言简介
- 优化工具介绍
- 实验1
- 一些有用的优化技术
- 实验2
- 了解指针优化
- 实验3
- 总结

优化工具

- 程序剖析（Profiling）技术
- C编译器
  - 编译警告
  - 列表和映射文件
- 软件模拟器
- 在线调试器
- 示波器/逻辑分析器

优化工具
程序剖析技术

- 程序剖析用于衡量时序条件下的代码性能。
- 程序剖析技术包括：
  - 产生汇编代码
  - 翻转引脚电平
  - 使用定时器
  - 使用跑表定时器

程序剖析技术
生成汇编代码

- 对于较短的代码，计算每条指令的周期数

```
if(TIMER_LED == OFF){
0x81: MOVLB 0x2                    ;1 cycle
0x82: BTFSC PORTD, 0x1             ;1 or 2 cycles
0x83: GOTO 0x85                    ;2 cycles
```

指令执行速度= FOSC/4 = 32 MHz /4 = 8 x 10^6 个周期/秒

4个指令周期的时间（以秒计）= 周期数/时钟速度

= 4 / (8 x 10^6) ≈ 0.5 µs

程序剖析技术
翻转引脚电平

```
#define TIMER_LED_ON do{LATD1 = 1;} while(0)
#define TIMER_LED_OFF do{LATD1 = 0;} while(0)
……
TIMER_LED_ON

   if (dc_index == 0)
            duty_cycle[dc_index]=255;
    else if (dc_index == 1)
              duty_cycle[dc_index]=128;
        ……
              ……
                   else if (dc_index == 7)
                        duty_cycle[dc_index]=2;

 TIMER_LED_OFF
```

程序剖析技术
使用定时器

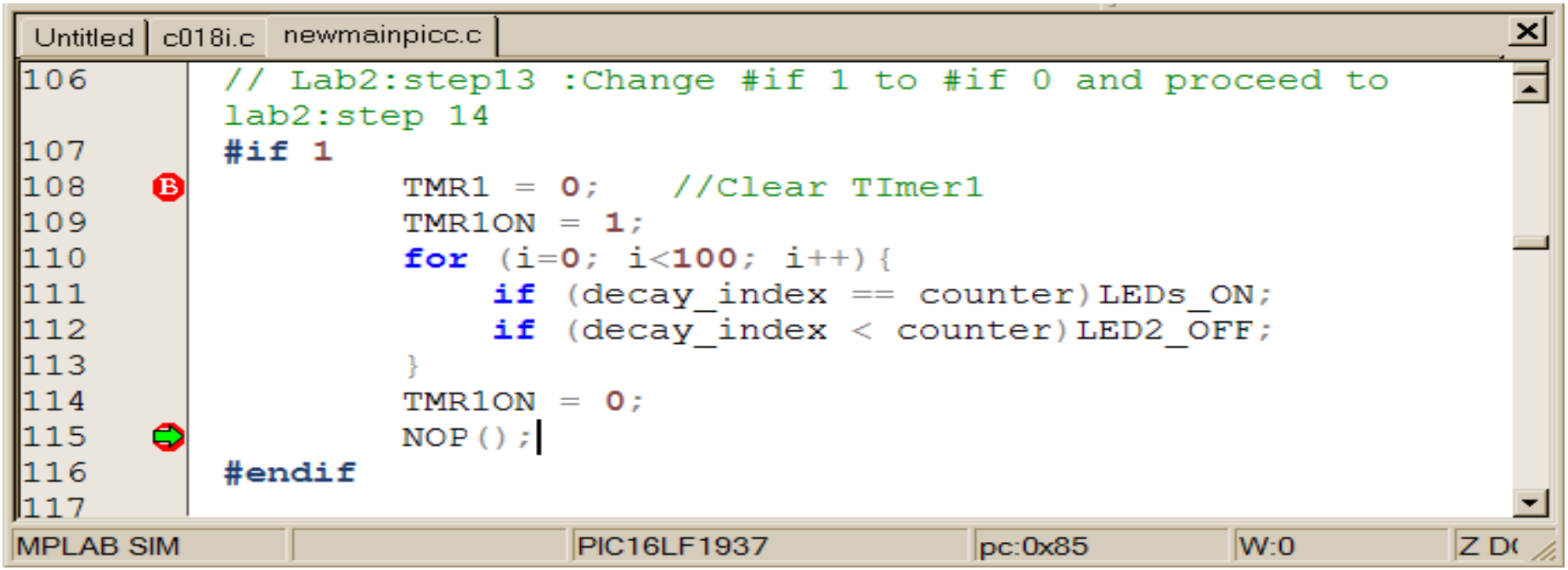
- 采用合适的参数设置定时器
  例如，`T1CON = 0x0F; (using instruction clock)`

```
TMR1 = 0;    //Clear Timer1
TMR1ON = 1; //Turn Timer1 ON
  for (i=0; i<100; i++){
     if (decay_index == counter)LEDs_ON;
     if (decay_index < counter)LED2_OFF;
    }
TMR1ON = 0; //Turn Timer1 OFF
NOP();
```

