MICROCHIP MASTERs

701 PIC

PICmicro® MCU – 介绍PIC16系列产品的汇编编程

目程

讲述 部分

- 讲述 90分钟
- 休息 20 分钟

实验部分1

- 2 个实验 60 分钟
- 休息 60 分钟

实验部分2

- 4个实验 2 小时
- 休息 20 分钟

讲述部分的内容

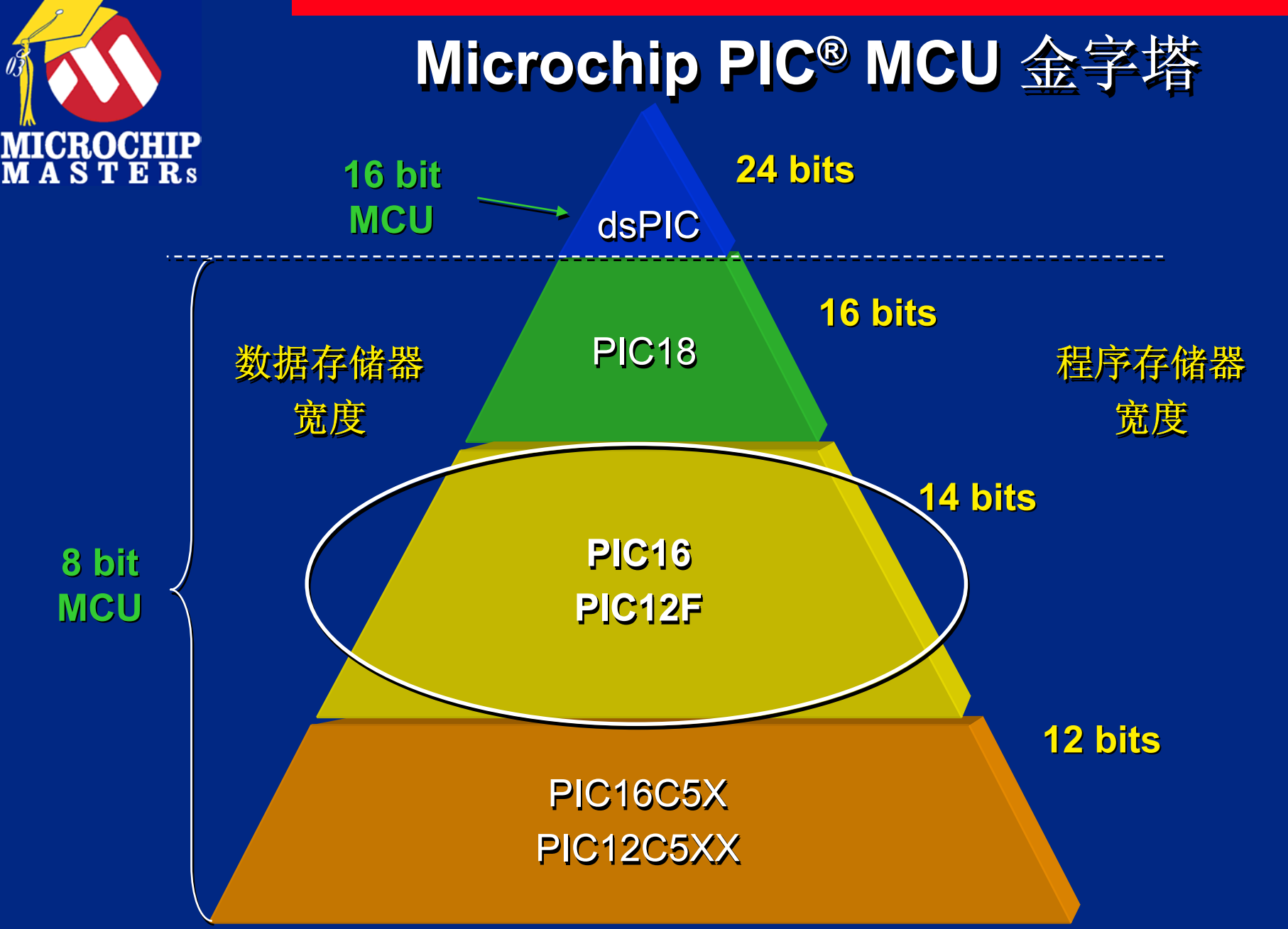
- 架构
- 存储器
- 指令集
- 开发软件 (MPLAB® IDE)

实验和演示部分的内容

- Labs
  - 在 MPLAB IDE创建项目
  - 调试和跳转控制
  - PWM
  - A/D
  - 中断
- 演示 (时间允许的话)
  - PIC16F877A实现多任务

架构

Microchip PIC® MCU 金字塔



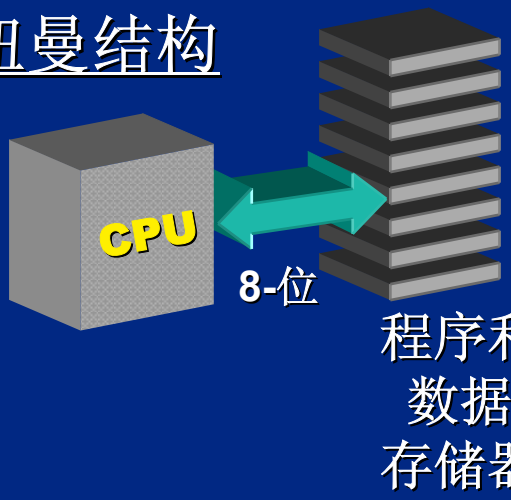
PICmicro Architecture
RISC-like Features

PIC单片机之所以有很高的性能是因为其具备如下特性:

- 内部为哈佛结构
- 寄存器文档
- 大多数单指令周期
- 指令流水线操作
- 长字型指令
- 指令数很少
- 指令实现的功能基本不重复

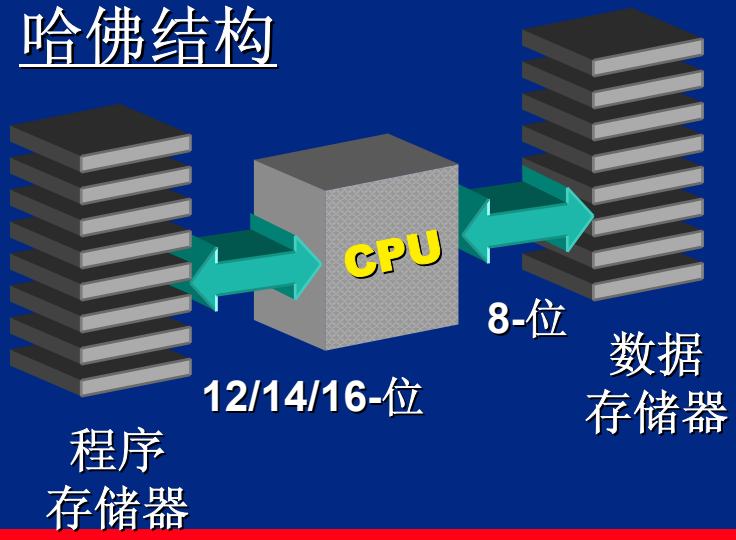
PIC单片机架构
哈佛结构

冯-纽曼结构



- 从同一存储器空间取指令和取操作数据.
  - 限制了数据流量

哈佛结构



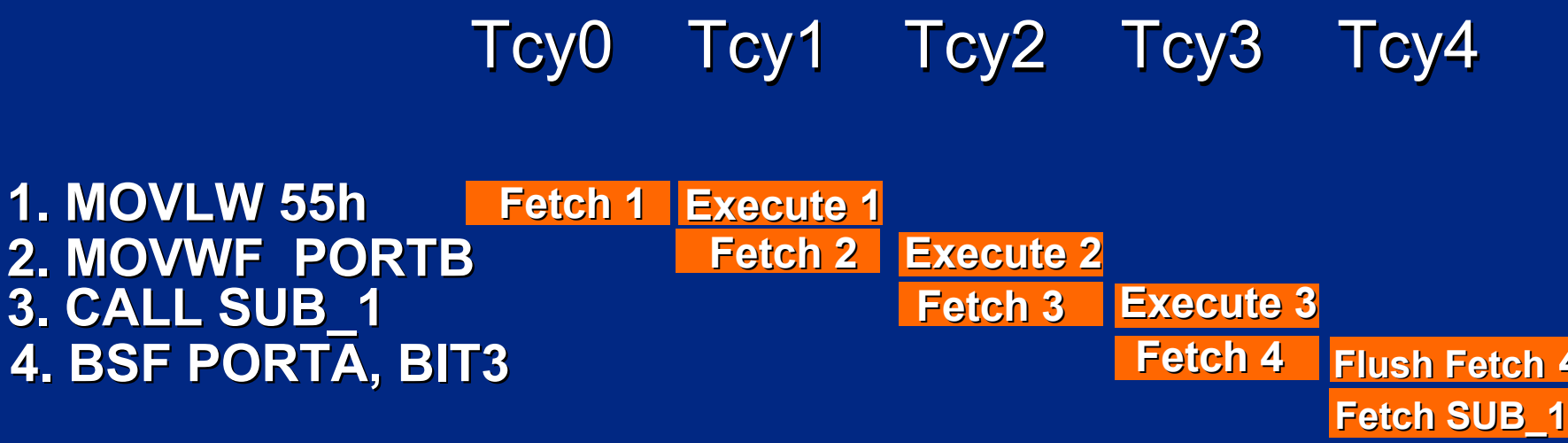
从两个独立的存储空间分别取指令和存取操作数.

数据流量增加

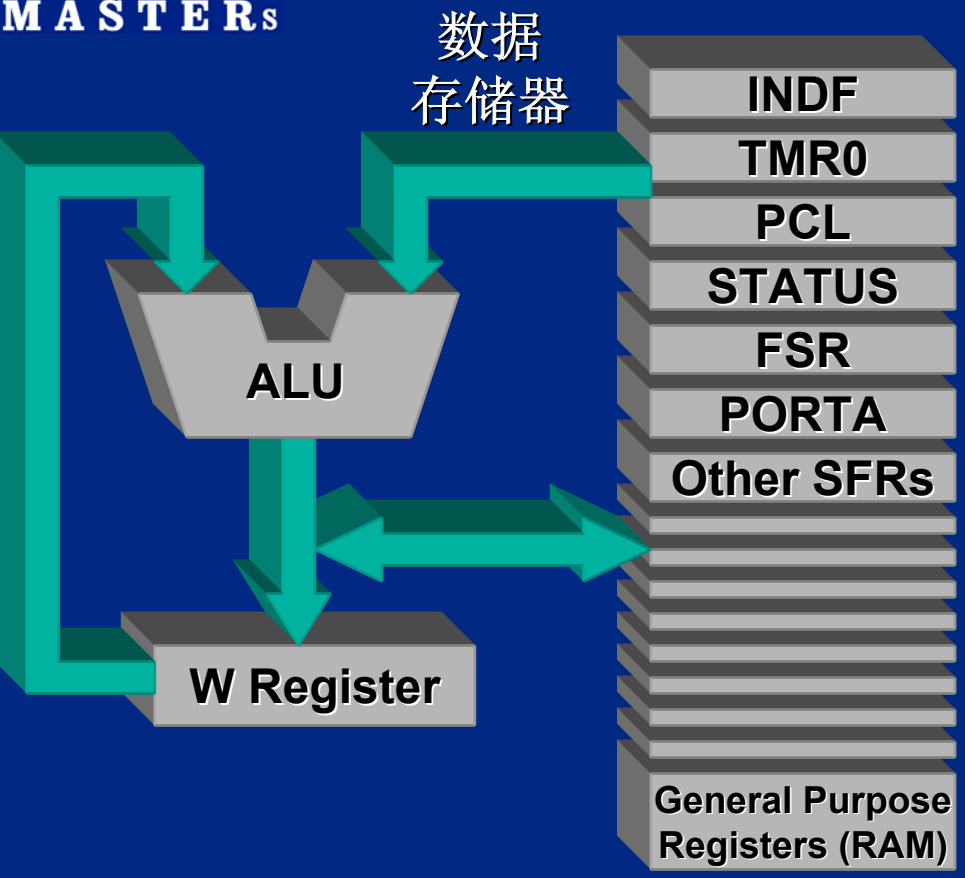
针对程序区和数据区可以设计不同的数据线宽度

PIC单片机架构
指令流水线

- 大部分单片机, 其取指和执行过程是顺序进行的.
- 指令流水线的引入允许取指和执行可以同步进行.
- 使得指令可以在一个指令周期内执行.
- 程序分支例外 (如 GOTO, CALL 或直接修改PC), 这需两个指令周期.



PIC单片机架构
寄存器文档概念



- RAM被看作是一组通用的寄存器.
- 周边模块(I/O)也作为寄存器.
- 所有的指令操作都可针对所有的寄存器.
- 长字指令使得在指令中直接寻址寄存器.

14-位字长的指令范例:

操作码 <7>	直接数据地址 <7>

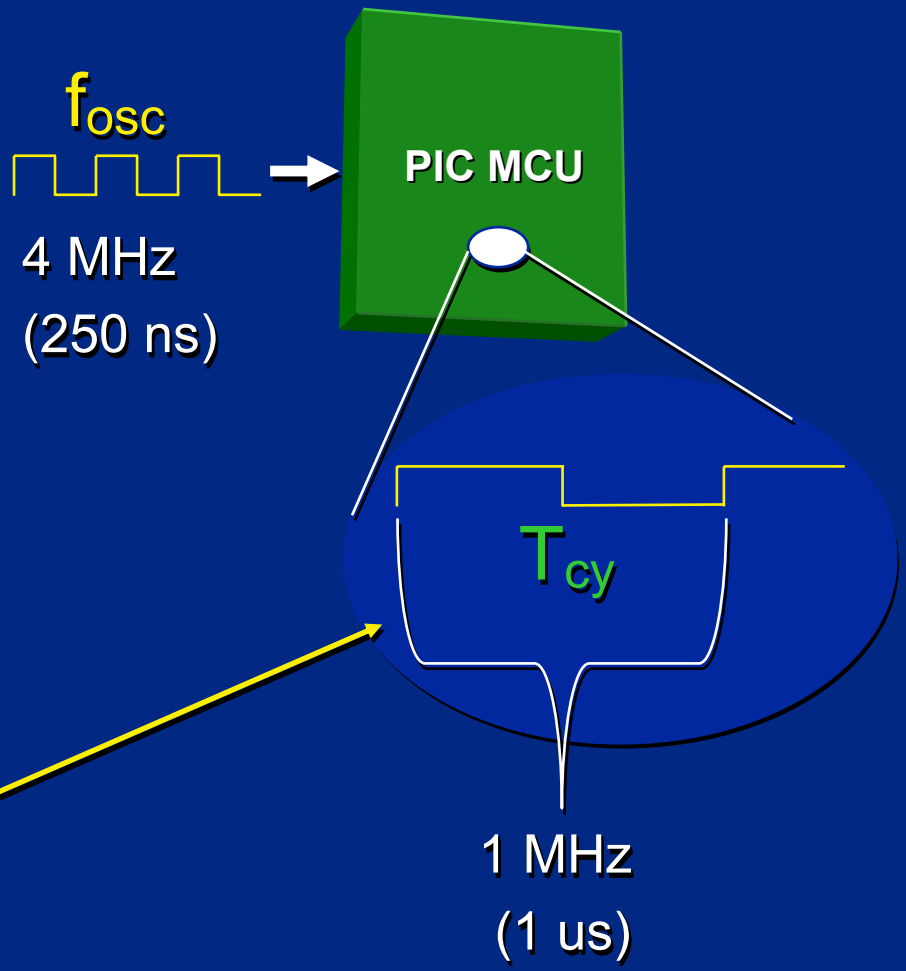
PICmicro 架构
指令实例

- PIC MCU 指令编码为操作码和参数
- 编码用一个字完成

立即数类指令

PIC MCU 晶振

- 指令率是输入时钟的 1/4
- 输入晶振频率叫做f_{osc}
- 一个指令周期占用 1 T_{cy}



存储器

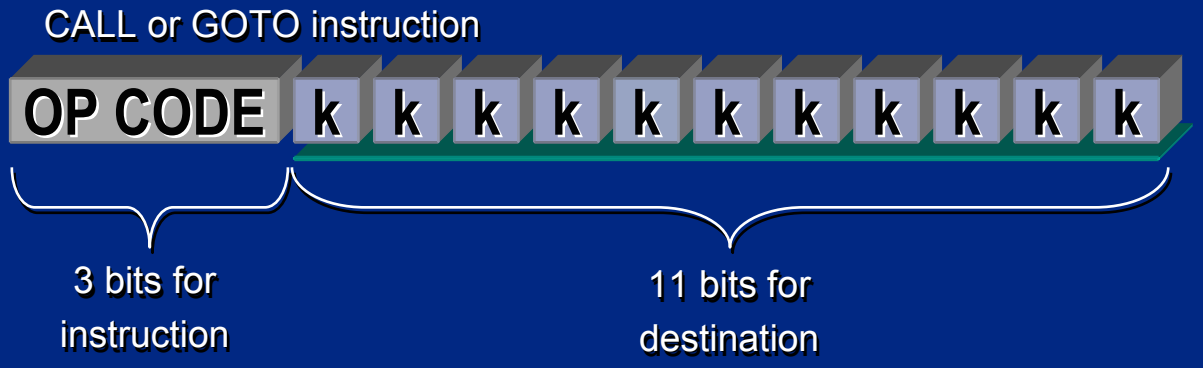
PICmicro 架构
存储器

- 两种类型
  - 程序
  - (RAM 和EEPROM)
- 组成
  - Pages - 页 (程序存储器)
  - Banks – 组 (数据存储器)

PICmicro架构
存储器表

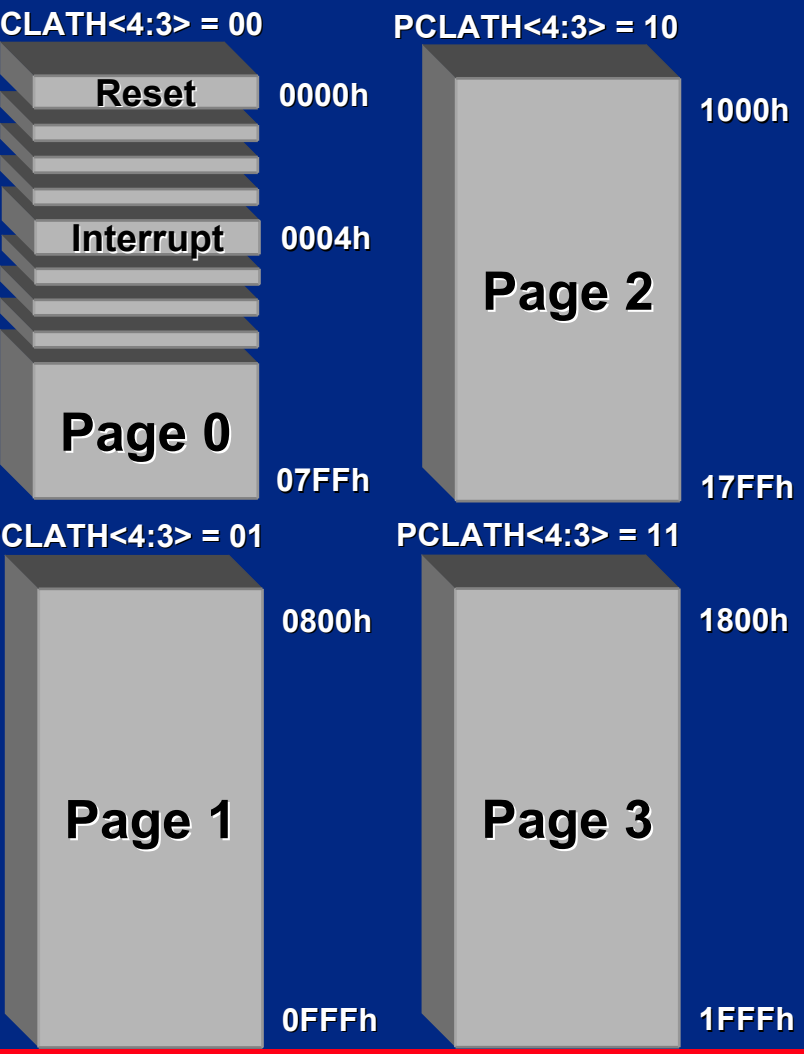
	FLASH 程序存储器	RAM 数据存储器
PICmicro MCU	大小 (words)	GPR 大小 (bytes)
PIC12F675	1 K	64
PIC16F877A	8 K	368

分页的程序存储器 (14bit 核)



- 在14-bit 核的器件中:
  - 11 位访问 2^{11} 或 2048 (2k) 空间
  - 访问更多的空间,需要更多的位
  - 程序存储器的分页提供这些位