- 在关闭定时器的代码后设置一个断点，然后运行代码
- 采用**watch**（观察）窗口来读取值

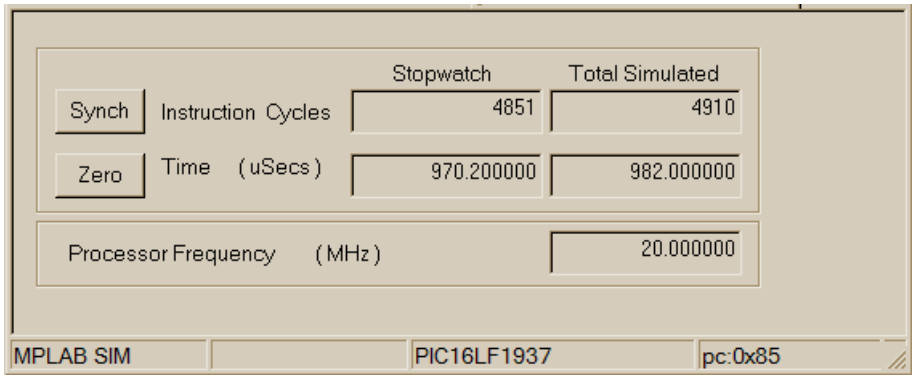
程序剖析技术
使用跑表定时器

- 在启动跑表的代码处添加一个断点
- 在停止跑表的代码处添加另一个断点



程序剖析技术
使用跑表定时器

- 要使用跑表定时器：
  - *Window>Debugging>Stopwatch*



- 再次运行程序以获取跑表计时结果

优化工具
列表和映射文件

- 列表文件
  - 由汇编器产生
  - 源代码和汇编代码之间的映射
  - C函数信息
  - 调用图形和调用图形表
  - 关键路径
  - 符号表

- 映射文件
  - 由链接器产生
  - 链接器选项
  - Psects
  - 未使用的类存储单元
  - 未使用的地址范围
  - 符号表

映射文件和优化时机

- 您可以找到函数和数据在存储器中的具体存放位置
- 您可以查看链接器选项进行修改

优化工具
映射文件链接器选项

链接器命令行：

```
-W9 --edf=c:\Program Files\HI-TECH Software\PICC\9.81\dat\en_msgs.txt -cs \
-h+dist/default/production\lab_2.X.production.sym -z -Q16F1937 -ol.obj \
-Mdist/default/production/lab_2.X.production.map -E1 -ACONST=00h-0FFhx32 \
-ACODE=00h-07FFhx4 -ASTRCODE=00h-01FFFh -AENTRY=00h-0FFhx32 \
-ASTRING=00h-0FFhx32 -ABANK0=020h-06Fh -ABANK1=0A0h-0EFh \
-ABANK2=0120h-016Fh -ABANK3=01A0h-01EFh -ABANK4=0220h-026Fh \
-ABANK5=02A0h-02EFh -ABANK6=0320h-032Fh -ABIGRAM=02000h-021EFh \
-ARAM=020h-06Fh,0A0h-0EFh,0120h-016Fh,01A0h-01EFh,0220h-026Fh,02A0h-
  02EFh,0320h-032Fh \
-AABS1=020h-07Fh,0A0h-0EFh,0120h-016Fh,01A0h-01EFh,0220h-026Fh,02A0h-
  02EFh,0320h-032Fh \
-ACOMMON=070h-07Fh -ASFR0=00h-01Fh -ASFR1=080h-09Fh -ASFR2=0100h-011Fh \
```