14-bit 核架构
程序存储器

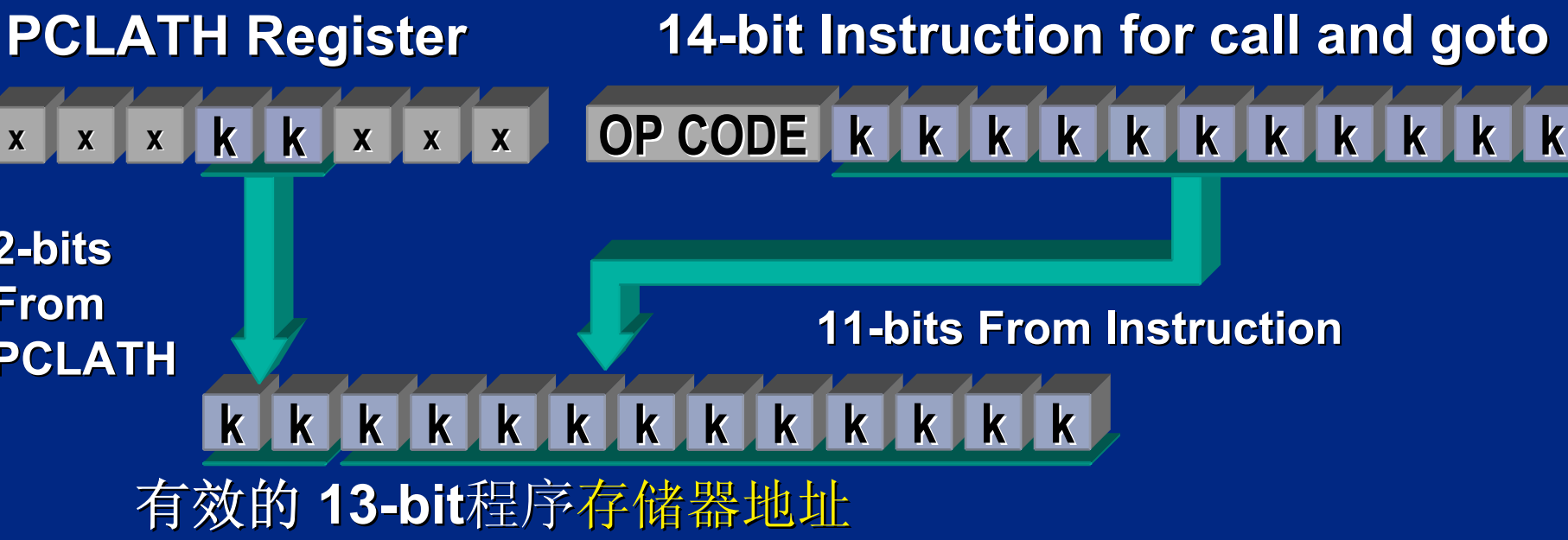


- 程序存储器最大8K字
- 四页,每页 2k
- 使用 PCLATH<4:3>访问页
- 复位地址 0000h
- 中断地址 0004h

PICmicro架构

程序存储器: PC绝对寻址

用CALL和GOTO指令修改PC



PICmicro架构
分页的程序存储器

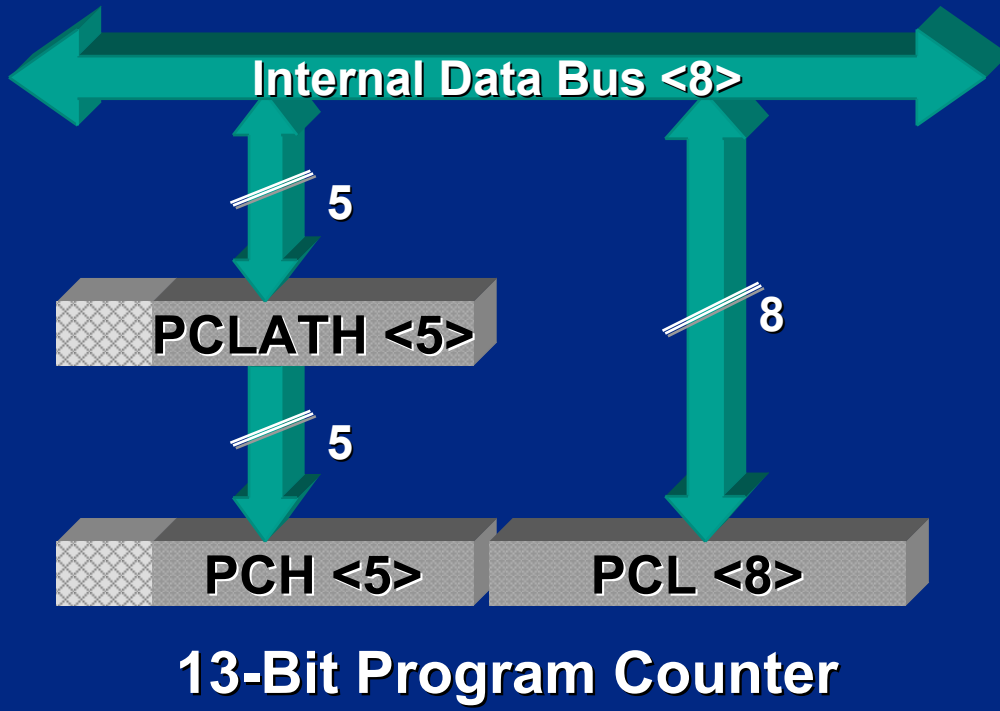
- 当执行CALL or GOTO 指令时,需考虑分页
- 当跳转到不同的页时,修改页位
- 调用页位:
  - GOTO <*地址*>
  - CALL <*地址*>
  - <Instruction> PCL,F ; e.g. ADDWF PCL,F
- 返回时不需考虑页

PICmicro 架构

PC 相对寻址 (14-bit core)

- 高字节写入**PCLATH.**
- 然后低字节写入 **PCL,** 将装入**13-bit** 值到 **PC.**

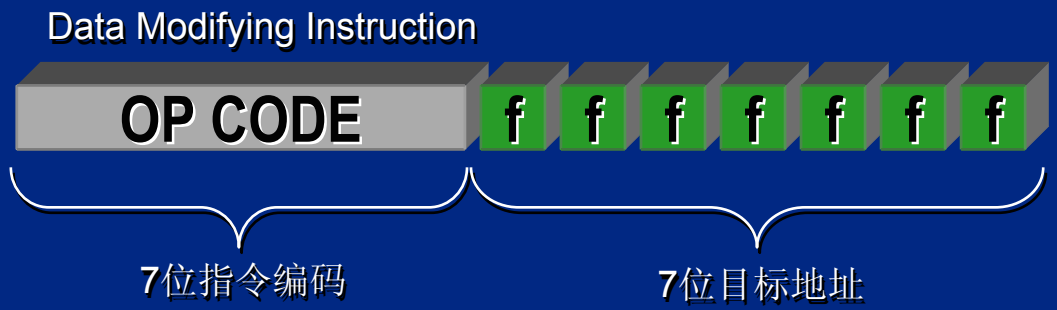
```
movlw    HIGH Delay
movwf    PCLATH
movlw    LOW Delay
movwf    PCL
```



13-Bit Program Counter

Note: PCH cannot be read

分组的 数据存储器
(14 bit core)

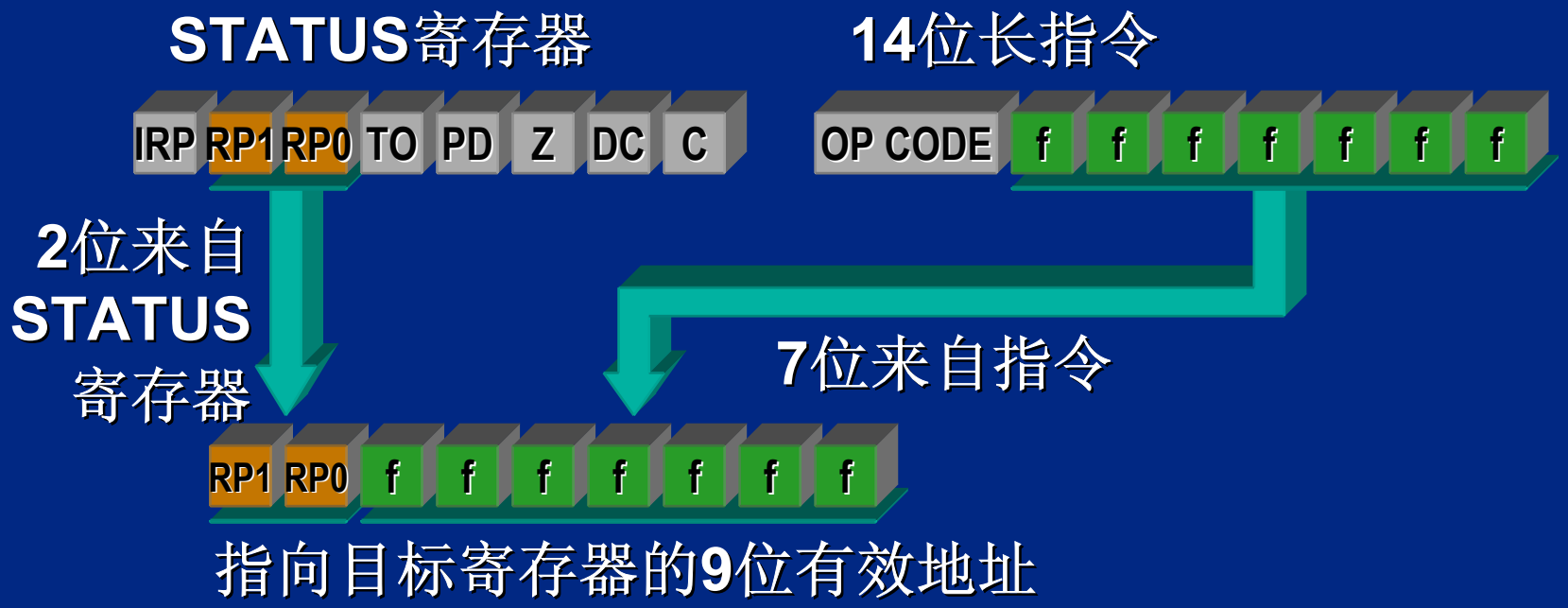


- 在14-bit 核的器件中:
  - 7 位访问 2^7 或 128 空间
  - 访问更多的空间,需要更多的位
  - 程序存储器的分组提供这些位

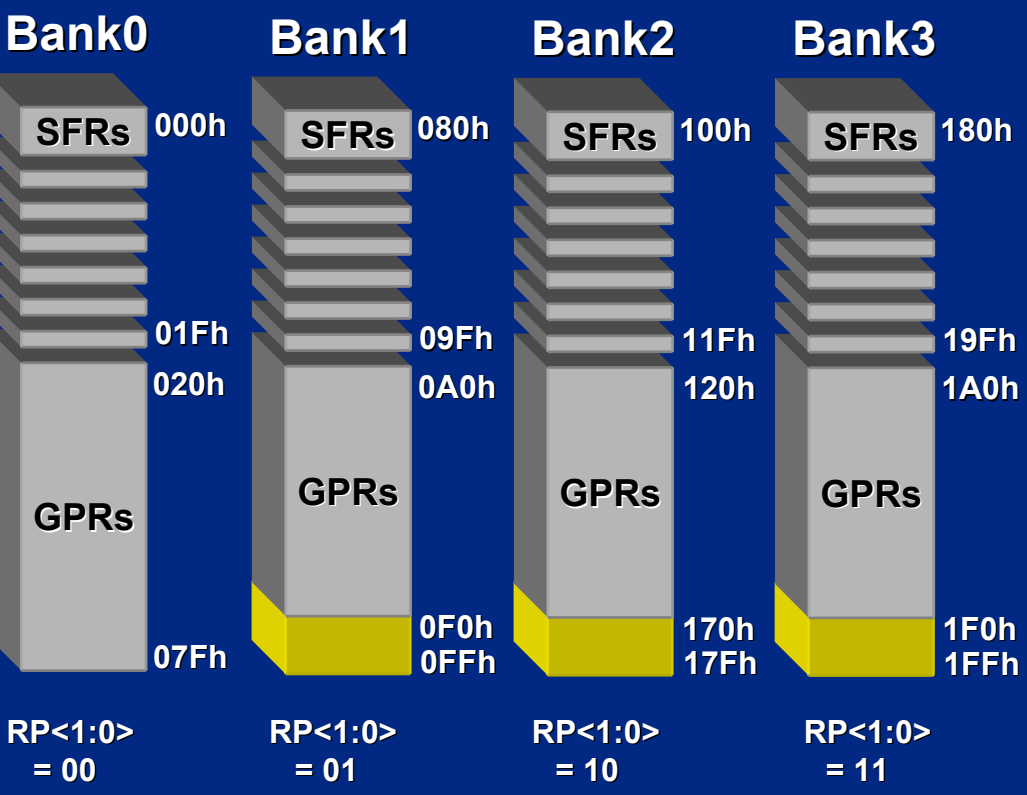
PICmicro 架构

数据存储器: 直接寻址

- 低7位的数据地址直接在指令中描述
- 高2位在STATUS寄存器中 (又称寄存器组号BANK)



PICmicro 架构
数据存储器组成



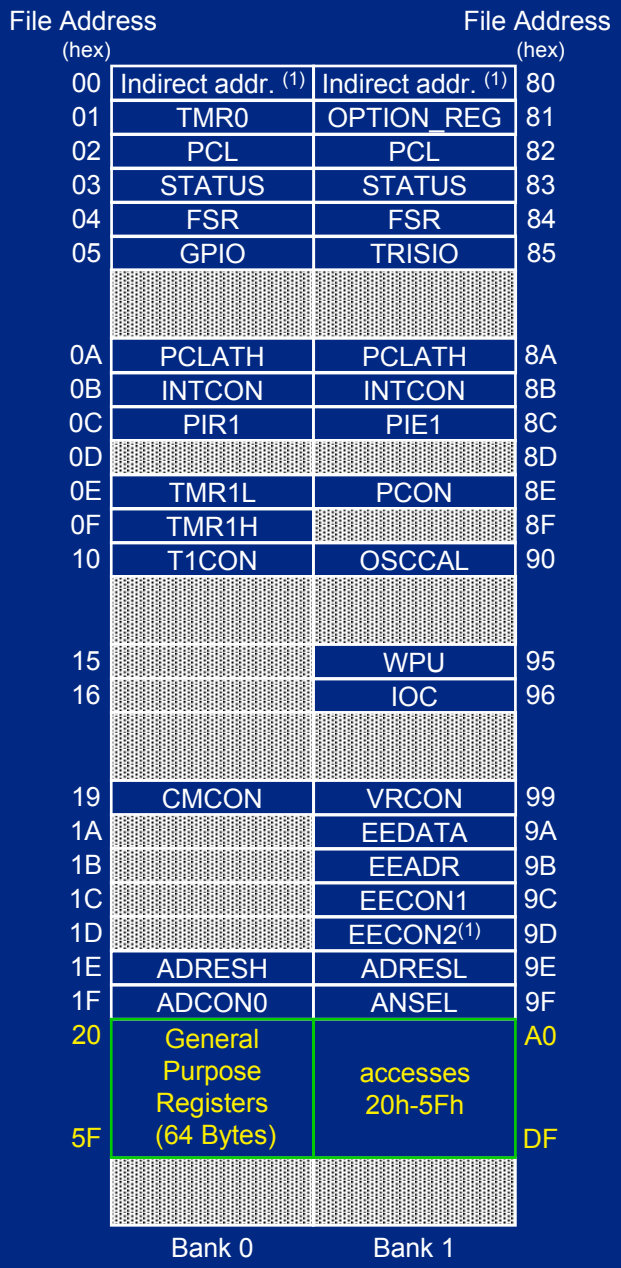
- 分成4个组
- SFR映射到最顶端32 位置
- 通过RP0,1 和 IRP 选择组

MICROCHIP MASTERs

PICmicro 架构
数据存储器

- Unimplemented data memory location; read as '0'.
- Special Function Registers (SFR).
- General Purpose Registers (GPR), user RAM.

1 Not a Physical Register.



寄存器图
PIC12F675

PICmicro MCU架构
PC相对寻址

```
  movlw   HIGH Decode
  movwf   PCLATH
  movf    DisplayValue,W
  call    Decode
  movwf   PORTB
  goto    Continue
ecode
  addwf   PCL,F
  retlw   B'00111111' ;decode 0
  retlw   B'00000110' ;decode 1
  retlw   B'01011011' ;decode 2
  retlw   B'01001111' ;decode 3
  retlw   B'01100110' ;decode 4
  retlw   B'01101101' ;decode 5
  retlw   B'01111101' ;decode 6
  retlw   B'00000111' ;decode 7
  retlw   B'01111111' ;decode 8
  retlw   B'01101111' ;decode 9
ontinue
```

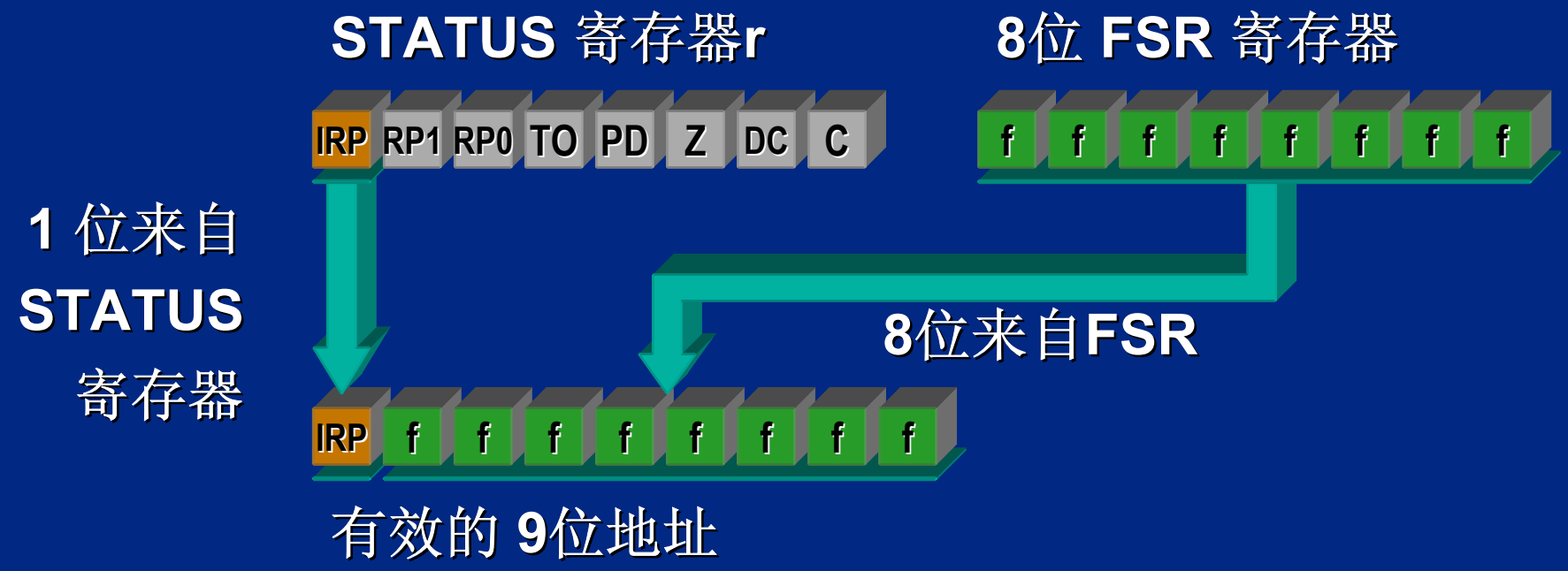
- 实现查找表范例

01101101
W Register

01101101
I/O Port B

PICmicro架构
间接寻址

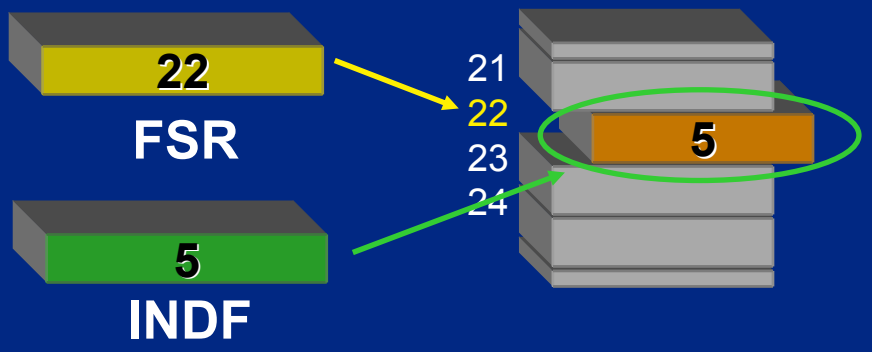
- 用FSR寄存器作为8位地址指针
- 另外1位在STATUS中描述



PICmicro架构
间接寻址

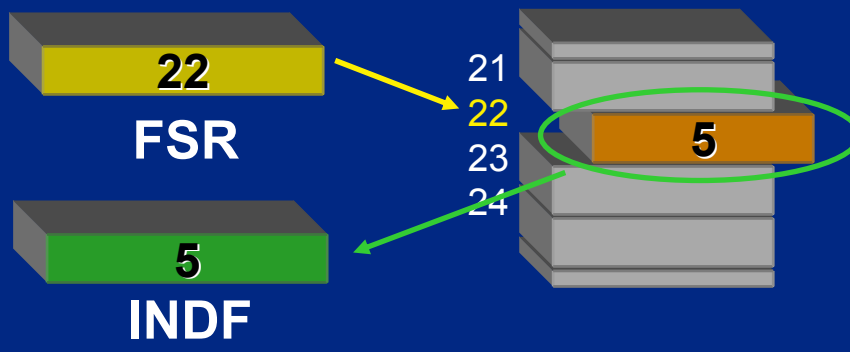
- FSR 和 INDF 用于间接寻址
- FSR 是地址指针
- INDF 保存移动的值