有关链接器选项的更多信息，请参见用户手册

中国技术精英年会

优化工具
映射文件：类和psects

CLASS	Name CONST swtext1	Link 43	Load 43	Length 10	Space 0
CLASS	CODE				
	end_init	71	71	3	0
	intentry	4	4	6D	0
	reset_vec	0	0	2	0
	cinit	7D	7D	171	0
CLASS	STRCODE				
CLASS	ENTRY				
CLASS	STRING				
	strings	74	74	9	0
CLASS	BANK0				
	cstackBANK0	20	20	11	1
	dataBANK0	31	31	9	1
	bssBANK0	3A	3A	1	1

优化工具
映射文件—没使用的存储区

未使用的地址范围

Name	Unused	Largest block	Delta
BANK0	00041-0006F	2F	1
BANK1	000A0-000EF	50	1
BANK2	00120-0016F	50	1
BANK3	001A0-001EF	50	1
BANK4	00220-0026F	50	1
BANK5	002A0-002EF	50	1
BANK6	00320-0032F	10	1
BIGRAM	02000-021EF	1F0	1
CODE	00002-00003	2	2
	002A9-01FFF	800	
COMMON	0007D-0007D	1	1
CONST	00002-00003	2	2
	002A9-01FFF	100	
EEDATA	0F000-0F0FF	100	2
ENTRY	00002-00003	2	2
	002A9-01FFF	100	
IDLOC	08000-08003	4	2
RAM	00041-0006F	2F	1
	000A0-000EF	50	
	00120-0016F	50	
	001A0-001EF	50	

列表文件和优化时机

- 通过研究汇编程序学习优化代码
- 避免或纠正堆栈溢出
- 关键路径
- 检查临时变量
- 可用的#pragma switch优化选项

优化工具
列表文件—汇编输出

```
376;newmainpicc.c: 123: for(i=0; i<5000; i++);

377  0077  0020          movlb 0  ; select bank0
378  0078  01BA           clrf    main@i
379  0079  01BB           clrf    main@i+1
380  007A  083B           movf    main@i+1,w
381  007B  3A80           xorlw 128
382  007C  00FF           movwf                 127
383  007D  3093           movlw                 147
384  007E  027F           subwf 127,w
385  007F  1D03           skipz
386  0080  2883          goto    u2955
387  0081  3088          movlw 136
388  0082  023A           subwf main@i,w
389  0083              u2955:
390  0083  1803          btfsc   3,0
391  0084  2895          goto    l2060
```

```
376;newmainpicc.c: 131: for(i=5000; i!=0; i--);

377  0077  3088          movlw   136
378  0078  0020          movlb   ; ;select bank0
379  0079  00BA          movwf   main@i
380  007A  3013          movlw   19
381  007B  00BB          movwf     main@i+1
382  007C  043A          iorwf    main@i,w
383  007D  1903          btfsc   3,2
384  007E  2886          goto    l2060
```

优化工具
列表文件函数调用关系图

```
407 ;; 函数调用关系图：
408
409 ;; _main (ROOT)
410 ;;   _initialize_hardware
412 ;;   _delayMS
411 ;;   _pattern_select
413 ;;     ___aldiv
414 ;;
415 ;; _interrupt_pwm (ROOT)
```

优化工具
列表文件关键路径

```
337 ;; Critical Paths under _main in COMMON
338 ;;
339 ;;   _delayMS->___aldiv
340 ;;
341 ;; Critical Paths under _interrupt_pwm in COMMON
342 ;;
343 ;;   None.
344 ;;
345 ;; Critical Paths under _main in BANK0
346 ;;
347 ;;   _main->_delayMS
348 ;;   _delayMS->___aldiv
349 ;;
350 ;; Critical Paths under _interrupt_pwm in BANK0
351 ;;
352 ;;   None.
```