Writing




```
movlw   22
movwf   FSR
movlw   5
movwf   INDF
```

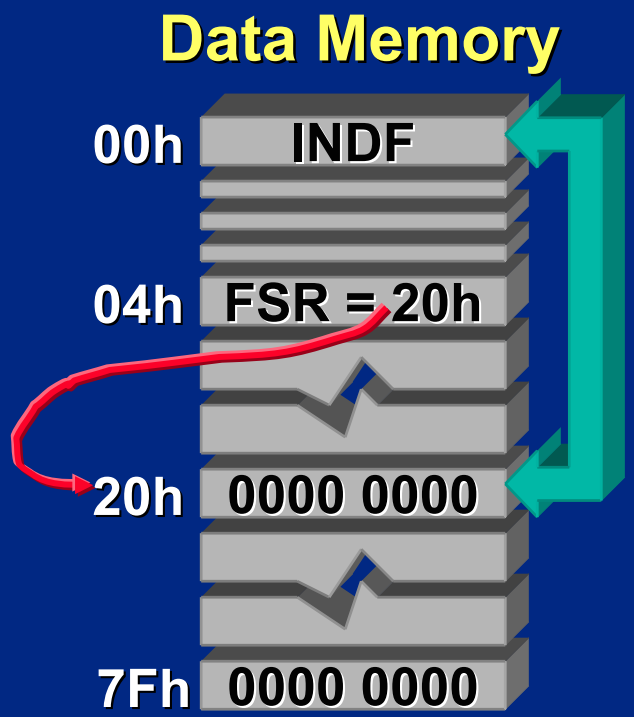
Reading




```
movlw   22
movwf   FSR
movf    INDF,W
movwf   5
```

PICmicro架构
间接寻址

- 清除 0x20 to 0x7F的RAM.




```
        movlw   0x20
        movwf   FSR
LOOP    clrf    INDF
        incf    FSR,F
        btfss   FSR,7
        goto    LOOP
        <next instruction>
```

指令系统

PICmicro MCU 指令集
(14-bit core)

- 35 条指令
  - 易学
  - 紧凑
  - 单字指令
  - 向上兼容

PICmicro指令集

字节操作

NOP	-
MOVWF	f
CLRW	-
CLRF	f
SUBWF	f,d
DECF	f,d
IORWF	f,d
ANDWF	f,d
XORWF	f,d
ADDWF	f,d
MOVF	f,d
COMF	f,d
INCF	f,d
DECFSZ	f,d
RRF	f,d
RLF	f,d
SWAPF	f,d
INCFSZ	f,d

14位内核之字节操作指令构成

d = Destination Bit

d = 0 for destination W

d = 1 for destination F

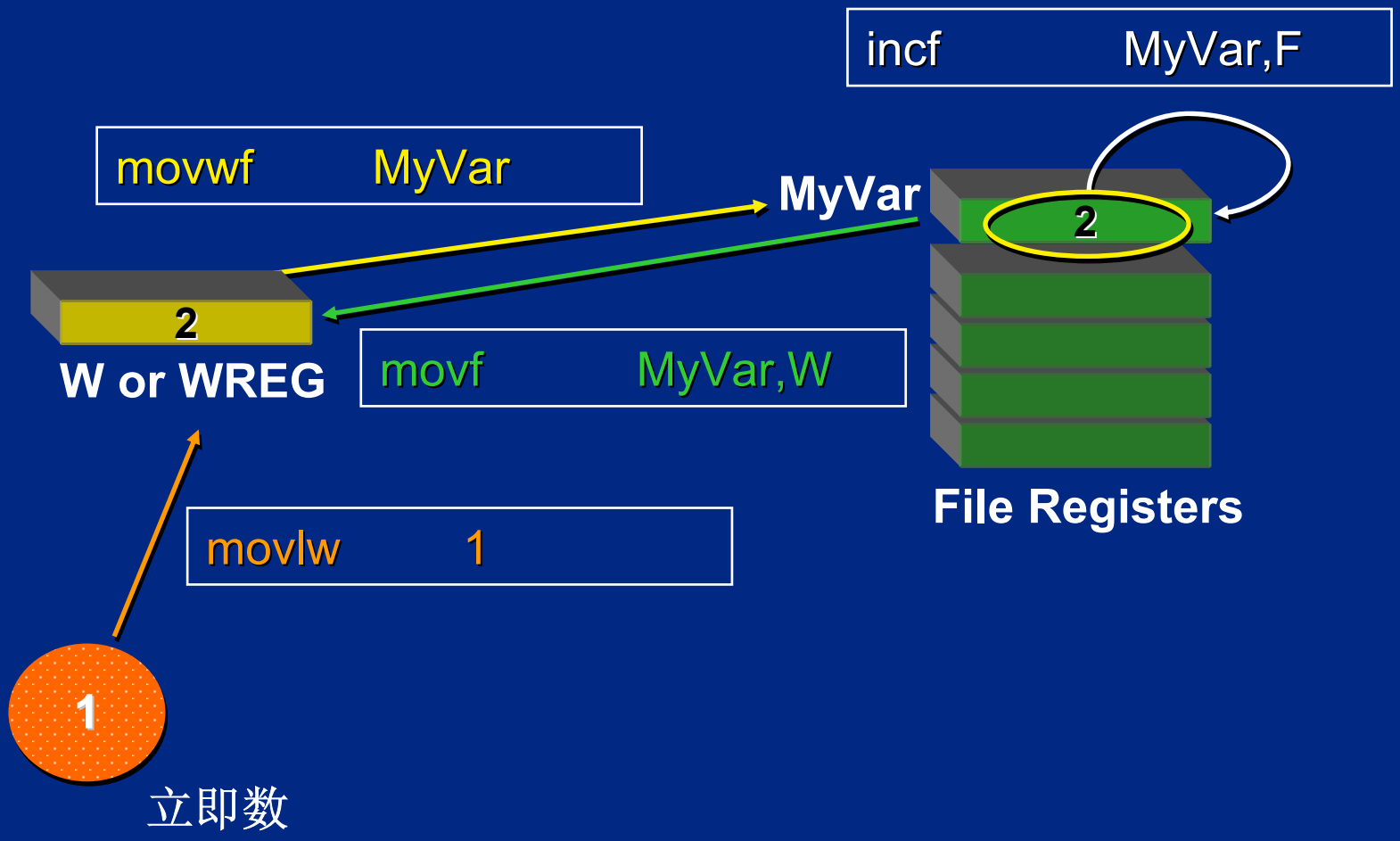
f = 7-bit Register Address

Example:

ADDWF REG, W

ADDWF f, d

移动数据: 立即数, W and F



PICmicro指令集汇总

MICROCHIP MASTERs

Byte-Oriented Operations

ADDWF	f,d	Add W and f
ANDWF	f,d	AND W and f
CLRF	f	Clear f
CLRW	-	Clear W
COMF	f,d	Complement f
DECF	f,d	Decrement f
DECFSZ	f,d	Decrement f, skip if zero
INCF	f,d	Increment f
INCFSZ	f,d	Increment f, skip if zero
IORWF	f,d	Inclusive OR W and f
MOVF	f,d	Move f
MOVWF	f	Move W to f
NOP	-	No Operation
RRF	f,d	Rotate right f through carry
RLF	f,d	Rotate left f through carry
SUBWF	f,d	Subtract W from f
SWAPF	f,d	Swap nibbles of f
XORWF	f,d	Exclusive OR W and f

Bit-Oriented Operations

BCF	f,b	Bit clear f
BSF	f,b	Bit set f
BTFSC	f,b	Bit test f, skip if clear
BTFSS	f,b	Bit test f, skip if set

Literal and Control Operations

SLEEP	-	Go into standby mode
CLRWDT	-	Clear watchdog timer
RETLW	k	Return, place literal in W
RETFIE	-	Return from interrupt
RETURN	-	Return from subroutine
CALL	k	Call subroutine
GOTO	k	Go to address (k is 9-bit)
MOVLW	k	Move literal to W
IORLW	k	Inclusive OR literal with W
ADDLW	k	Add literal with W
SUBLW	k	Subtract W from literal
ANDLW	k	AND literal with W
XORLW	k	Exclusive OR literal with W

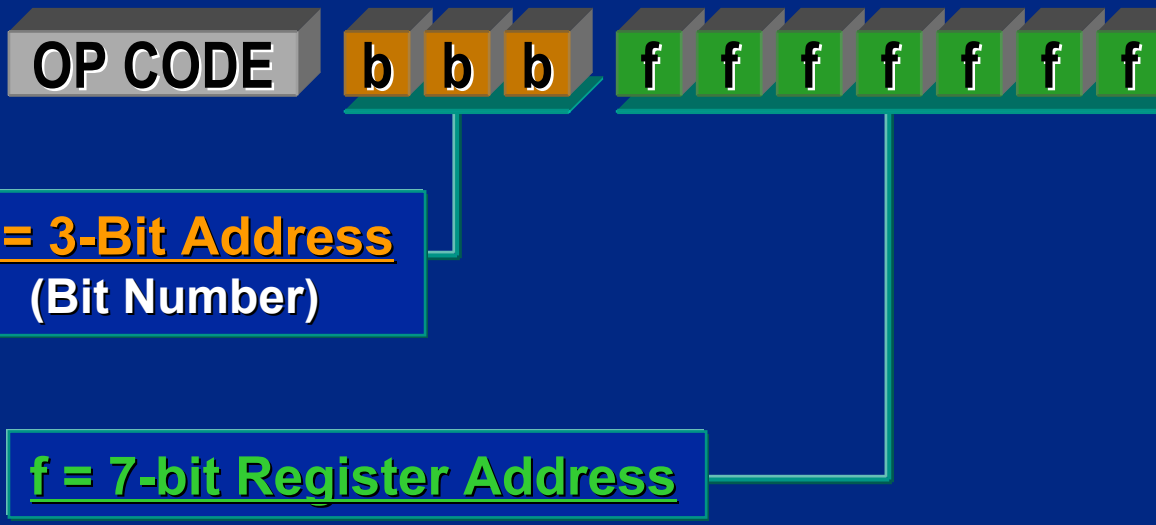
f = File Register, k = literal value (8-bit), b = bit address <0 to 7>, d = destination (W or file register)

PICmicro指令集
位操作

位操作

BCF f,b
BSF f,b
BTFSC f,b
BTFSS f,b

14位内核之位操作指令构成



0000 0000

Bit 7 Bit 0

例:

BTFSC STATUS, C

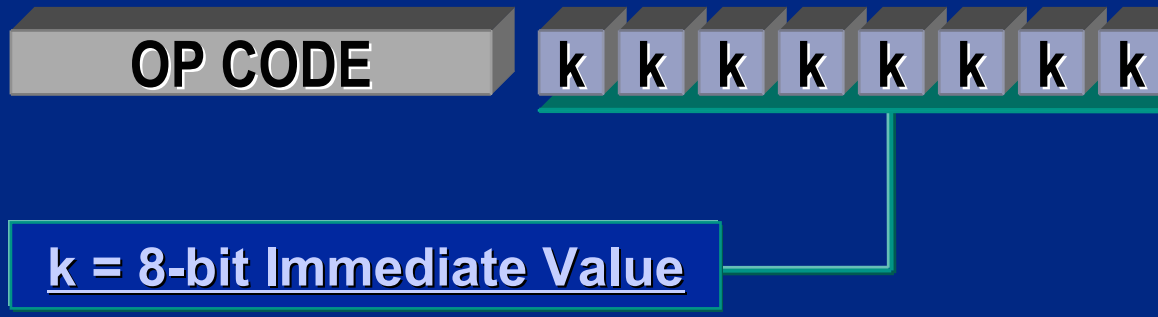
BTFSC f, b

PICmicro指令集
立即数和控制操作

立即数和控制操作

SLEEP	-
CLRWDT	-
RETLW	k
RETFIE	-
RETURN	-
CALL	k
GOTO	k
MOVLW	k
IORLW	k
ADDLW	k
SUBLW	k
ANDLW	k
XORLW	k

14位内核之立即数和控制指令构成



例:

MOVLW 0x2F

MOVLW k

14-bit单片机指令集
操作实例

NOP	No Operation
Syntax:	NOP
Operands:	None
Operation:	No operation
Status:	None
Encoding:	00 0000 0000 0000
Words:	1
Cycles:	1

- 例:

  NOP

14-bit 核指令集
字节操作

Byte-Oriented Operations	
NOP	-
MOVWF	f
CLRW	-
CLRF	f
SUBWF	f,d
DECF	f,d
IORWF	f,d
ANDWF	f,d
XORWF	f,d
ADDWF	f,d
MOVF	f,d
COMF	f,d
INCF	f,d
DECFSZ	f,d
RRF	f,d
RLF	f,d
SWAPF	f,d
INCFSZ	f,d

14位内核之字节操作指令构成

d = Destination Bit

d = 0 for destination W

d = 1 for destination F

f = 7-bit Register Address

Example:

ADDWF REG, W

ADDWF f, d

PICmicro MCU指令集
字节操作

MOVWF	Move W to f
Syntax:	MOVWF f
Operands:	0 <= f <= 127
Operation:	(W) -> (f)
Status:	None
Encoding:	00 0000 1fff ffff
Words:	1
Cycles:	1

- 例:

MOVWF FSR

执行前

FSR = 0xFF

W = 0x4F

执行后

FSR = 0x4F

W = 0x4F

PICmicro MCU指令集

字节操作

CLRW	Clear W
Syntax:	CLRW
Operands:	None
Operation:	00h -> (W) 1 -> Z
Status:	Z
Encoding:	00 0001 0000 0000
Words:	1
Cycles:	1

- 例:

CLRW

执行前

W = 0x4F

执行后

W = 0x00

Z = 1

PICmicro MCU指令集
字节操作

CLRF	Clear f
Syntax:	CLRF f
Operands:	0 <= f <= 127
Operation:	00h -> (f) 1 -> Z
Status:	Z
Encoding:	00 0001 1fff ffff
Words:	1
Cycles:	1

- 例:

CLRF FSR

执行前

FSR = 0x4F

执行后

FSR = 0x00

Z = 1

PICmicro MCU指令集

字节操作

SUBWF	Subtract W from f
Syntax:	SUBWF f,d
Operands:	0 <= f <= 127
	d = {0,1}
Operation:	(f) - (W) -> dest
Status:	C,DC,Z
Encoding:	00 0010 dfff ffff
Words:	1
Cycles:	1

- 例:

SUBWF FSR,W

执行前

FSR = 0x03
W = 0x02
C = ?
Z = ?

执行后

FSR = 0x03
W = 0x01
C = 1
Z = 0

PICmicro MCU指令集
字节操作

DECF	Decrement f
Syntax:	DECF f,d
Operands:	0 <= f <= 127
	d = {0,1}
Operation:	(f) - 1 -> dest
Status:	Z
Encoding:	00 0011 dfff ffff
Words:	1
Cycles:	1

- 例:

DECF FSR,F

执行前

FSR = 0x01

Z = 0

执行后

FSR = 0x00

Z = 1

PICmicro MCU指令集

字节操作

MOVF	Move f
Syntax:	MOVF f,d
Operands:	0 <= f <= 127
	d = {0,1}
Operation:	(f) -> dest
Status:	Z
Encoding:	00 1000 dfff ffff
Words:	1
Cycles:	1

- 例:

MOVF FSR,F

执行前

FSR = 0x00

Z = 0

执行后

FSR = 0x00

Z = 1

PICmicro MCU指令集

字节操作

DECFSZ	Dec. f, Skip if 0
Syntax:	DECFSZ f,d
Operands:	0 <= f <= 127
	d = {0,1}
Operation:	(f) - 1 -> dest
	skip if result = 0
Status:	None
Encoding:	00 1011 dfff ffff
Words:	1
Cycles:	1(2)

例:

```
Loop    DECFSZ  CNT,F
        GOTO    Loop
Continue
```

执行前

PC = address *Loop*

执行后

CNT = CNT - 1

if CNT = 0,

PC = address *Continue*

else CNT != 0,

PC = address *Loop*+1

PICmicro MCU指令集

字节操作

RRF	Rotate Right f -> C
Syntax:	RRF f,d
Operands:	0 <= f <= 127
	d = {0,1}
Operation:	
Status:	C
Encoding:	00 1100 dfff ffff
Words:	1
Cycles:	1

例:

RRF CNT,W

执行前

CNT = 1110 0110

C = 0

执行后

CNT = 1110 0110

W = 0111 0011

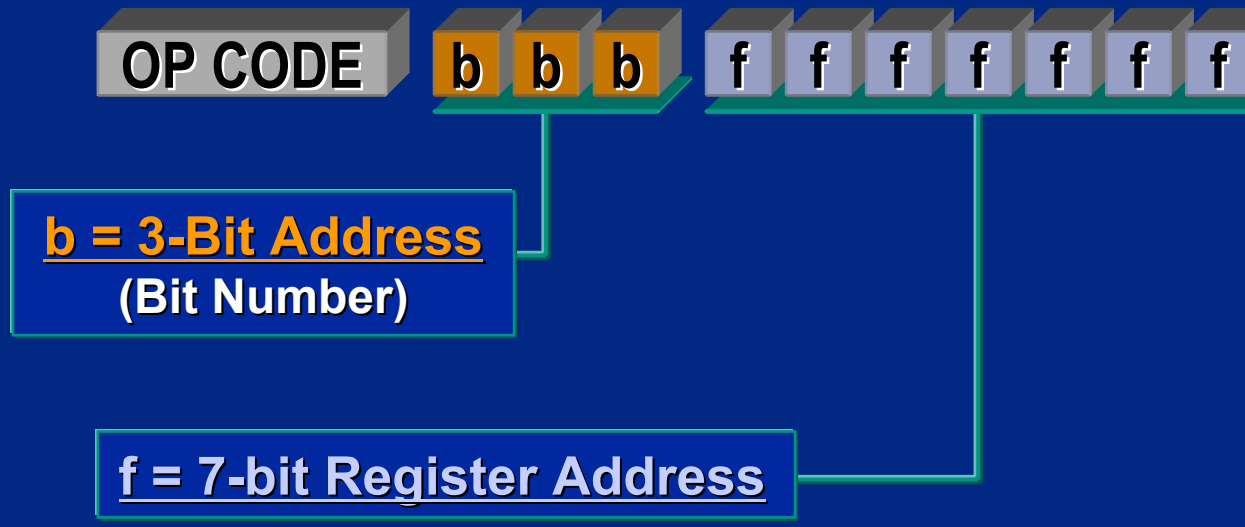
C = 0

PICmicro MCU指令集

位操作

BCF f,b
BSF f,b
BTFSC f,b
BTFSS f,b

14位内核之位操作指令构成



Example:

BTFSC STATUS, C

BTFSC f, b

PICmicro MCU指令集
位操作

BCF	Bit Clear f
Syntax:	BCF f,b
Operands:	0 <= f <= 127
	0 <= b <= 7
Operation:	0 -> (f<b>)
Status:	None
Encoding:	01 00bb bfff ffff
Words:	1
Cycles:	1

- 例:

BCF FSR,4

执行前

FSR = 0011 0000

执行后

FSR = 0010 0000

PICmicro MCU指令集

位操作

BTFSS	Bit Test f, Skip if 1
Syntax:	BTFSS f,b
Operands:	0 <= f <= 127
	0 <= b <= 7
Operation:	skip if (f<b>) = 1
Status:	None
Encoding:	01 11bb bfff ffff
Words:	1
Cycles:	1(2)