优化工具
列表文件：临时变量

- 由??_funcName表示
- 示例：*Unsigned int += signed char + signed constant*

```
237                                 ;skh.c: 14: ui += sc + 0x1000;
238
239                                 ;                       RP0=0 RP1=0 IRP=0
240   07D0   0872                         movf    _sc,w   ;volatile  ; take sc
241   07D1   00F4                         movwf   ??_main
242   07D2   01F5                         clrf    ??_main+1     ; sign extend it
243   07D3   1BF4                         btfsc   ??_main,7     ; "
244   07D4   03F5                         decf    ??_main+1,f   ; "
245   07D5   3E00                         addlw   0
246   07D6   00F6                         movwf   ??_main+2
247   07D7   0875                         movf    ??_main+1,w
248   07D8   1803                         skipnc
249   07D9   3E01                         addlw   1
250   07DA   3E10                         addlw   16
251   07DB   00F7                         movwf   ??_main+3
252   07DC   0876                         movf    ??_main+2,w
253   07DD   07F0                         addwf   _ui,f
254   07DE   1803                         skipnc
255   07DF   0AF1                         incf    _ui+1,f
256   07E0   0877                         movf    ??_main+3,w
257   07E1   07F1                         addwf   _ui+1,f
```

优化工具
列表文件：临时变量

- 由??_funcName表示
- 示例：*Unsigned int += Unsigned char + signed constant*

```
272                                  ;skh.c: 17: ui += uc + 0x1000;
273
274                                  ;                              RP0=0 RP1=0 IRP=0
275  07E9  0873                             movf    _uc,w    ;volatile
276  07EA  3E00                             addlw   0
277  07EB  00F4                             movwf   ??_main
278  07EC  3010                             movlw   16
279  07ED  1803                             skipnc
280  07EE  3011                             movlw   17
281  07EF  00F5                             movwf   ??_main+1
282  07F0  0874                             movf    ??_main,w
283  07F1  07F0                             addwf   _ui,f
284  07F2  1803                             skipnc
285  07F3  0AF1                             incf    _ui+1,f
286  07F4  0875                             movf    ??_main+1,w
287  07F5  07F1                             addwf   _ui+1,f
```

中国技术精英年会

优化工具
列表文件：#pragma switch选项

- 语法：

#pragma switch rangetable

```
698                             ; Switch size 1,requested type "rangetable"
699                             ; Number of cases is 8, Range of values is
                                            0 to 7
700                             ; switch strategies available:
701                             ; Name          Instructions Cycles
702                             ; direct_byte           22     6 (fixed)
703                             ; simple_byte           25    13 (average)
704                             ; jumptable            260     6 (fixed)
705                             ; rangetable            12     4 (fixed)
706                             ; spacedrange           21     6 (fixed)
707                             ; locatedrange           8     3 (fixed)
708                             ;       Chosen strategy is rangetable
```

优化工具
符号表

- 全局C符号 *"_name" 例如 _GIE*
- 局部C符号 *"funcname@name" 例如main@cycles*
- 返回值：*?_funcname 例如?_delayMS*
- 特殊的自动变量和参数表示为如下形式：

??_funcName 例如?? _pattern_select

映射文件

```
??_interrupt_pwm    cstackCOMMON 070
??_main             cstackCOMMON 07C
??_patterns_select  cstackCOMMON 072
?___aldiv           cstackCOMMON 072
?___awdiv           cstackCOMMON 072
?_delayMS           cstackCOMMON 07A
_GIE                (abs)        05F
_LATD               (abs)        10F
_LATE               (abs)        110
```

列表文件

```
_end_of_pattern_select 0123
??_delayMS             0022
?_pattern_select       0070
pbssBANK0              0037
_pintentry             0004
_interrupt_pwm         0004
main@cycles            002B
ptext138               00B2
?_delayMS              0020
pswtext1               0043
aldiv@counter          0078
main@cycles            002B
```

课程安排

- C语言简介
- 优化工具介绍
- **实验1**
- 一些有用的优化技术
- 实验2
- 了解指针优化
- 实验3
- 总结

实验1

实验1目标

- 演示采用列表和映射文件进行的优化
  - 找到并运行MPLABX
  - 连接硬件
  - 编译程序，然后看见灯灭掉和闪烁
  - 打开列表文件，并了解要进行优化应该查看列表文件的什么位置
  - 打开映射文件并熟悉其内容

实验1总结
始终保持优化理念！

- 汇编输出
- 关键路径
- **#pragma switch**
- 临时变量
- **Psects**、类和未用的地址范围
- 符号表

课程安排

- C语言简介
- 优化工具介绍
- 实验1
- 一些有用的优化技术
- 实验2
- 了解指针优化
- 实验3
- 总结

优化技术
数据类型

- 尽可能使用最小数据类型
  示例： 如果计数器不超过255,便使用*char*而不是*int*来定义

- 定义一个char类型的常数
  - *const char time_out = 1;*
- #define
  - *#define time_out 1*
- 枚举
  - *enum consts{time_out =1, delay = 50};*

好

更好

最好!