例:

```
Here   BTFSS   CNT,2
False  GOTO    Elsewhere
True
```

执行前

PC = address *Here*

执行后

if CNT<2> = 1,

PC = address *True*

if CNT<2> = 0,

PC = address *False*

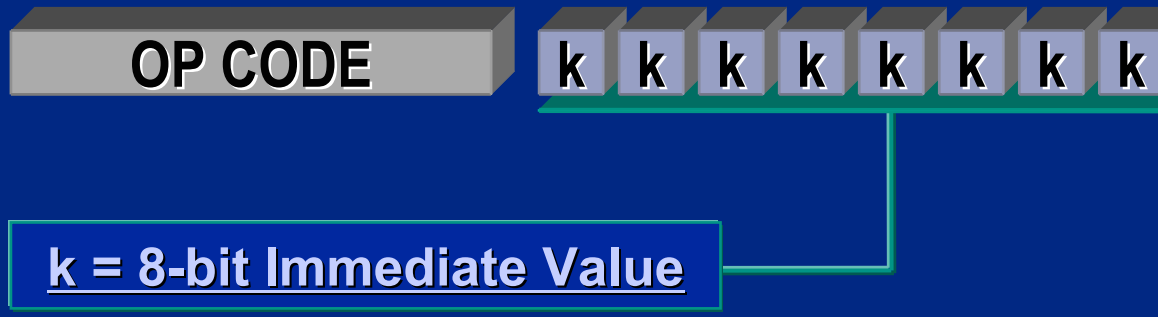
PICmicro MCU指令集

立即数操作

立即数操作

MOVLW	k
IORLW	k
ADDLW	k
SUBLW	k
ANDLW	k
XORLW	k

14位内核之立即数操作指令构成



例:

MOVLW 0x2F

MOVLW k

PICmicro MCU指令集

立即数操作

MOVLW	立即数送给 W
Syntax:	MOVLW k
Operands:	0 <= k <= 255
Operation:	k -> (W)
Status:	None
Encoding:	11 0000 kkkk kkkk
Words:	1
Cycles:	1

- 例:

MOVLW 0x5A

执行后

W = 0x5A

PICmicro MCU指令集

立即数操作

ADDLW W与立即数相加

Syntax: ADDLW k

Operands: 0 <= k <= 255

Operation: (W) + k -> (W)

Status: C,DC,Z

Encoding: 11 1110 kkkk
kkkk

Words: 1

Cycles: 1

- 例:

ADDLW 0x15

执行前

W = 0x10

执行后

W = 0x25

PICmicro MCU指令集

立即数操作

ANDLW	W 与立即数相与
Syntax:	ANDLW k
Operands:	0 <= k <= 255
Operation:	(W) AND k -> (W)
Status:	Z
Encoding:	11 1001 kkkk kkkk
Words:	1
Cycles:	1

- 例:

ANDLW 0x5F

执行前

W = 0xA3

执行后

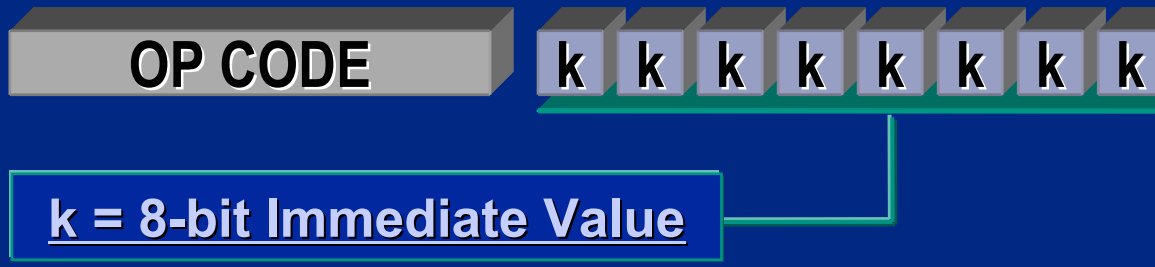
W = 0x03

PICmicro MCU指令集
控制类操作

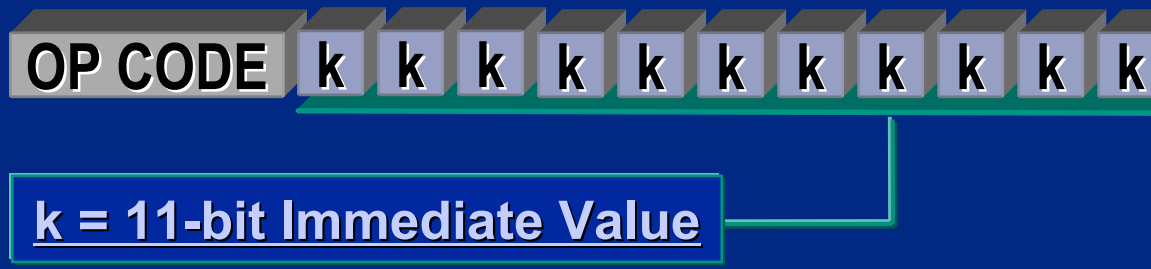
控制类操作

SLEEP	-
CLRWDT	-
RETLW	k
RETFIE	-
RETURN	-
CALL	k
GOTO	k

14位内核之RETLW指令构成



14位内核之CALL /GOTO指令构成



PICmicro MCU指令集

控制类操作

SLEEP	进入 SLEEP
Syntax:	SLEEP
Operands:	None
Operation:	00h -> WDT
	1 -> $\overline{TO}$
	0 -> $\overline{PD}$
Status:	$\overline{TO}$, $\overline{PD}$
Encoding:	00 0000 0110 0011
Words:	1
Cycles:	1

- 例:

  SLEEP

PICmicro指令集
控制类操作

CLRWDT	清 Watchdog
Syntax:	CLRWDT
Operands:	None
Operation:	00h -> WDT
	0 -> WDT prescaler
	1 -> $\overline{TO}$
	1 -> $\overline{PD}$
Status:	$\overline{TO}$, $\overline{PD}$
Encoding:	00 0000 0110 0100
Words:	1
Cycles:	1

- 例:

CLRWDT

PICmicro MCU指令集

控制类操作

RETLW 返回

Syntax: RETLW k

Operands: 0 <= k <= 255

Operation: k -> (W)
TOS -> PC

Status: None

Encoding: 11 0100 kkkk kkkk

Words: 1

Cycles: 2

- 例:

RETLW 0x5A

执行后

W = 0x5A

PICmicro MCU指令集

控制类操作

RETFIE	中断返回
Syntax:	RETFIE
Operands:	None
Operation:	TOS -> PC
	1 -> GIE
Status:	None
Encoding:	00 0000 0000 1001
Words:	1
Cycles:	2

- 例:

RETFIE

PICmicro MCU指令集

控制类操作

CALL	调用子程序
Syntax:	CALL k
Operands:	0 <= f <= 2047
Operation:	(PC) + 1 -> TOS k -> PC<10:0> PCLATH<4:3 -> PC<12:11>
Status:	None
Encoding:	10 0kkk kkkk kkkk
Words:	1
Cycles:	2

例:

Here CALL There

执行前

PC = address *Here*

执行后

PC = address *There*

TOS = address *Here+1*

PICmicro MCU指令集

控制类操作

GOTO 无条件转移

Syntax: GOTO k

Operands: 0 <= f <= 2047

Operation: k -> PC<10:0>
PCLATH<4:3> -> PC<12:11>

Status: None

Encoding: 10 1kkk kkkk kkkk

Words: 1

Cycles: 2

例:

GOTO *There*

执行后

PC = address *There*

使用 MPLAB® 集成开发环境 (IDE)
MPASM™ 编译器

汇编 指令

- #INCLUDE
  - 语法: #include “文件名.*” 或<文件名.*> 或无括号 或 引号:
  - 示例: #INCLUDE p12f675.inc

汇编 指令

- #DEFINE
  - 用于定义变量和分配初始值
    - 用户友好的名字被分配给常数,寄存器 ,管脚名等.
    - 使程序易读
  - 例: #DEFINE MYCONSTANT H'5A'

- _ _CONFIG
  - 设置配置位
  - 见 Lab 2

汇编 指令

- ORG
  - 语法: ORG <address>
  - 例: ORG 0x0000
  - 描述: 定义起始地址(0x0000).

汇编指令

- EQU
  - 语法: <label> equ <expr>
  - 例: Seconds EQU 0x24
  - 描述: 定义常数或变量的替换名.
  - …可用做常数
    - movlw SECONDS ; put 24 (hex) into W
  - ... 或当作寄存器地址
    - movwf SECONDS ; put value in W into
      ; RAM location 24 (hex).

汇编指令

- BANKSEL
  - 语法: BANKSEL [<variable>]
  - 例:

```
BANKSEL     TRISA
movwf       TRISA
```

  - 设置相关的 bank 位

汇编指令

- END
  - 语法: END
  - 例: 表示原代码结束.

- 你必须在想结束汇编的地方有一条 END语句.

MICROCHIP MASTERs

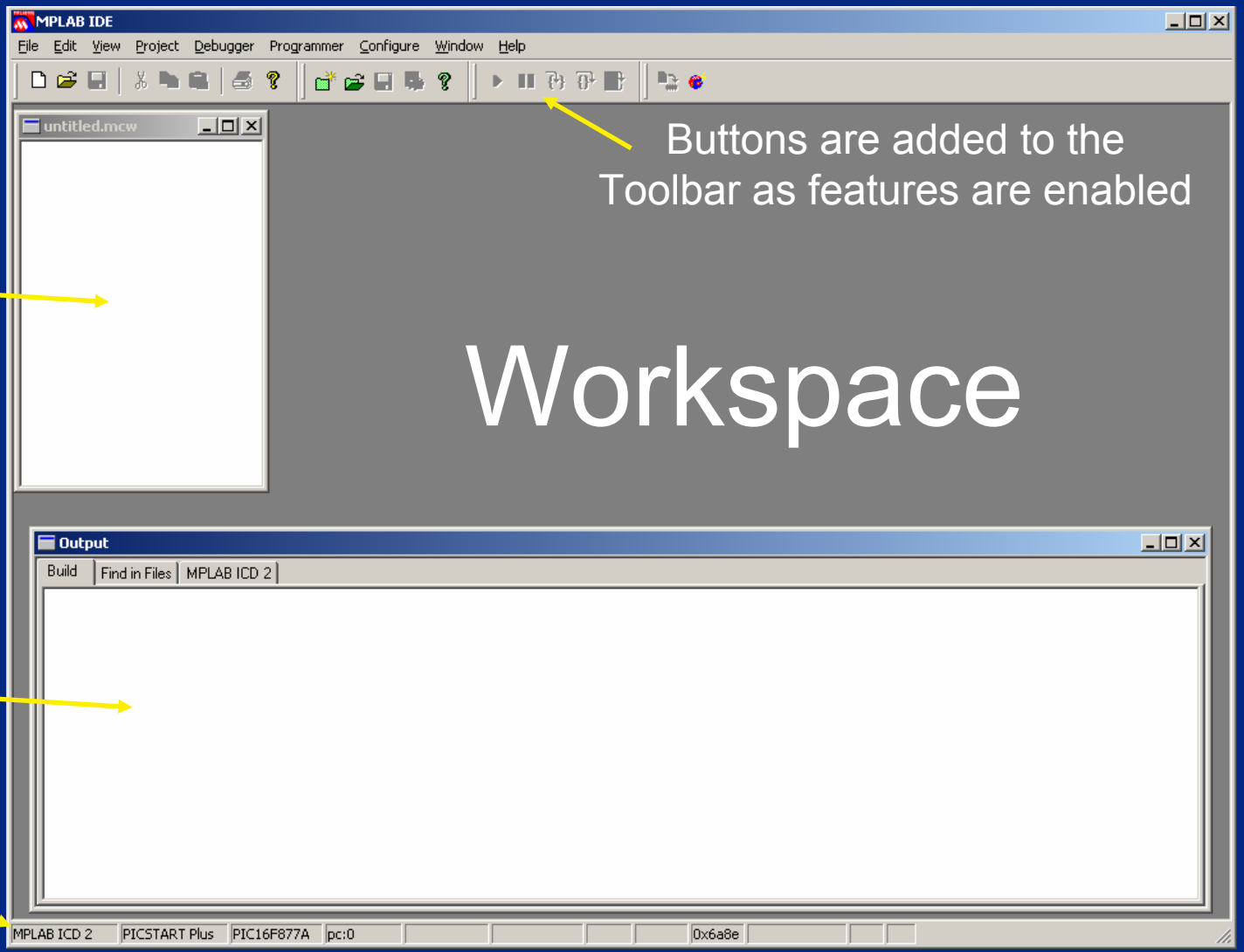
休息

MICROCHIP MASTERs

Lab 1

在MPLAB 7.x中创建一个项目

Lab 1
MPLAB® 7.xx 布局



图标栏

项目窗口

输出窗口

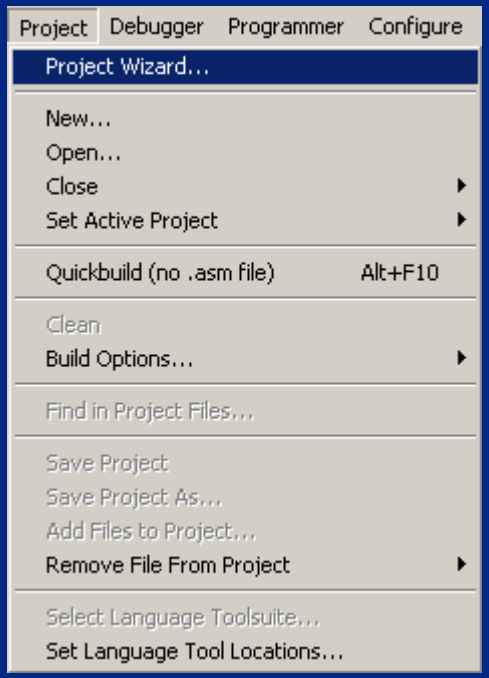
状态栏

创建项目

- Lab 1 目标
  - 使用Project Wizard
  - 编写简单的MPASM代码
  - 编译代码
  - 介绍 输出窗口 和Program Memory 窗口
  - 选择Build选项

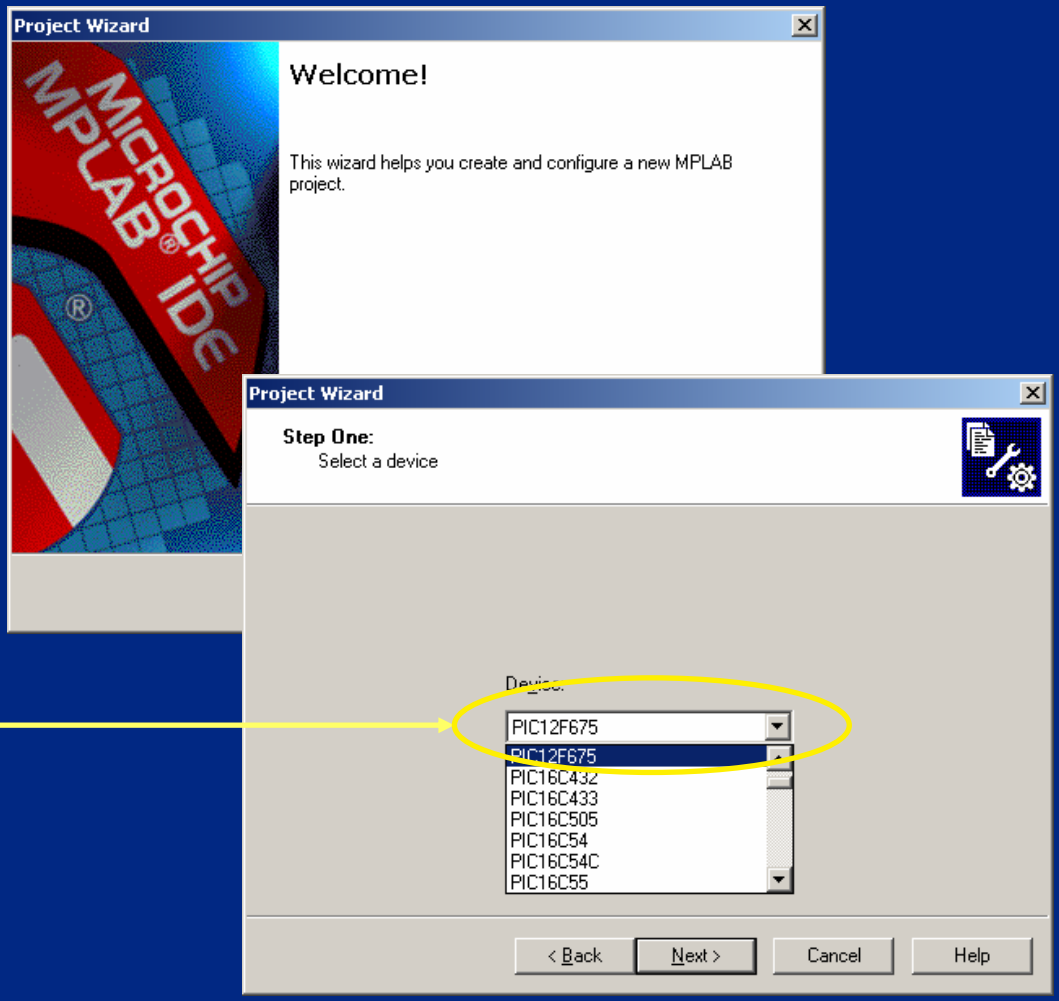
创建项目

- Project -> Project Wizard...
  - 打开 Project Wizard ,创建一个项目



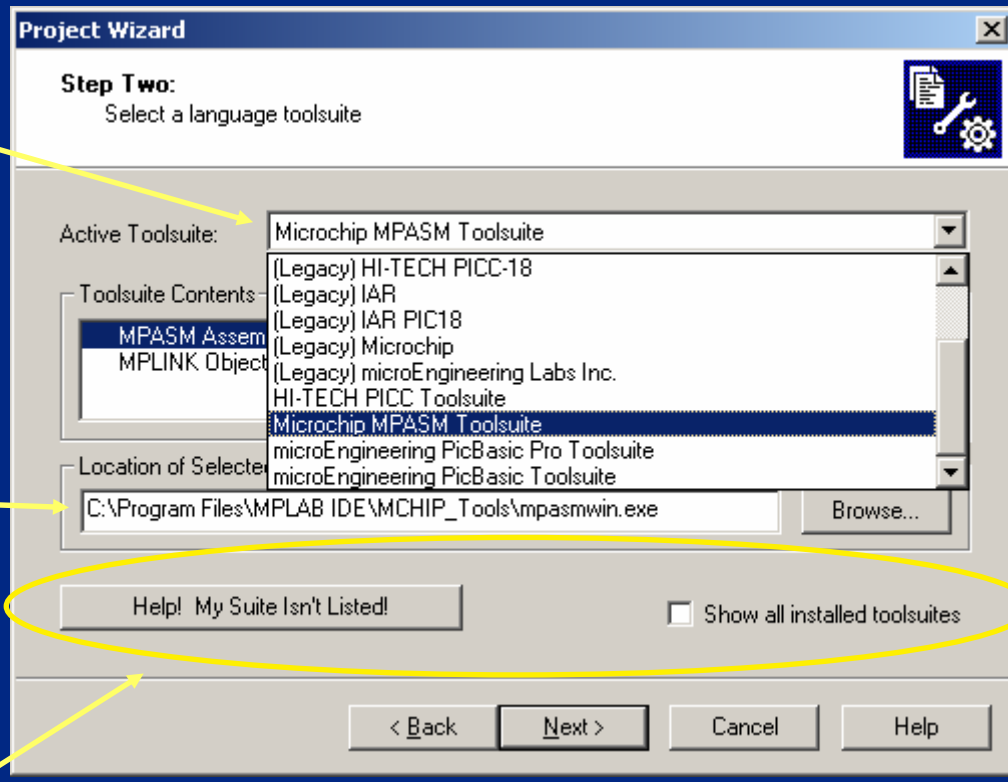
Lab 1
从任一项目开始

- 按照下列步骤创建项目:
  - 点击“Next”继续.
- 选择MCU型号
  - 使用最新的 MPLAB IDE 版本



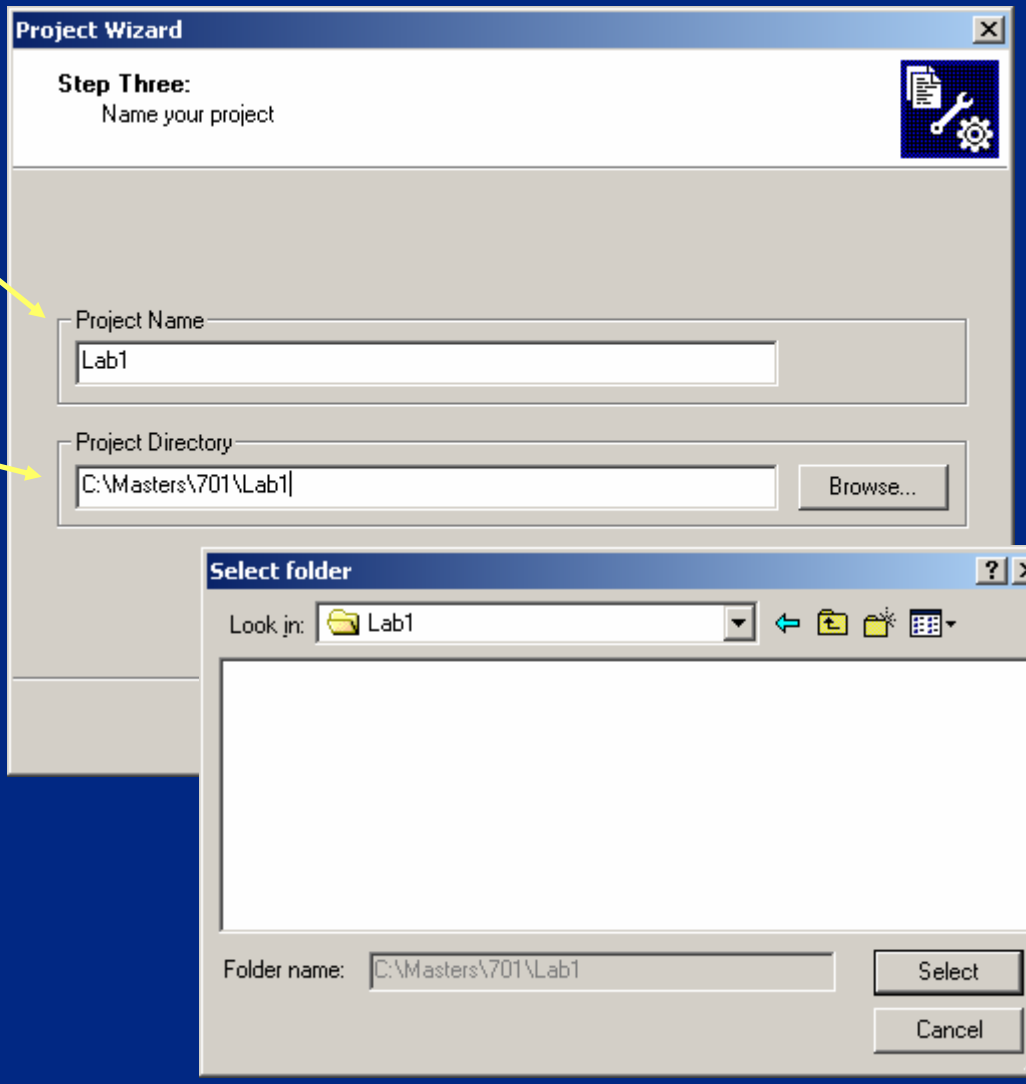
Lab 1
从任一项目开始

- 选择语言工具:
  - MPASM 汇编器 免费,本课将使用 .
  - 显示安装后子目录, 你也可以更改 .
  - 当然你可以点出 “Help” 或 “Show all”查看所有的语言工具.

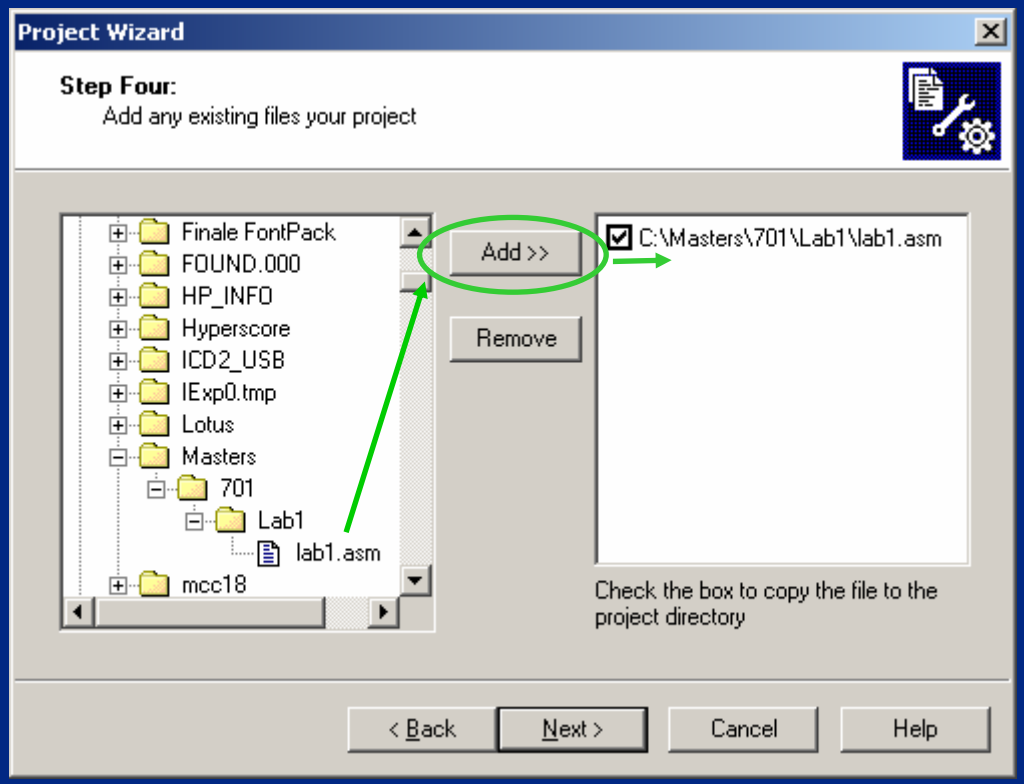


Lab 1
从任一项目开始

- 选择路径:
  - 键入项目名字.
  - 键入项目路径.
  - 注意路径尽可能短.



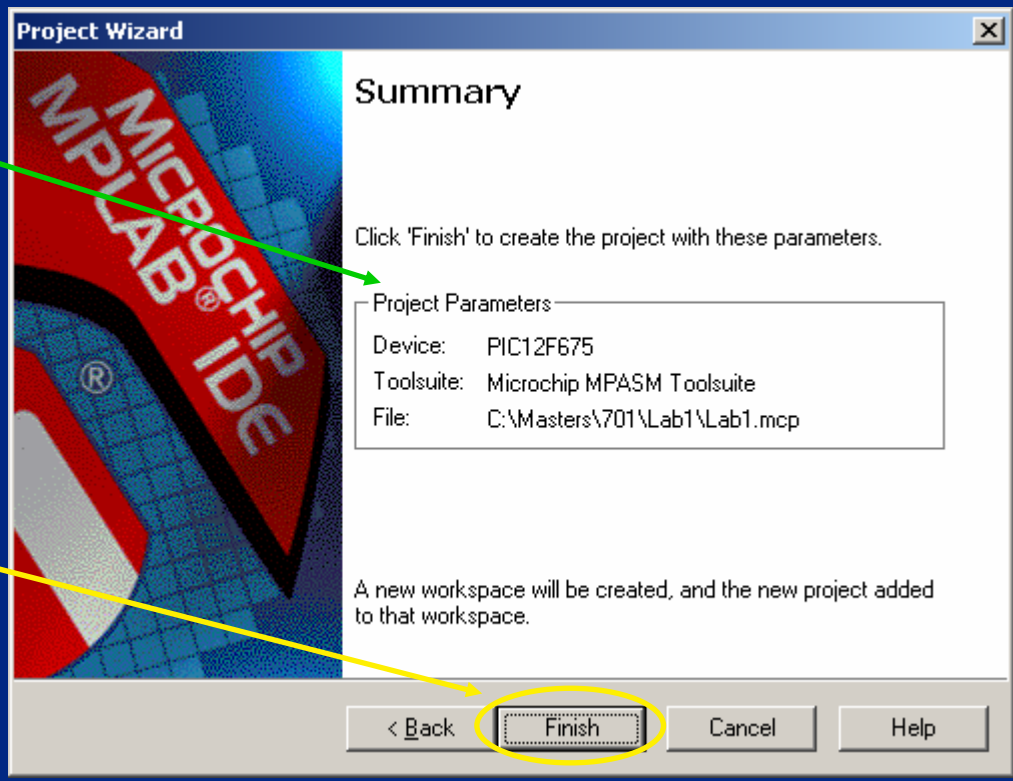
Lab 1
从任一项目开始



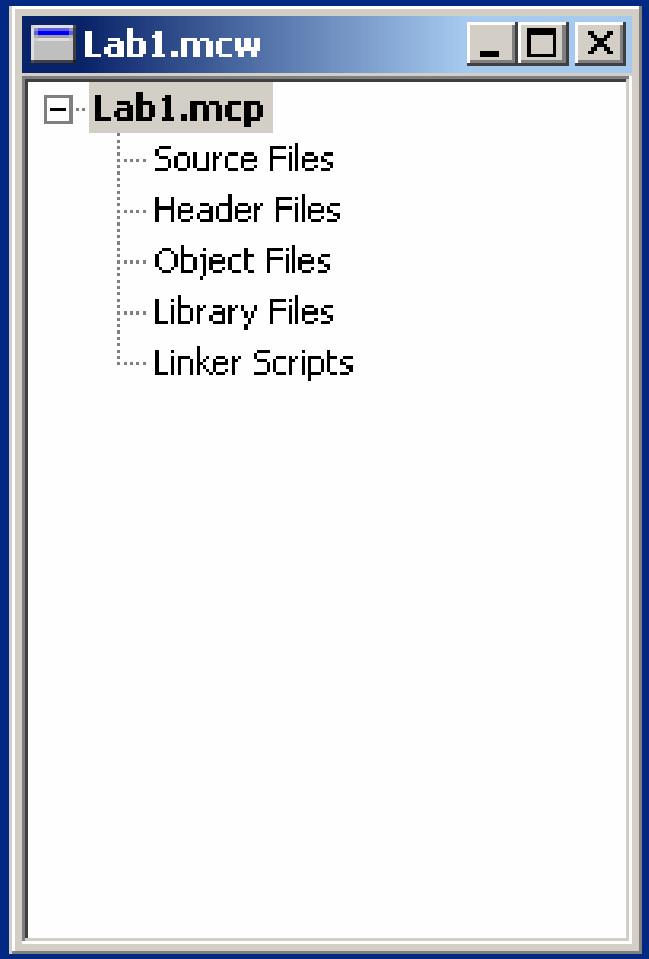
- 增加文件
  - 增加文件到项目中.

Lab 1
从任一项目开始

- 完成
  - 创建的项目的小结被显示，点击“finish”完成 .
  - 记住，Project Wizard 新版本将会增加一些新的功能.

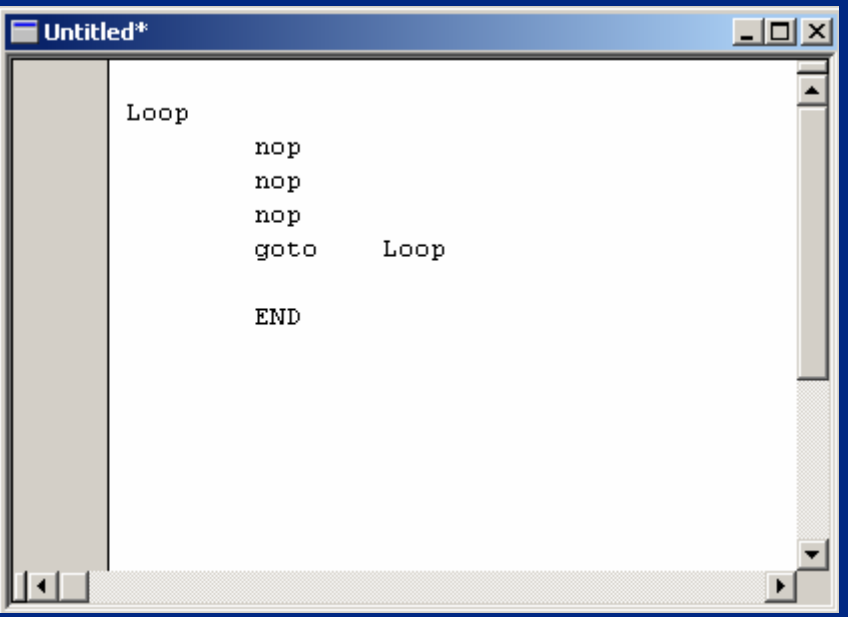


Lab 1
从任一项目开始



- (*.mcp)和 (*.mcw)文件名将被显示 .
- 在窗口你可以添加任何类型相关的文件到项目中

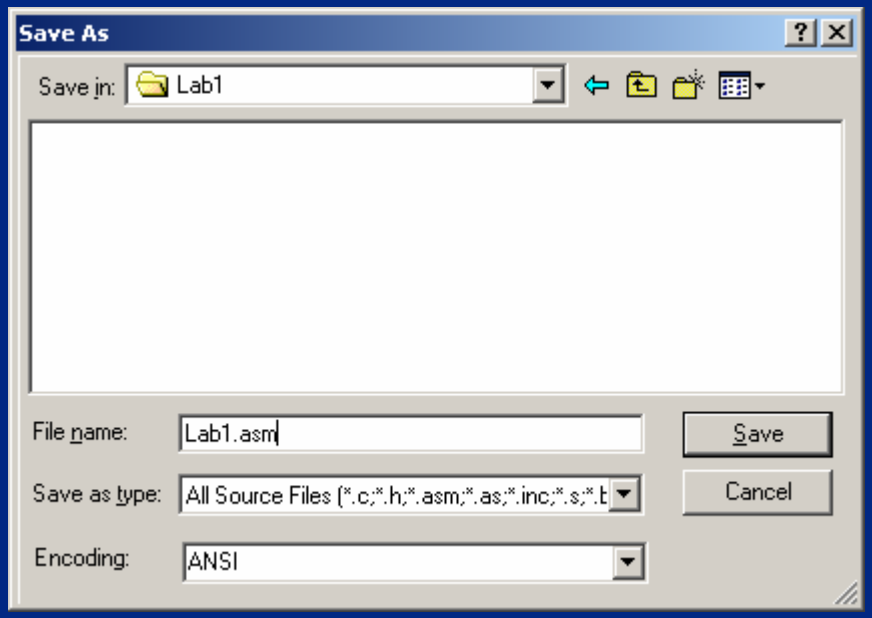
Lab 1
从任一项目开始



- 选择File > New创建新文件
- 键入右图所示的代码.
- “*”表示文件已被改变.
- 键入的代码执行*nop*’s功能.

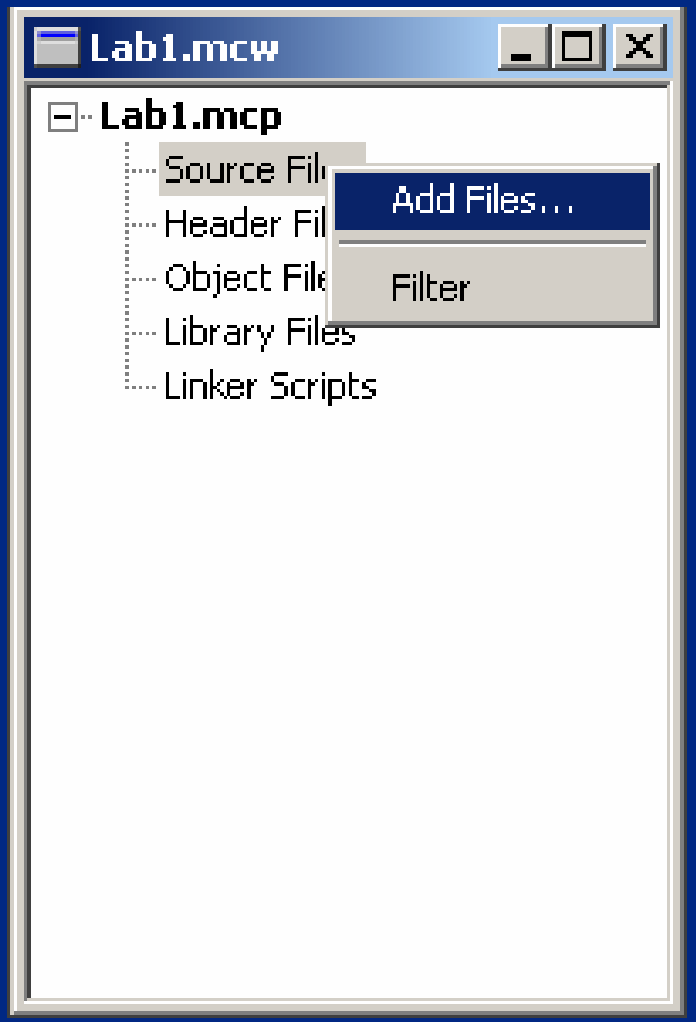
Lab 1
从任一项目开始

- 选择File > Save As保存文件..
- 保存 例子文件 为 *Lab1.asm.*

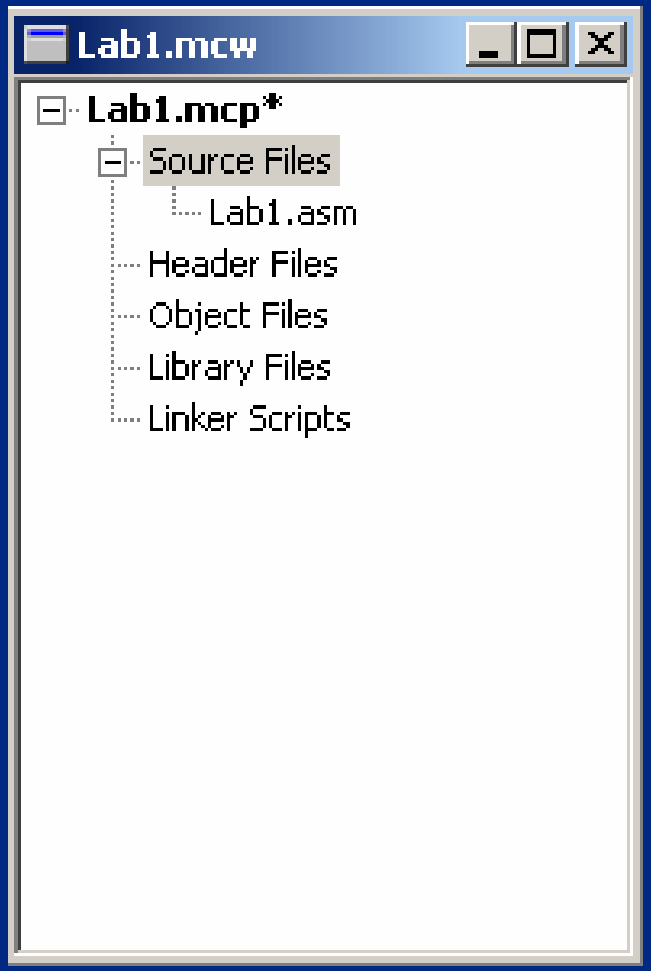


Lab 1
从任一项目开始

- 在source code栏增加文件
- 鼠标右击 “Source Files” 选择 “Add Files”.
- 增加 *Lab1.asm文件.*

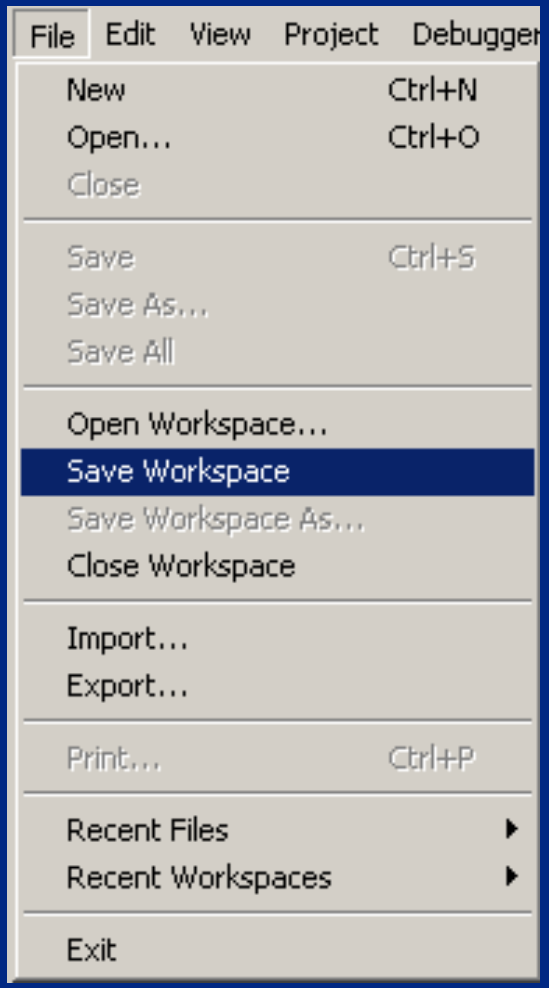


Lab 1
从任一项目开始



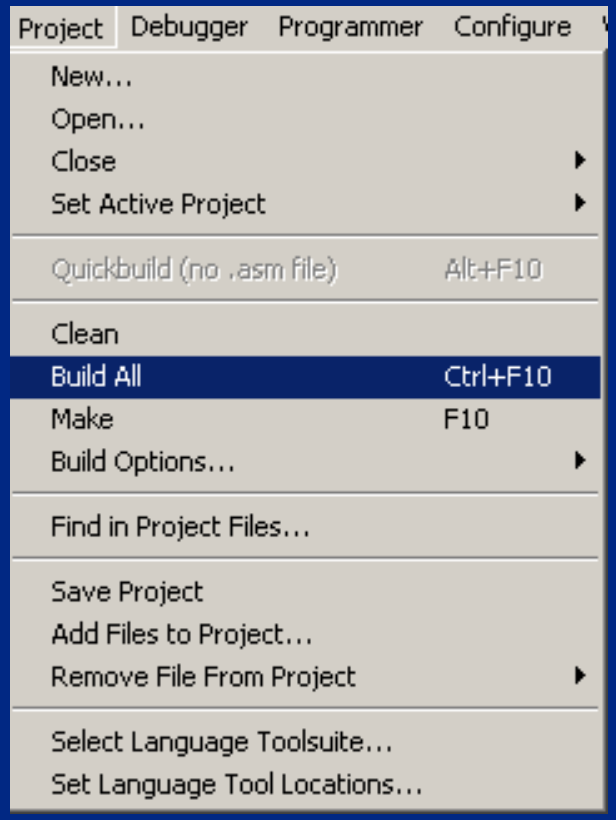
- 文件现在增加在Project中
- 注意“*”表示文件已被改变未保存.