优化技术
数据类型

- 避免比较有符号和无符号数
- 应注意所用数据类型的长度
  - 指针
  - 整型

优化技术
Hi-Tech C中的C99扩展

- `#include <stdint.h>`
- Unsigned char : `uint8_t`
- Signed char: `int8_t`
- Unsigned int : `uint16_t`
- Signed int : `int16_t`
- Unsigned long : `uint32_t`
- Signed long : `int32_t`

不用这些

而用这些！

优化技术
比较:有符号和无符号数

示例：*Unsigned int += signed char + signed constant*

```
237                 ;skh.c: 14: ui += sc + 0x1000;
238
239                     ; RP0=0 RP1=0 IRP=0
240  07D0  0872        movf    _sc,w   ;volatile
241  07D1  00F4        movwf   ??_main
242  07D2  01F5        clrf    ??_main+1
243  07D3  1BF4        btfsc   ??_main,7     ; "
244  07D4  03F5        decf    ??_main+1,f   ; "
245  07D5  3E00        addlw   0
246  07D6  00F6        movwf   ??_main+2
247  07D7  0875        movf    ??_main+1,w
248  07D8  1803        skipnc
249  07D9  3E01        addlw   1
250  07DA  3E10        addlw   16
251  07DB  00F7        movwf   ??_main+3
252  07DC  0876        movf    ??_main+2,w
253  07DD  07F0        addwf   _ui,f
254  07DE  1803        skipnc
255  07DF  0AF1        incf    _ui+1,f
256  07E0  0877        movf    ??_main+3,w
257  07E1  07F1        addwf   _ui+1,f
```

示例：*Unsigned int += Unsigned char + signed constant*

```
272     ;skh.c: 17: ui += uc + 0x1000;
273
274                     ; RP0=0 RP1=0 IRP=0
275  07E9  0873        movf    _uc,w   ;volatile
276  07EA  3E00        addlw   0
277  07EB  00F4        movwf   ??_main
278  07EC  3010        movlw   16
279  07ED  1803        skipnc
280  07EE  3011        movlw   17
281  07EF  00F5        movwf   ??_main+1
282  07F0  0874        movf    ??_main,w
283  07F1  07F0        addwf   _ui,f
284  07F2  1803        skipnc
285  07F3  0AF1        incf    _ui+1,f
286  07F4  0875        movf    ??_main+1,w
287  07F5  07F1        addwf   _ui+1,f
```

优化技术
定义结构体

- 8位编译器没有对齐或位填充功能
- 将位域放置在一起
- 将最常用的数据类型放在最顶部

```
typedef struct{
            unsigned a:1;
            uint8_t  c;
            unsigned b:1;
            uint16_t most_used;}example;
```

不是这样做

```
typedef struct{
            uint16_t most_used;
            unsigned a:1;
            unsigned b:1;
            uint8_t  c;} example;
```

而是这样做

优化技术
保持变量为局部变量

- C语言编程鼓励采用局部变量：
  - 存储区可能重叠
  - 执行后存储区可用
  - 在压缩代码时，优化程序首先处理局部变量

优化技术
交给编译器完成

- **sizeof(a)**
  - a的字节数
- **sizeof(a) / sizeof(*(a))**
  - 数组a中的元素数
- **预处理程序数学置换**

```
#define BRG9600 ((_XTAL_FREQ / (9600UL * 16UL))+1)
```

优化技术
处理循环内代码

- 首先优化循环内的代码

```
if(SystemError)
{
 for (a=0;a<100;a++)
  {
     LED1_ON();
     LED2_OFF();
     ErrorData[a]++;
  }
}
```

不这样做

```
if(SystemError)
{
  LED1_ON();
  LED2_OFF();
  for(a=0;a<100,a++)ErrorData[a]++;
}
```