Lab 1



- 选择File > Save Workspace保存项目

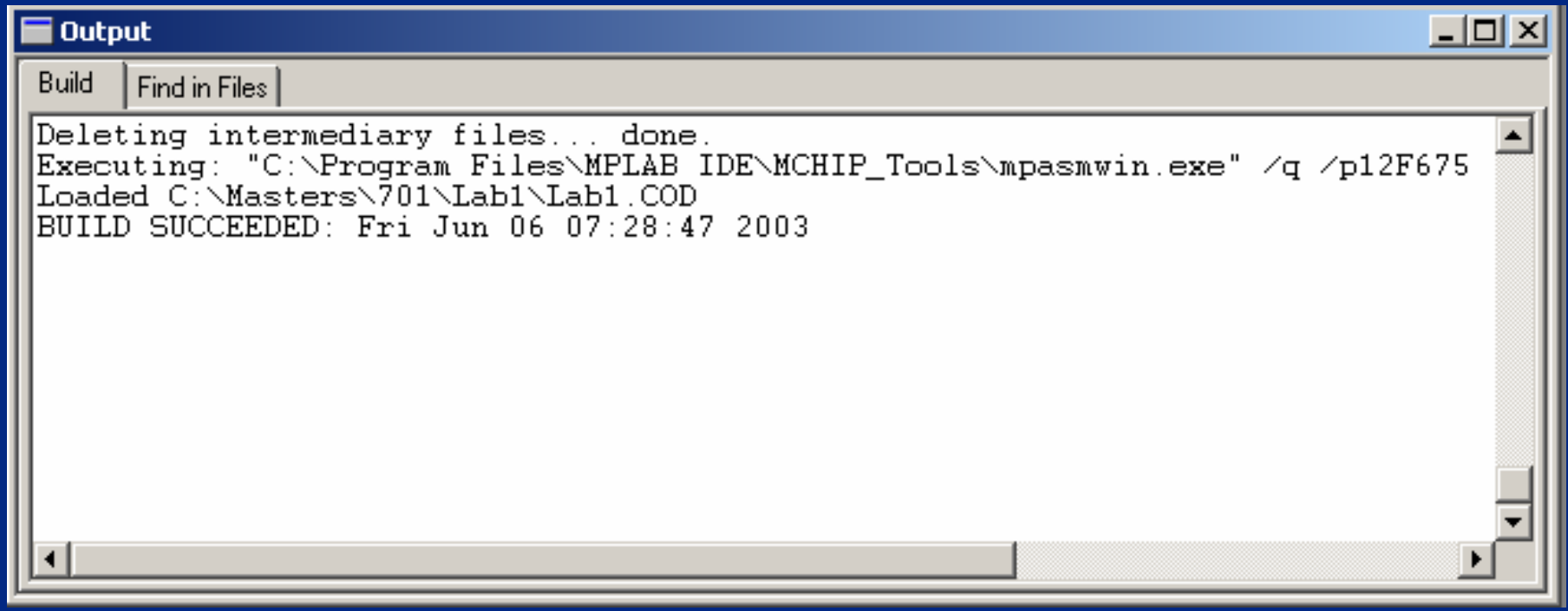
编译代码



- 选择Project -> Build All编译项目
- “Build All” 表示 每一个文件被改变,因此重新编译所有项目

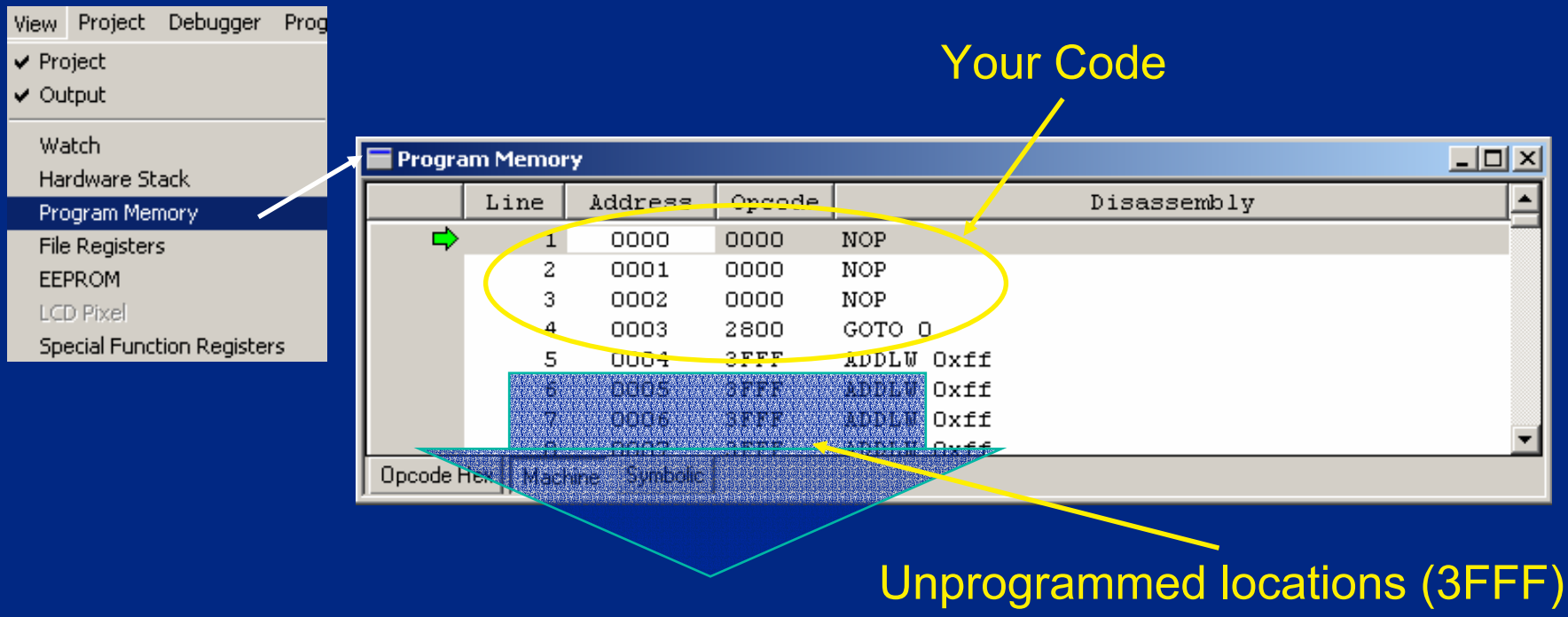
Lab 1
从任一项目开始

- 输出窗口 *显示* build的过程.



Lab 1
从任一项目开始

- 选择View > Program Memory可以查看程序区



MPLAB® IDE
设置信息

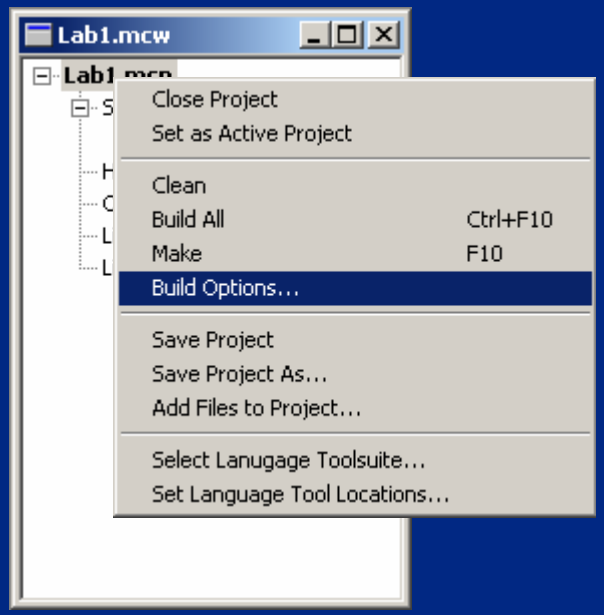
- Workspace:
  - 包含下列信息:
    - MCU型号
    - 调试工具或编程器
    - 打开的窗口
    - IDE configuration 的设置值
- Project:
  - 包含所有的源文件

MPLAB® IDE 窗口

- 新的工作界面有下列窗口
  - Project - 工程项目
  - Output - 输出信息
  - Disassembly - 反汇编
  - Watch - 变量观察
  - Hardware Stack - 硬件堆栈信息
  - Program memory - 程序空间内容
  - File Registers - 寄存器内容
  - EEPROM data - EEPROM数据内容
  - Special Function Registers - 特殊功能寄存器

Note: There are often more windows, but they will depend on the device used and debugger/programmer tool.

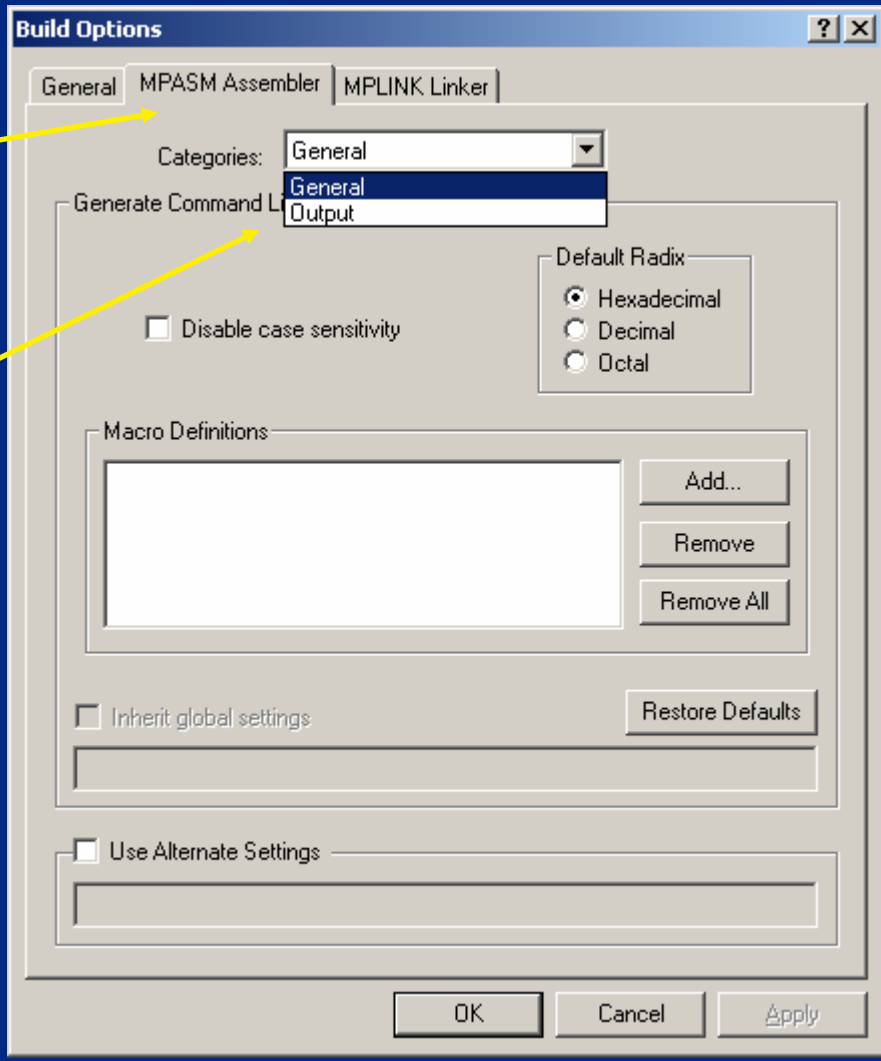
Lab 1
设置 Build 选项



- 在Project Window右击项目文件 (也可以选择“Project”菜单)
- 选择“Build Options...”更改设置

Lab 1
设置 Build 选项

- The Build Options 窗口有多个设置窗口
- 使用下拉菜单改变类型
- 选择你需要的选项
- 通常采用缺省值

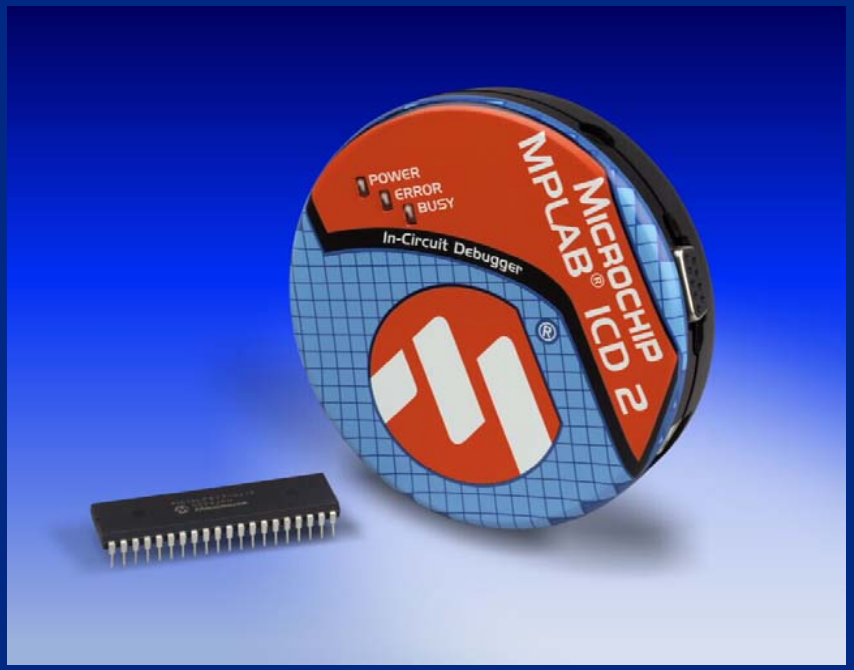


Lab 1 – 注意事项

- 首先必须生成项目 或 文件
- .asm文件必须添加到项目
- 使用 Configuration 菜单选择型号
- 编译 项目生成 .hex 文件

MICROCHIP MASTERs

Lab 2
使用 MPLAB ICD 2



开始 Lab 2

- Lab 2 目标
  - 介绍 MPLAB ICD 2 和 PICDEM™ 2 Plus
  - 验证逻辑跳转
  - 编写代码测试按键输入

MPLAB ICD 2 套件(DV164006)

- 套件包含:
  - MPLAB ICD 2
  - 所有电缆包括USB电缆
  - 电源
  - CD-ROM
  - 资料
  - PICDEM 2 Plus 演示板及:
    - 样品 (PIC18F452, PIC16F877)
    - 功能! (LCD, 温度传感器, 等)

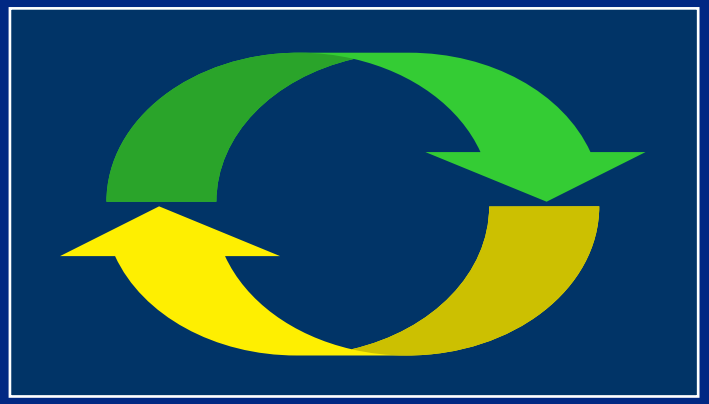
PICDEM 2 Plus 演示板

- 可单独购买
- DV164006 ICD 2套件内含
- 与 ICD 2直接相连
- ICD 2供电或外接电源
- 可演示许多功能

- 注意订单时订购DV164006!
  - DV164007 不包含 PICDEM 2 Plus 和 样品
  - DV164005 仅包含ICD2,无 电缆, 电源,样品等.

跳转

- 循环种类:
  - 定时
  - 嵌套
  - 计数
  - 延时



- 指令:
  - BTFSS　Bit Test File Skip if Set
  - BTFSC　Bit Test File Skip if Clear
  - DECFSZ　DECrement File Skip when Zero
  - INCFSZ　INCrement File Skip when Zero

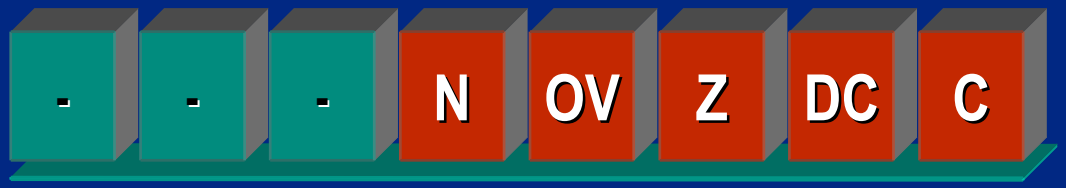
跳转

```
Delay                                ; Delay code using decfsz instruction
        decfsz      MyVar,F          ; Decrement MyVar, skip if 0
        goto        Delay            ; since MyVar is not 0, go back
        bsf         PORTB, RB0       ; MyVar now 0, light LED and continue
```

MyVar

- 指令
  - BTFSS　Bit Test File Skip if Set
  - BTFSC　Bit Test File Skip if Clear
  - DECFSZ　DECrement File Skip when Zero
  - INCFSZ　INCrement File Skip when Zero

跳转指令



STATUS

- Status Bits
  - N Negative
  - OV Overflow
  - Z Zero
  - DC Digit Carry
  - C Carry

- 使用跳转指令测试任何一个位：
  - SFR
  - 变量 (GPR)
  - STATUS 寄存器

Lab 2: 内容

- 测试按键输入
  - 读PORTx,y
    - “x” 是口的名字(例PORTA)
    - “y” 是第几位(0 到 7)
- 程序跳转
  - 通过btfsc or btfss 跳转
    - 指令跳过 或 或不跳过 下一条指令
- 根据跳转点亮 LED
- 用MPLAB® SIM, MPLAB® ICD2 和 PICDEM™ 2 Plus验证

LAB 2

- 测试按键输入

```
#INCLUDE p16F877A.inc          ; Include the standard definitions

clrf    PORTB                  ; Initialize PORTB so that all LEDs are off by default
BANKSEL TRISB                  ; BANKSEL is used to get to bank with TRISB in it (bank 1)
clrf    TRISB                  ; Clear TRISB to make all PORTB pins outputs
movlw   B'00010000'            ; Make pin RA4 an input (rest of PORTA pins are outputs)
movwf   TRISA                  ; This is redundant as TRISA bits are 1's by default (inputs)
                               ; (the rest of the pins are set low)

movlw   H'07'                  ; Writing 7 into ADCON1 causes all analog pins to allow
movwf   ADCON1                 ; digital input.  You must set ADCON1 when using pins with
                               ; A/D functions as they are analog mode by default.
                               ; This code isn't needed in this example - do you know why?
```

LAB 2

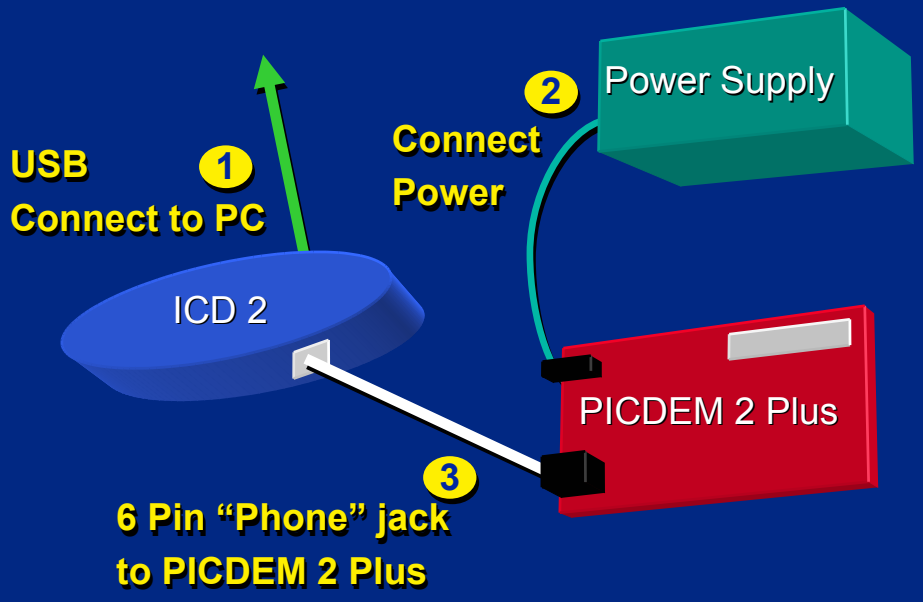
```
        BANKSEL     PORTA       ; BANKSEL is used to return to bank 0 (for PORTA, PORTB)
Loop    btfss       PORTA,4     ; Test RA4 on PORTA, skip if pin is high
        goto        On          ; go to "On" code if button is pressed (pressed button is low)

Off     movlw       B'00000010' ; "On" code is skipped so "Off" code runs - it writes a value to
        movwf       PORTB       ; PORTB to turn on RB1 (high) and sets rest of PORTB low
        goto        Loop        ; The test of the button happens again by branching to "Loop"

On      movlw       B'00001000' ; "On" code writes a value to PORTB to turn on RB3 (high)
        movwf       PORTB       ; and sets rest of PORTB low, including RB1.
        goto        Loop        ; The test of the button happens again by branching to "Loop"

Here    goto        Here        ; This code is not needed either.  Why?
        END                     ; All programs must have an END directive
```

Lab 2:
开始

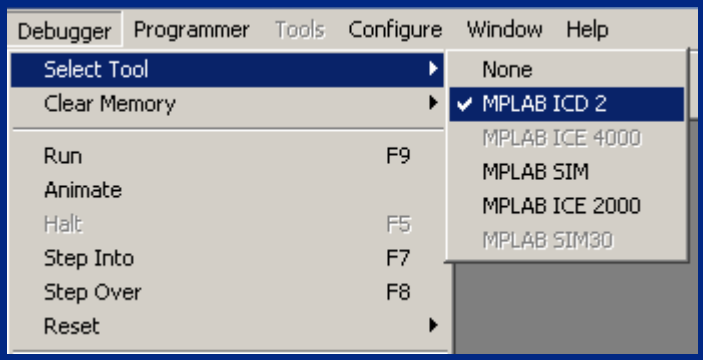
- 打开 Lab2.asm
  - 使用 File -> Open
- 创建项目
  - 使用 Project Wizard

Lab 2:
设置 MPLAB ICD 2

- ICD 2与计算机相连，目标板上电,然后ICD 2 与目标板相连



- 使能 MPLAB ICD 2

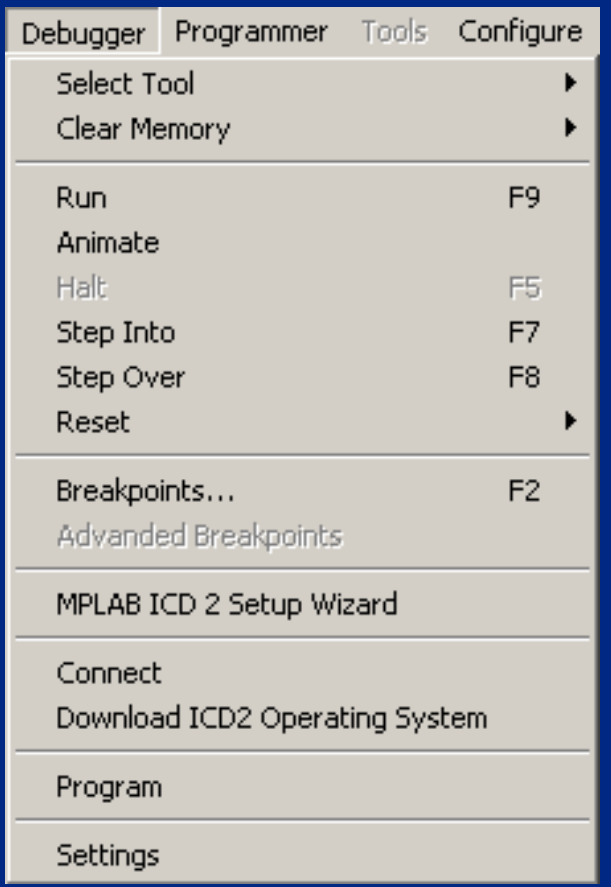


Lab 2
用 MPLAB ICD 2调试

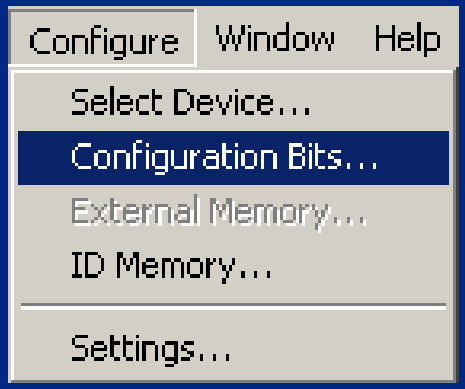
- 使能ICD2, 编译代码.
- 你必须先编程*器件*
- 编程后, 可尝试全速运行,单步走.
- 查看 watch 窗口.

Lab 2
使能 MPLAB ICD 2

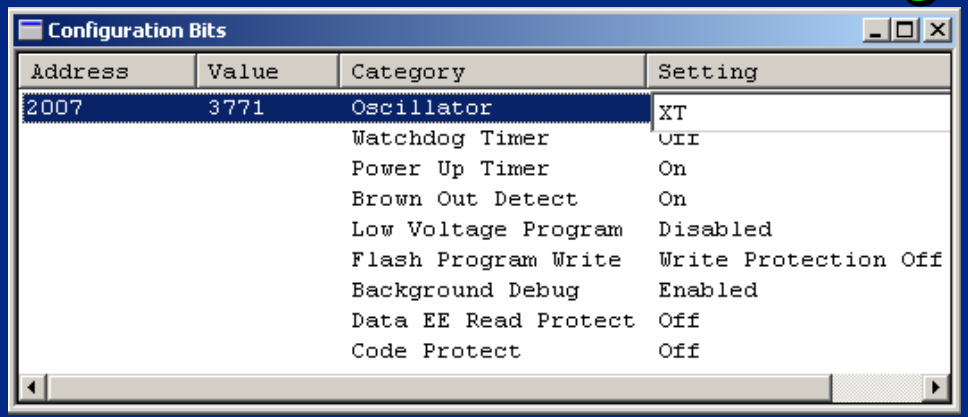
- 一旦使能, 许多选项将允许.
- 你可编程或调试
- 记住设置Configuration Bits!



Lab 2
配置位



- PIC® MCU有 Configuration Bits
- Configuration bits 在编程或调试时被设置.
- 打开Configure -> Configuration Bits, 先设置后编程.
- Configuration Bits 非常重要!



用__CONFIG设置配置位

- _ _CONFIG
  - CONFIG字键入两个下划线
  - 使用INCLUDE(*.inc)
  - 用来设置*CONFIGURATION 寄存器*

用__CONFIG设置配置位

- 例:(PIC16F877A):
  - __CONFIG _XT_OSC & _BODEN_OFF & ....

C:\Program Files\MPLAB IDE\MCHIP_Tools\P16F877A.INC

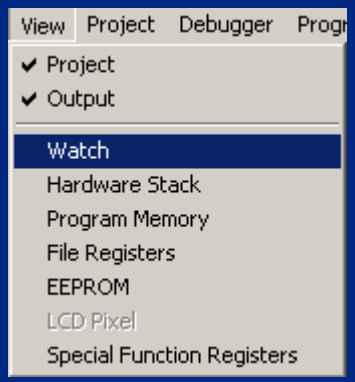
```
;==========================================================
;
;       Configuration Bits
;
;==========================================================

_CP_ALL                         EQU     H'1FFF'
_CP_OFF                         EQU     H'3FFF'
_DEBUG_OFF                      EQU     H'3FFF'
_DEBUG_ON                       EQU     H'37FF'
_WRT_OFF                        EQU     H'3FFF'
_WRT_256                        EQU     H'3DFF'
_WRT_1FOURTH                    EQU     H'3BFF'
_WRT_HALF                       EQU     H'39FF'
_CPD_OFF                        EQU     H'3FFF'
_CPD_ON                         EQU     H'3EFF'
_LVP_ON                         EQU     H'3FFF'
_LVP_OFF                        EQU     H'3F7F'
_BODEN_ON                       EQU     H'3FFF'
_BODEN_OFF                      EQU     H'3FBF'
_PWRTE_OFF                      EQU     H'3FFF'
_PWRTE_ON                       EQU     H'3FF7'
_WDT_ON                         EQU     H'3FFF'
_WDT_OFF                        EQU     H'3FFB'
_RC_OSC                         EQU     H'3FFF'
_HS_OSC                         EQU     H'3FFE'
_XT_OSC                         EQU     H'3FFD'
_LP_OSC                         EQU     H'3FFC'
```

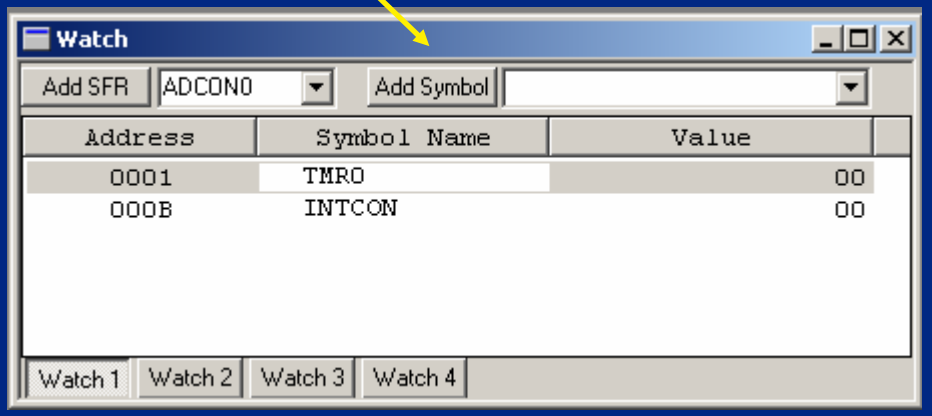
Bitwise AND (&) together any desired terms
(may grow long on some parts)

Open the Processor Include file (*.INC) and scroll to end to see the configuration bit options

使用 Watch Window

- View -> Watch
  - 键入观察的 寄存器/变量
  - 点击“Add SFR” 或 “Add Symbol”





用MPLAB IDE调试

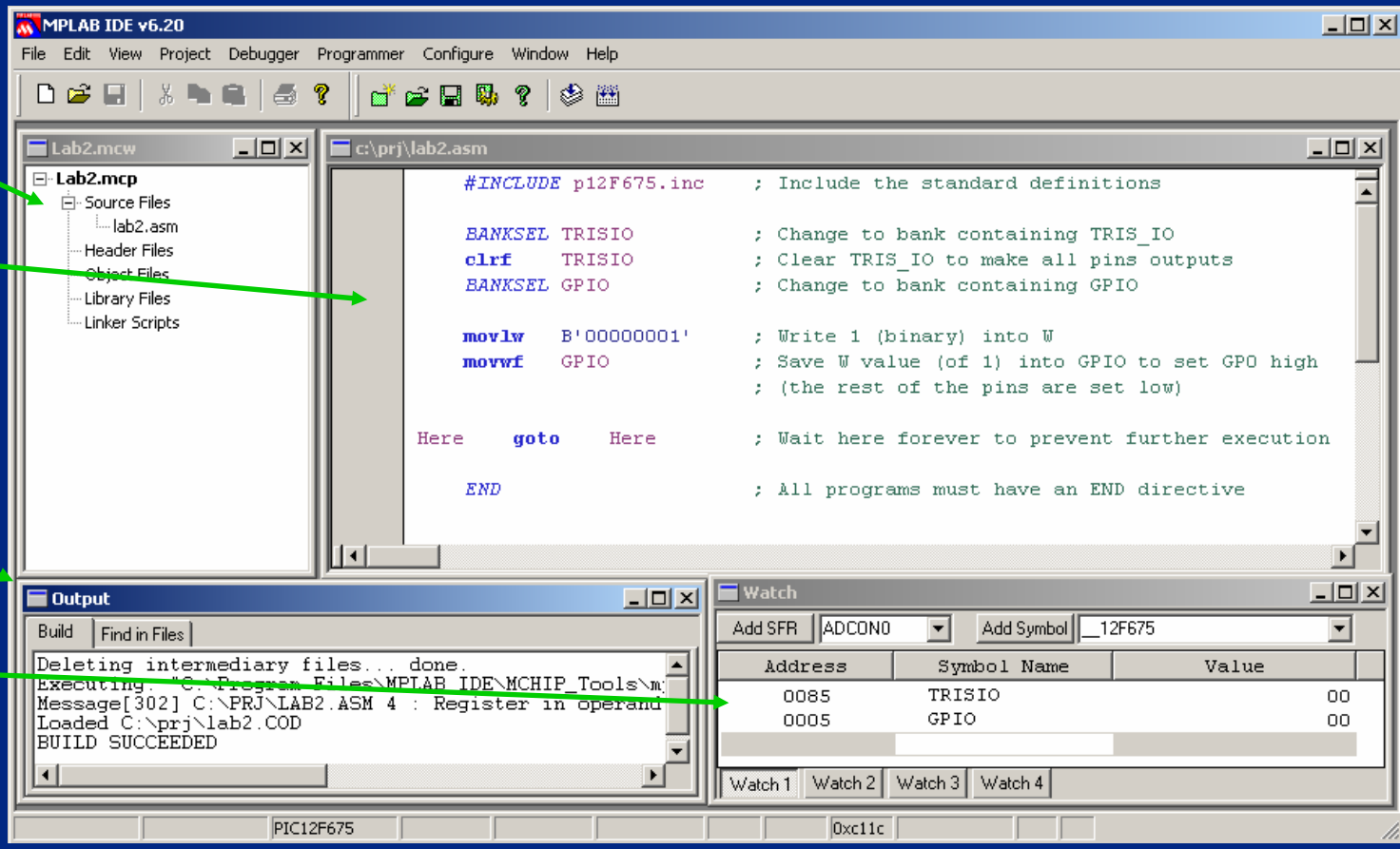
- MPLAB IDE 界面

工程项目窗口

源代码窗口

输出信息窗口

变量观察窗口



MPLAB IDE
快捷键

- -调试快捷键
  - F5 Halt - 暂停
  - F6 Reset - 复位
  - F7 Step - 单步运行
  - F9 Run - 全速运行

- 或使用 图标栏!

Lab 2 – 注意

- MPLAB IDE功能强大, 下列免费
  - 编辑器
  - 编译器
  - 模拟器
  - 与其它工具的接口
- 使用 Watch Windows 调试你的代码
- 编程前先设置 configuration bits
- 跳转指令
  - btfss, btfscc, incfsz, decfsz