而是这样做

优化技术
针对循环进行优化

- `for(i=0;i<100;i++)`
- `for(i=100;i>0;i--)`
- `for(i=100;i!=0;i--)`

中国技术精英年会

针对循环进行优化
汇编比较

```
for(int8_t i=100; i>0; i--);
0x60: MOVLW 0x64
0x61: MOVLB 0x0
0x62: MOVWF i
0x63: DECF i, F
0x64: MOVF i, W
0x65: XORLW 0x80
0x66: ADDLW 0x7F
0x67: BTFSC STATUS,0x0
0x68: GOTO 0x63
```

```
for(int8_t i=0; i<100;i++);
0x60: MOVLB 0x0
0x61: CLRF i
0x62: INCF i, F
0x63: MOVF i, W
0x64: XORLW 0x80
0x65: ADDLW 0x1C
0x66: BTFSS STATUS,0x0
0x67: GOTO 0x62
```

针对循环进行优化
汇编比较

```
for(int8_t i=0; i<100; i++);
0x60: MOVLB 0x0
0x61: CLRF i

0x62: INCF i, F
0x63: MOVF i, W
0x64: XORLW 0x80
0x65: ADDLW 0x1C
0x66: BTFSS STATUS, 0x0
0x67: GOTO 0x62
```

```
for(int8_t i=100; i!=0; i--);
0x6B: MOVLW 0x64
0x6C: MOVLB 0x0
0x6D: MOVWF I

0x6E: DECFSZ i, F
0x6F: GOTO 0x6E
```

针对循环进行优化
汇编比较

```
for(uint8_t i=0; i<100; i++);

0x60: MOVLB 0x0
0x61: CLRF i
0x62: MOVLW 0x64
0x63: INCF i, F
0x64: SUBWF i, W
0x65: BTFSS STATUS, 0x0
0x66: GOTO 0x62
```

```
for(uint8_t i=100; i!=0; i--);

0x6A: MOVLW 0x64
0x6B: MOVLB 0x0
0x6C: MOVWF i
0x6D: DECFSZ i, F
0x6E: GOTO 0x6D
```

针对循环进行优化
汇编比较

```
for(uint8_t i=100; i>0; i--);

0x72: MOVLW 0x64
0x73: MOVLB 0x0
0x74: MOVWF i
0x75: DECFSZ i, F
0x76: GOTO 0x75
```

```
for(uint8_t i=100; i!=0; i--);

0x6A: MOVLW 0x64
0x6B: MOVLB 0x0
0x6C: MOVWF i
0x6D: DECFSZ i, F
0x6E: GOTO 0x6D
```

优化技术
Switch与if else的对比

switch

- 采用跳转表
- 所有语句的执行时间相同
- 采用的case值为编译时间常量

if else

- 顺序语句
- 执行时间随满足的条件而变
- 不是所有的if else语句都能被写成 switch语句

优化技术
#pragma switch

- 用于switch语句
- 可用策略：

```
#pragma switch direct_byte
#pragma switch simple_byte
#pragma switch jumptable
#pragma switch rangetable
#pragma switch spacedrange
#pragma switch locaterange
```

优化工具
列表文件：#pragma switch选项

- 语法：

#pragma switch rangetable

```
698                             ; Switch size 1,requested type "rangetable"
699                             ; Number of cases is 8, Range of values is
                                            0 to 7
700                             ; switch strategies available:
701                             ; Name          Instructions Cycles
702                             ; direct_byte           22     6 (fixed)
703                             ; simple_byte           25    13 (average)
704                             ; jumptable            260     6 (fixed)
705                             ; rangetable            12     4 (fixed)
706                             ; spacedrange           21     6 (fixed)
707                             ; locatedrange           8     3 (fixed)
708                             ;       Chosen strategy is rangetable
```

优化技术

#pragma switch及对应的汇编程序

不采用 #pragma locatedrange

```
switch(dc_index){
0x112: MOVF dc_index, W
0x113: MOVWF FSR0L
0x114: MOVLW 0x8
0x115: SUBWF FSR0L, W
0x116: BTFSC STATUS, 0x0
0x117: GOTO 0x11C
0x118: MOVLP 0x1
0x119: LSLF FSR0L, W
0x11A: ADDLW 0xBE
0x11B: MOVWF PCL
0x1BE: MOVLP 0x0
0x1BF: GOTO 0xE8
0x1C0: MOVLP 0x0
0x1C1: GOTO 0xED
0x1C2: MOVLP 0x0
0x1C3: GOTO 0xF2
0x1C4: MOVLP 0x0
0x1C5: GOTO 0xF7
0x1C6: MOVLP 0x0
0x1C7: GOTO 0xFC
0x1C8: MOVLP 0x1
0x1C9: GOTO 0x101
0x1CA: MOVLP 0x1
0x1CB: GOTO 0x106
0x1CC: MOVLP 0x1
0x1CD: GOTO 0x10B
```

采用 #pragma locatedrange

```
switch(dc_index){
0x112: MOVF dc_index, W
0x113: BRW
0x114: GOTO 0xE8
0x115: GOTO 0xED
0x116: GOTO 0xF2
0x117: GOTO 0xF7
0x118: GOTO 0xFC
0x119: GOTO 0x101
0x11A: GOTO 0x106
0x11B: GOTO 0x10B
```

课程安排

- C语言简介
- 优化工具介绍
- 实验1
- 一些有用的优化技术
- **实验2**
- 了解指针优化
- 实验3
- 总结

实验2

实验2目标

- 学习和使用一些基本程序剖析技术
- 学习和使用一些优化技术

实验2总结

- 选择最佳数据类型
- 减少循环内的迭代次数
- 循环逆置
- **If else与switch语句的对比**
- 采用**#pragma switch**的好处
- 使用**timer1**来计算执行您的代码需要的指令周期数

课程安排

- C语言简介
- 优化工具介绍
- 实验1
- 一些有用的优化技术
- 实验2
- 了解指针优化
- 实验3
- 总结

指针和数组

- 数组ID和指向该数组的指针具有相同的类型
- “数组语法”和“指针语法”相同

```
arrayID[idx]  =>  *(arrayID + idx)
```

- 总是使用“指针算术运算”来处理地址

指针的算术运算

- 索引值必须与所引用对象的长度相乘才能得到地址偏移量

```
*(arrayID + idx)
```

被计算为：

```
*(arrayID + idx * sizeof(*arrayID))
```

- 编译器将简化任何常数表达式

指针的算术运算

- 与非小值（**non trivial value**）相乘会导致代码膨胀
  - 访问大对象可能会调用乘法库程序
  - PIC18器件可以采用硬件乘法
- 采用循环来访问数组可以避免乘法

使用数组进行访问

- 每次迭代都会重新计算数组的偏移量（乘数）

```
unsigned char c;
struct DATA data[20];
for(c=0; c!=NUM_ELEM(data); c++)
    total += data[c].value;
```

通过指针进行访问

- 偏移量累加至指针所存储的地址值

```
struct DATA data[20];

struct DATA * ptr;

for(ptr=data; ptr!=&data[NUM_ELEM(data)]; ptr++)
    total += ptr->value;
```

- 记住在此循环代码外指针将不指向该数组

比较指针

- 比较*必须*在相同对象的地址间进行
- C语言允许地址为数组结尾后的下一个单元

```
... ptr!=&data[NUM_ELEM(data)] ...
```

使用指针时的注意事项

- 若器件仅有一个FSR寄存器请小心
  - 非增强型中档单片机和基本型单片机
- FSR保存要采用如下方式间接访问的地址：
  - 所有指针变量
  - 采用一个变量作为索引的数组

使用指针时的注意事项

- 避免在一个表达式中采用两个复引用，尤其在以下情形：
  - 采用基本型PIC®单片机，而所有目标不在同一存储区
  - 指针和目标在不同的存储区

```
for(c=SIZE; c!=0; c++)
     *dst++ = *scr++;
```

通用指针

- 通用指针用于保存指向不同目标的指针
- 可用作函数参数
- 在复引用之前对指针进行指派

```
void put_data(void *data, int size)
 {
  for( ; size!=0; size--)
    TXREG = *(char *)data++;
 }
```

访问字节

- 编译器通常可优化位移量为8的整数倍的移位操作
- 指针在访问多字节对象的一个字节时更为高效
- 示例：按照从MSB到LSB的顺序逐一发送对象的每个字节

访问字节

- 高效，但前提是对象为long类型并且不可移植

```
void sendByte(unsigned long l)
 {
  send(l>>24);   // MSB
  send(l>>16);
  send(l>>8);
  send(l);       // LSB
 }
```

访问字节

- 将long地址转换为char型并添加一个偏移量

```
void sendByte(unsigned long l)
 {
  unsigned char c;

  for(c = sizeof(l); c != 0; c--)
    send(*((unsigned char *)&l + c - 1));
 }
```

- 这种指针转换是合法的
  - 不要遗漏指针的目标修饰符

通过指针传递

- 将参数传递至函数和从函数返回值时，均会复制对象
  - 对于较大的对象，复制过程很慢并且会增加RAM的用量
- 考虑使用指向对象的指针

通过指针传递

- 比较：

```
struct DATA initStruct(struct DATA st){
    st.init = getCurrentPos();
    st.cnt = getCurrentTick();
    return st;
}
```

```
void initStruct(struct DATA * stp) {
    stp->init = getCurrentPos();
    stp->cnt = getCurrentTick();
    return;
}
```

课程安排

- C语言简介
- 优化工具介绍
- 实验1
- 一些有用的优化技术
- 实验2
- 了解指针优化
- **实验3**
- 总结

实验3

实验3目标

- 遍历数组时，循环配置对时序有影响
- 指针有助于提高速度
- 结构成员的位置会影响间接访问时序

实验3

- 比较lab 3.c第1部分中两个循环的时序
  - 为什么其中一个比另一个快呢？
- 比较第2部分中两个循环的时序
  - 为什么其中一个比另一个快呢？

实验3

- 测量第3部分中循环的时序
  - 为什么此循环比第2部分中的同一循环慢这么多呢？
- 在第3部分中编写采用指针的循环替代代码
  - 在所有示例中，循环执行完毕后，`total`的值为0x78?

实验3

- 复制您刚刚编写的循环，但对结构的第一个成员 (`dat1`) 求和
- 比较此循环和前一个对`dat4`求和的循环的时序

实验3总结

- 为循环选择一个合适的索引和停止条件
- 采用指针来避免执行乘法
- 合理选择结构中成员的顺序

课程安排

- C语言简介
- 优化工具介绍
- 实验1
- 一些有用的优化技术
- 实验2
- 了解指针优化
- 实验3
- 总结

结论
优化工具、程序剖析和优化技术

- 采用不同的程序剖析技术来优化代码的执行效率
- 采用映射文件来查找所有代码和数据在存储器中的确切位置
- 采用列表文件来查看代码的反汇编形式，并由此发现缩减代码的机会
- 采用优化技术来缩减代码的尺寸并提高执行效率

结论
指针

- 指针和数组是等效的
- 数组暗含乘法运算
- 可能的话采用指针
- 通过指针传递结构成员

- 务必小心！

更多资源

- **Hi-TECH C for PIC10/12/16 User's Guide：**

  http://ww1.microchip.com/downloads/en/DeviceDoc/manual_PICC_982.pdf

- 网上链接：

  http://en.wikipedia.org/wiki/C_preprocessor

  http://publications.gbdirect.co.uk/c_book

商标

Microchip的名称和徽标组合、Microchip徽标、dsPIC、KeeLoq、KeeLoq徽标、MPLAB、PIC、PICmicro、PICSTART、PIC32徽标、rfPIC和 UNI/O均为Microchip Technology Inc.在美国和其他国家或地区的注册商标。

FilterLab、Hampshire、HI-TECH C、Linear Active Thermistor、MXDEV、 MXLAB、SEEVAL和The Embedded Control Solutions Company 均为Microchip Technology Inc.在美国的注册商标。

Analog-for-the-Digital Age、Application Maestro、chipKIT、chipKIT徽标、CodeGuard、dsPICDEM、dsPICDEM.net、dsPICworks、dsSPEAK、ECAN、ECONOMONITOR、FanSense、HI-TIDE、In-Circuit Serial Programming、ICSP、Mindi、MiWi、MPASM、MPLAB Certified徽标、MPLIB、MPLINK、mTouch、Omniscient Code Generation、PICC、PICC-18、PICDEM、PICDEM.net、PICkit、PICtail、REAL ICE、rfLAB、Select Mode、Total Endurance、TSHARC、UniWinDriver、WiperLock和ZENA均为Microchip Technology Inc.在美国和其他国家或地区的商标。

SQTP是Microchip Technology Inc.在美国的服务标记。

在此提及的所有其他商标均为各持有公司所有。