MICROCHIP MASTERs

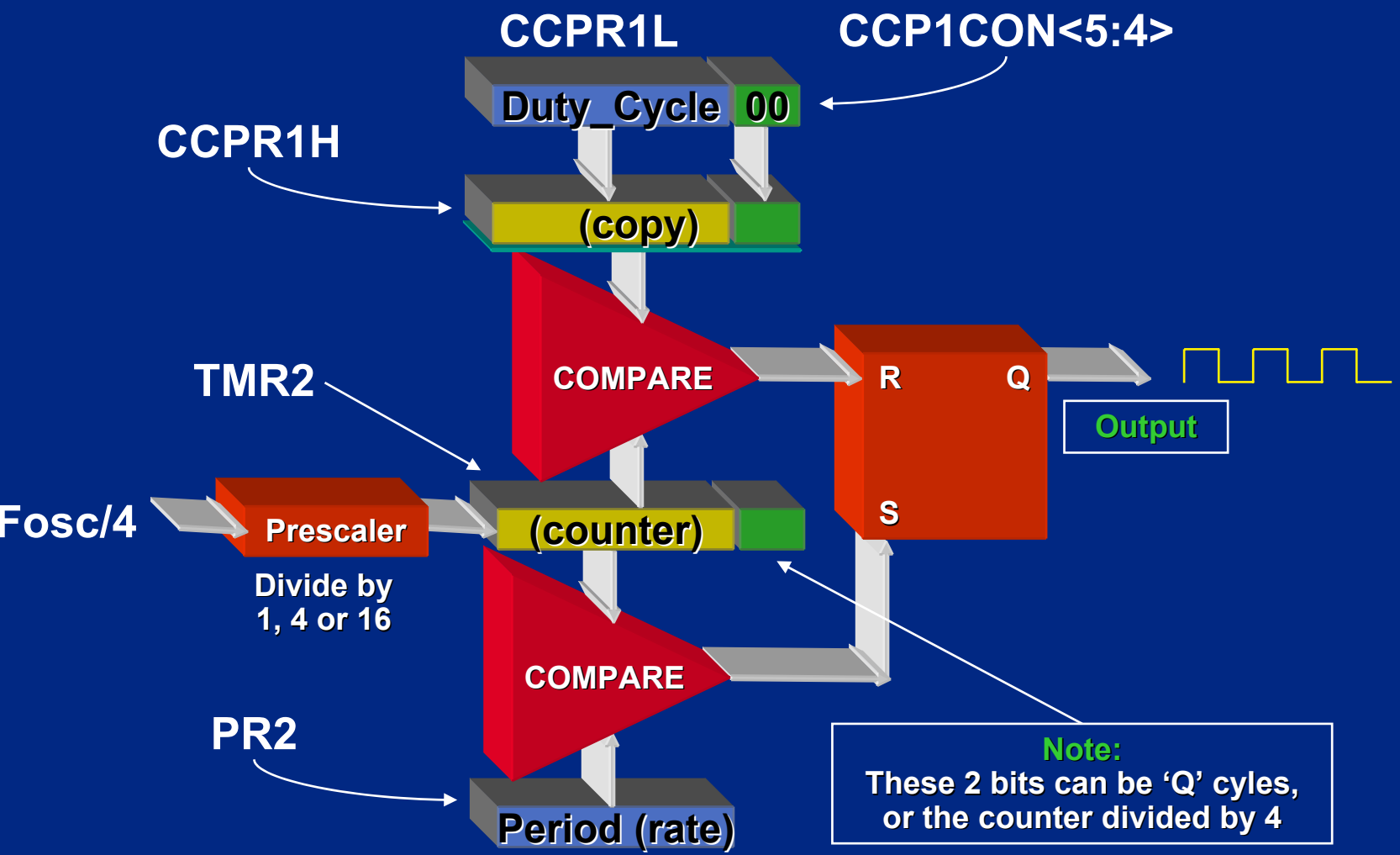
Lab 3

使用 CCP 模块

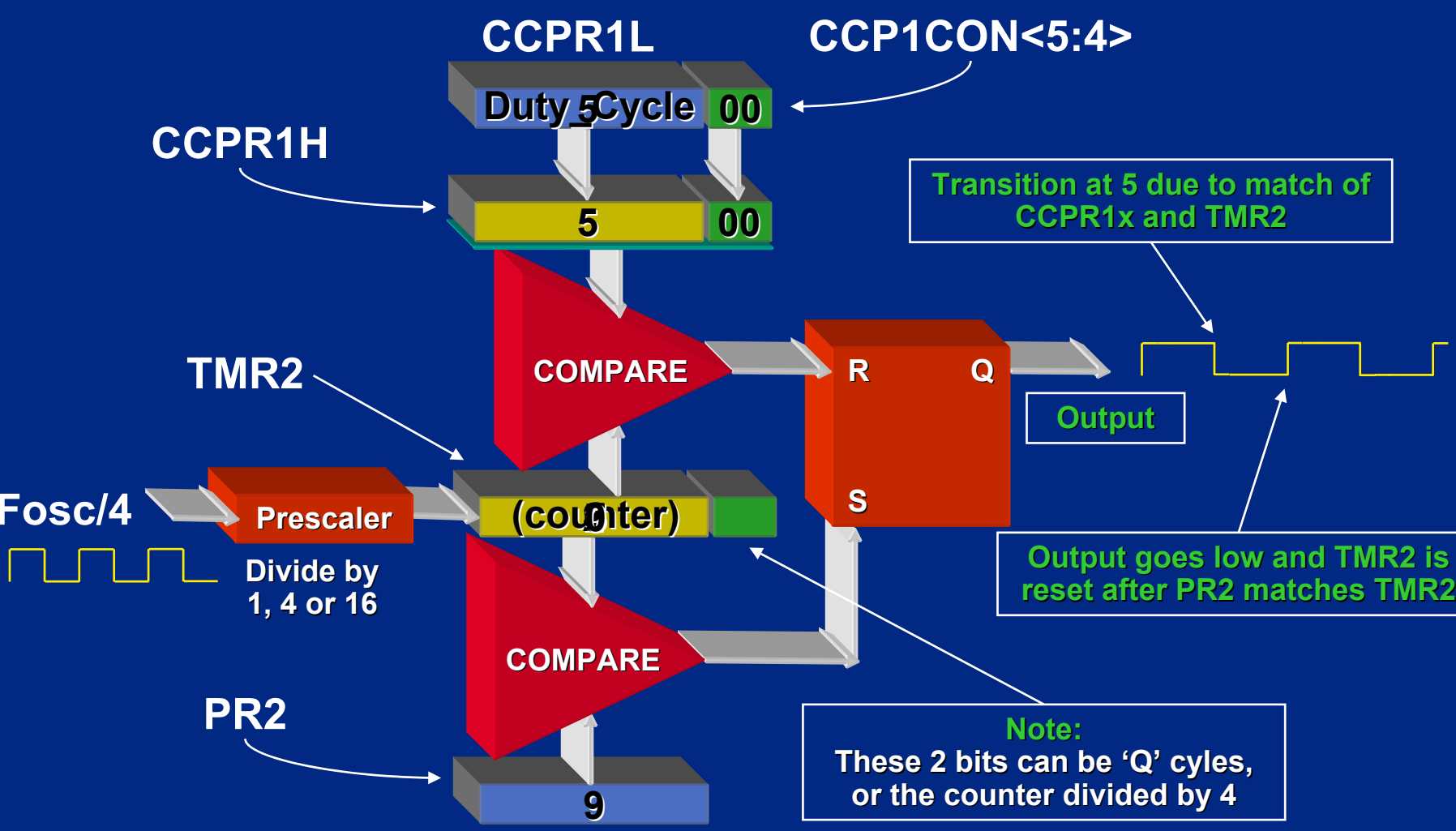
Lab 3

- Lab 3 目标
  - 介绍 CCP 和PWM
  - 让蜂鸣器鸣叫

Lab 3: CCP


CCPR1L
CCP1CON<5:4>
Duty_Cycle
00
CCPR1H
(copy)
COMPARE
TMR2
R
Q
Output
S
Fosc/4
Prescaler
(counter)
Divide by
1, 4 or 16
COMPARE
PR2
Period (rate)
Note:
These 2 bits can be 'Q' cyles,
or the counter divided by 4

Lab 3: CCP


CCPR1L
CCP1CON<5:4>
Duty_Cycle
00
CCPR1H
5
00
Transition at 5 due to match of CCPR1x and TMR2
TMR2
COMPARE
R
Q
Output
Fosc/4
Prescaler
(counter)
0
S
Divide by 1, 4 or 16
Output goes low and TMR2 is reset after PR2 matches TMR2
COMPARE
PR2
9
Note:
These 2 bits can be 'Q' cyles, or the counter divided by 4

Lab 3: CCP 预分频

- 使用TMR2 预分频

- 例:
  - Fosc = 4 MHz
  - Fcy = 1 MHz
  - Tcy = 1 us

T1CKPS1:0	Divides by	Count Rate
00	1	1.00 us
01	4	4.00 us
1x	16	16.00 us

Lab 3:
计算CCP 值

- 例: 880 Hz(4 MHz 的主晶振)

- 选择TMR2 预分频值 16
  - 计算 PR2
    - (4 MHz / 16) / 4 = 62500 Hz
    - 62500 Hz / 880 = 71.022727 - 1 = 70
  - 计算 CCPR1L :
    - 50% 的占空比 是35.

Lab 3:
问题

- 为何选择 16 作为预分频?
- 我们提前计算:
- 回忆 PR2 的结果(70)
- 如果预分频选择4,
  - 计算 PR2 :
    - (4 MHz / 4) / 4 = 250000 Hz
  - 250000 Hz / 880 = 284.09 = 284 = >255
- 结果比之前的预分频值更糟

Lab 3: 内容

- 设置PWM为 880Hz, 50% 占空比
  - 这个频率可听得见, 计算:
    - *PR2*
    - *CCPR1L*
- 设置 T2CON
  - *TMR2 用于 PWM*
  - *选择TMR2 预分频*
- RA4 按下鸣叫
  - *在LAB 3基础上基于 RA4 输入进行跳转*
  - *设置CCP1CON 打开或 关闭PWM*
- 用MPLAB SIM, MPLAB ICD2 , PICDEM 2 Plus验证调试

LAB 3

- 让PIC16F877A发出音乐

```
        #INCLUDE p16F877A.inc           ; Include the standard definitions

Start
        BANKSEL     TRISC               ; BANKSEL to get to TRISC bank (bank 1)
        bcf         TRISC,2             ; Set CCP1 as an output
        movlw       D'70'               ; Manually Calculated Period
        movwf       PR2                 ;  loaded into PR2 (also in bank1)
        BANKSEL     CCPR1L              ; BANKSEL is needed to get bank to bank 0.
        movlw       D'35'               ; Manually Calculated Duty Cycle (50%)
        movwf       CCPR1L              ;  loaded into CCPR1L
        movlw       0x06                ; Turn on TMR2 (for PWM)
        movwf       T2CON               ; and /16 prescale
```

LAB 3

```
CheckButton
        btfsc   PORTA,4         ; Test the button - if pressed, its a low signal.
        goto    ButtonOff       ; if high (not pressed), go to “Button Off” code

ButtonOn
        movlw   H'0C’           ; if low (pressed), fall here to “Button On” code.
        movwf   CCP1CON         ; this turns on CCP1CON by writing 0C (hex) to it.
        goto    CheckButton     ; go check the button again (back to CheckButton)

ButtonOff
        clrf    CCP1CON         ; Turn off CCP1 module by writing 0 to it
        goto    CheckButton     ; go check the button again (back to CheckButton)
        END                     ; All programs must have an END directive
```

Lab 3 – 提高

- 尝试一下!
  - 尝试改变频率. 能改变成440 Hz?
  - 计算:
    - Timer 2 预分频值(1,4 or 16)
    - PR2 值, 然后减 1
    - CCPR1L 值
  - 测试:
    - 使用 'ICD 2 and F7, F5进行调试.

MICROCHIP MASTERs

休息 (II)

Lab 4

使用 A/D Converter (PIC16F877A)

Lab 4目标

- Lab 4 目标
  - A/D Converter 在 PIC16F877A执行A/D转换
  - 基于模拟输入点亮 LEDs
  - 学习使用宏命令

A/D

- 使用 A/D
  - 设置口方向(TRISx)
  - 设置模拟和数字通道(ADCON1 或 ANSEL)
  - 设置A/D 时钟 (ADCON0 / ADCON1)
  - 执行转换循环

A/D

- 设置 A/D
  - 时钟 (内部RC 或 Fosc) — ADCON0 / ADCON1
  - 通道 (AN0, AN7) — ADCON0
  - 打开A/D (ADON 置1) — ADCON0
  - 设置参考电压 (V_{DD} 或外部.) — ADCON1
  - 选择对齐 — ADCON1

设置 TRISx

- 设置口方向
  - 通过TRISA, TRISE设置为输入或输出.
  - 相关的模拟口必须为输入.

A/D

- 设置模拟和数字通道
  - 必须设置口为数字或模拟口
  - 通过ADCON1和ANSEL来设置

This chart is a sample of modes, there are 15 modes for this part

Bit Setting	Channel Mode Options								Voltage References		
PCFG<3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	Vref+	Vref-	C/R
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	Vref+	A	A	A	AN3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0

We will be using mode 1110 (1 A/D channel, VDD and Vss reference)

1110	D	D	D	D	D	D	D	A	Vdd	Vss	1/0

A/D

- 选择 参考电源
  - 参考电源可为 VDD Vss 或外接
  - 当设置数字和模拟引脚模式时同时设置

Bit Setting	Channel Mode Options								Voltage References		
PCFG<3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	Vref+	Vref-	C/R
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0

高参考源

低参考源

A/D 通道 / 参考输入

A/D

- 设置 A/D 时钟
- 两个主要选项- 内部RC 和 Fosc
  - Fosc 有许多分频选项
  - 内部RC 是独立的 A/D 振荡器
  - ADCON0 (有时 ADCON1) 设置 A/D 时钟

A/D

- A/D 时钟计算
  - 假设 4 MHz 主频, PIC18F877A
    - 则Tosc = 250 ns
  - T_{AD} 必须 1.6 us 到20 us

250 ns / 1.6 us = 1/6.4

因此选择Fosc/8

Bit Setting		Clock Mode
ADCON1 <ADCS2>	ADCON0 <ADCS1:0>	
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
x	11	FRC (internal RC)

A/D

- 选择对齐
  - 结果是10 位宽
  - 结果存于: ADRESH ,ADRESL

左对齐

右对齐

- 打开 A/D
  - ADCON0的ADON位置1

A/D

- 循环转换
  - 插入取样时间 (你的代码)
  - 开始转换 (GO/DONE置1)
  - 等待GO/DONE 清零 (或ADIF=1)
  - 读A/D 值(ADRESH / ADRESL)

宏命令

- 宏命令
- 用MACRO开始 ,用ENDM结束

```
Name_Of_Macro      MACRO    Variables
nop ;My Code Goes Here
                   ENDM
```

宏命令

- 宏命令

例:

```
AddW    MACRO   Input
addlw   Input
        ENDM
```

你的代码中:

```
AddW    5       ; Add 5 to W
```

Lab 4: 内容

- 读 A/D值,用 ADRESH 值点亮 LEDs
  - 当值小于 100
    - 点亮 LED 0 和关闭 LED 1
  - 当值大于 100
    - 点亮 LED 1 和关闭 LED 0
- 别忘了取样时间
  - GO/DONE 置1前必须延时, 1 ms足够
  - 例: Dly32 D’50’ 延时1 ms

Lab 4:宏命令

- 可使用下列宏命令:
  - **LEDEnable <*bitpattern*>**
    - 1 输入, 0 输出.
    - 允许LED 0 和 1, 使用:
      - LEDEnable B’11111100’ or LEDEnable H’FC’
  - **LEDOn <*LED number*>**
    - *<LED number>必须在0和3之间.*
    - 例, 点亮 LED 0
      - LEDOn 0

Lab 4:宏命令

- **LEDOff** **<*LED number*>**
  - LEDOff 关闭 LED
  - 例
    - LEDOff 2

- **Dly32** **<*Cycles x 20*>**
  - 延时子程序
  - 例延时 1 ms (1000 cycles at 4 MHz) :
    - Dly32 D'50' *; Delay of 50 x 20 cycles = 1000 Tcy*

LAB 4

```
        #INCLUDE p16F877A.inc           ; Include the standard definitions
        #INCLUDE 701PIC.inc             ; Include MACROs from MASTER's 03, PIC 102 class
        #DEFINE     TESTVAL  D'100'     ; High Value to compare A/D reading against
Start
        LEDEnable   B'11111100'         ; MACRO to setup I/O port to drive LEDs RB1 and RB0
        BANKSEL     TRISA               ; BANKSEL to TRISA in bank 1
        movlw       H'01'               ; Initialize PORTA so that RA0 (analog channel 0) is an input,
        movwf       TRISA               ; all other PORTA pins are outputs
        movlw       H'0E'               ; Initialize the A/D to be analog on, partly selects clock mode
        movwf       ADCON1              ;  channel 0 (RA0), all other pins are digital mode
        BANKSEL     ADCON0              ; BANKSEL to ADCON0 in bank 0
        movlw       H'41'               ; Also, A/D clock is Fosc/8 (good for up to 5 MHz, 5V) and,
        movwf       ADCON0              ; A/D is turned on but not yet converting and ready on Ch 0.

CheckAgain
        Dly32       D'50'               ; MACRO to Wait 1 ms (at 4 MHz)
        bsf         ADCON0,GO           ; Start the Conversion
```

LAB 4

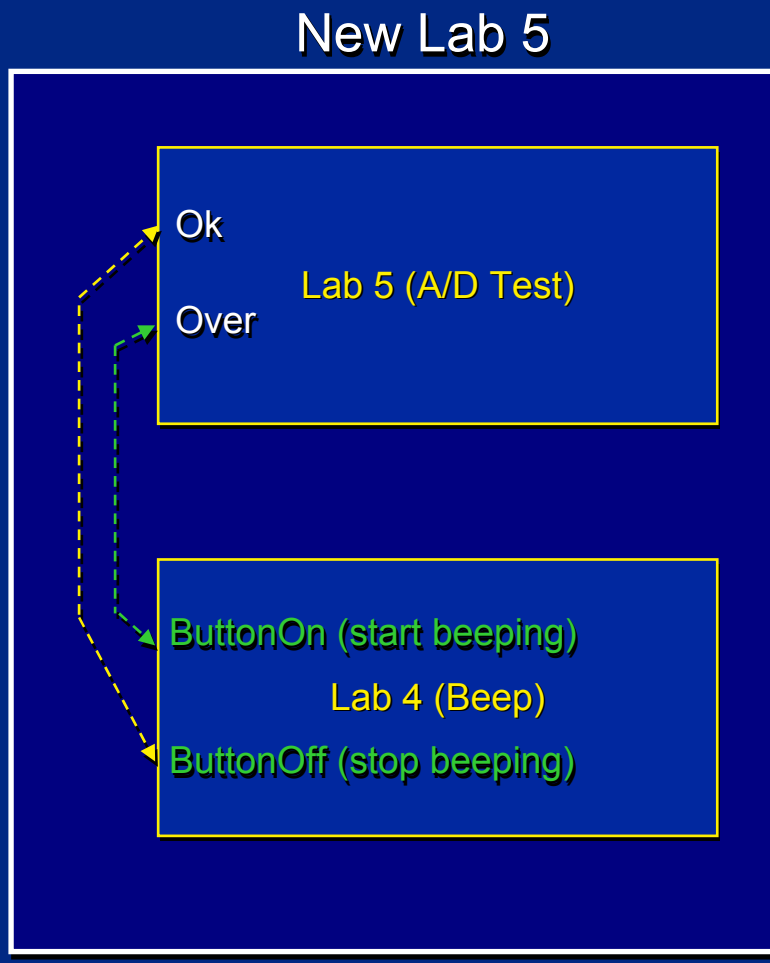
```
PollAD
        btfsc       ADCON0,GO       ; Wait for Conversion to finish
        goto        PollAD          ; go back and poll the A/D GO/DONE bit again
ConvertDone
        movf        ADRESH,W        ; Move A/D result (high byte) into W for later testing
        sublw       TESTVAL         ; Test A/D result against test value
        btfss       STATUS,C        ; go to “over” if A/D result larger than test value
        goto        Over            ; value is too large, run “Over” code
Ok
        LEDOn       0               ; MACRO to turn on LED 0
        LEDOff      1               ; MACRO to turn off LED 1 (this shows “Ok” condition)
        goto        CheckAgain      ; Do A/D read and value test over again
Over
        LEDOn       1               ; MACRO to turn on LED 1
        LEDOff      0               ; MACRO to turn off LED 0 (this shows “Over” condition)
        goto        CheckAgain      ; Do A/D read and value test over again
        END
```

Lab 4 – 警示灯

- 试一试!
  - 调节电位器RA0. LEDs 改变吗?

Lab 4b – 报警!

- 尝试下列功能:
  - 复制Lab 4 代码粘贴到 Lab 5.
    - 如果值大于 “TESTVAL”鸣叫
    - 使用GOTO or CALL/RETURN 指令



MICROCHIP MASTERs

Lab 5

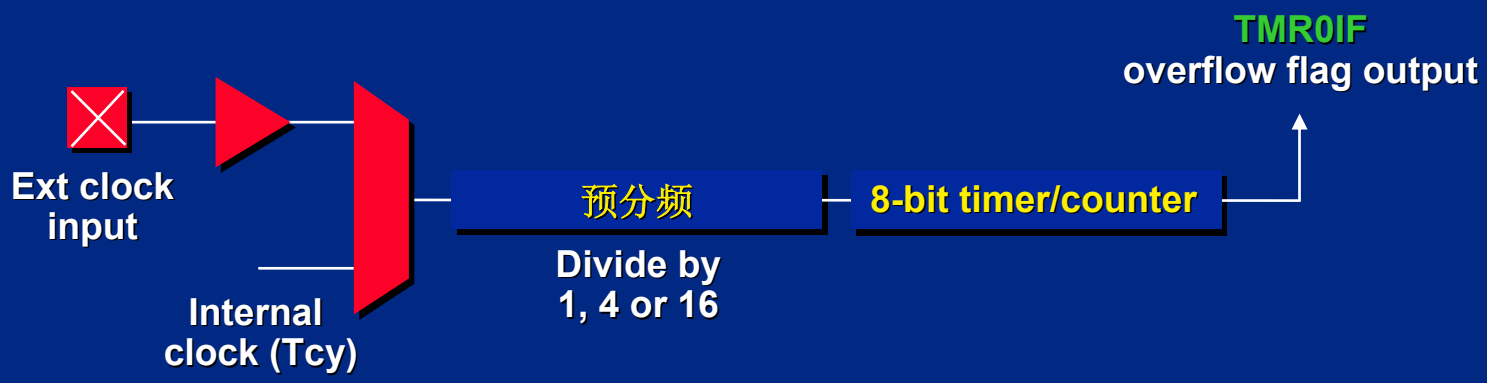
定时器及中断
(on the PIC16F877A)

Lab 5的目标

- Lab 5 的目标
  - 学习TMR0
  - 理解查询和中断
  - 写一个基于中断的延时程序
  - LED 以中断设定的速度闪烁

Lab 5:
TMR0 外设

- 可读写
- 从 FF 到 00 (hex)溢出时产生中断
- 预分频可选择 2, 4, 8, 16... 直到 256
- 定时模式: 最快速度是 OSC/4 (对于 20 MHz 晶振来说是5 MHz)
- 外部事件计数模式: 上升沿或下降沿触发可供选择
- TMR0 位宽是8



MICROCHIP MASTERs

使用 TMR0

- TMR0 时钟源是 Fosc / 4
  - 您的Demo板上使用 4 MHz 时钟 (Fosc)
- 当使用预分频时, 输入可进一步除上 2, 4, 8, 16... 直到 256.
- TMR0 从 00 到 FF (hex)向上计数, 计到FF时溢出从00开始重新计数.
- 溢出时标志位置1以供中断或查询使用.

计算 TMR0 的溢出

- 使用 4 MHz 时钟
- TMR0 计数时钟速率是 Fosc/4
- 对于一 200 ms (0.2 second)速率中断,
  - 您将如何配置 TMR0 的预分频?
  - TMR0 初值置多少?
- 考虑:
  - 计数器以1 us速率计数, 无预分频, 能计到多少?
  - Time = 预分频 x maximum count (8 bits) x rate
  - Time = 256 x 256 x 1 us
  - Time = 65536
  - 但我们需 200000 来得到 0.2 秒!

计算 TMR0 的溢出
(第三部分)

- 将溢出时间作为计数单元
- 本例 50000 us
- 以整数的倍数 (4)计数来得到总时间.
- 这种设定中能计到50000 us吗?
  - 除数是 (256), 时间 (50000) 得到商:
  - 500000 / 256 = 195.3125 counts
- 选择:
  - 忽略相对较小的误差 (195 * 256 = 49920 us)
  - 改变主晶振以得到整数
  - 改变除数, 获得快中断等诸多方法.

计算 TMR0 的溢出
(第三部分)

- 本例中我们采用忽略误差的方法，当然在您的应用中方法会不一样
- 故，这里TMR0是:
  - 256 预分频 速率
  - 195 计数值
  - 4 个溢出

  - 这样，我们需给定时器预装一个初值，定时器从该值开始向上计到256时，给出溢出信号:
    - 这个值是 256 - 195 = 61

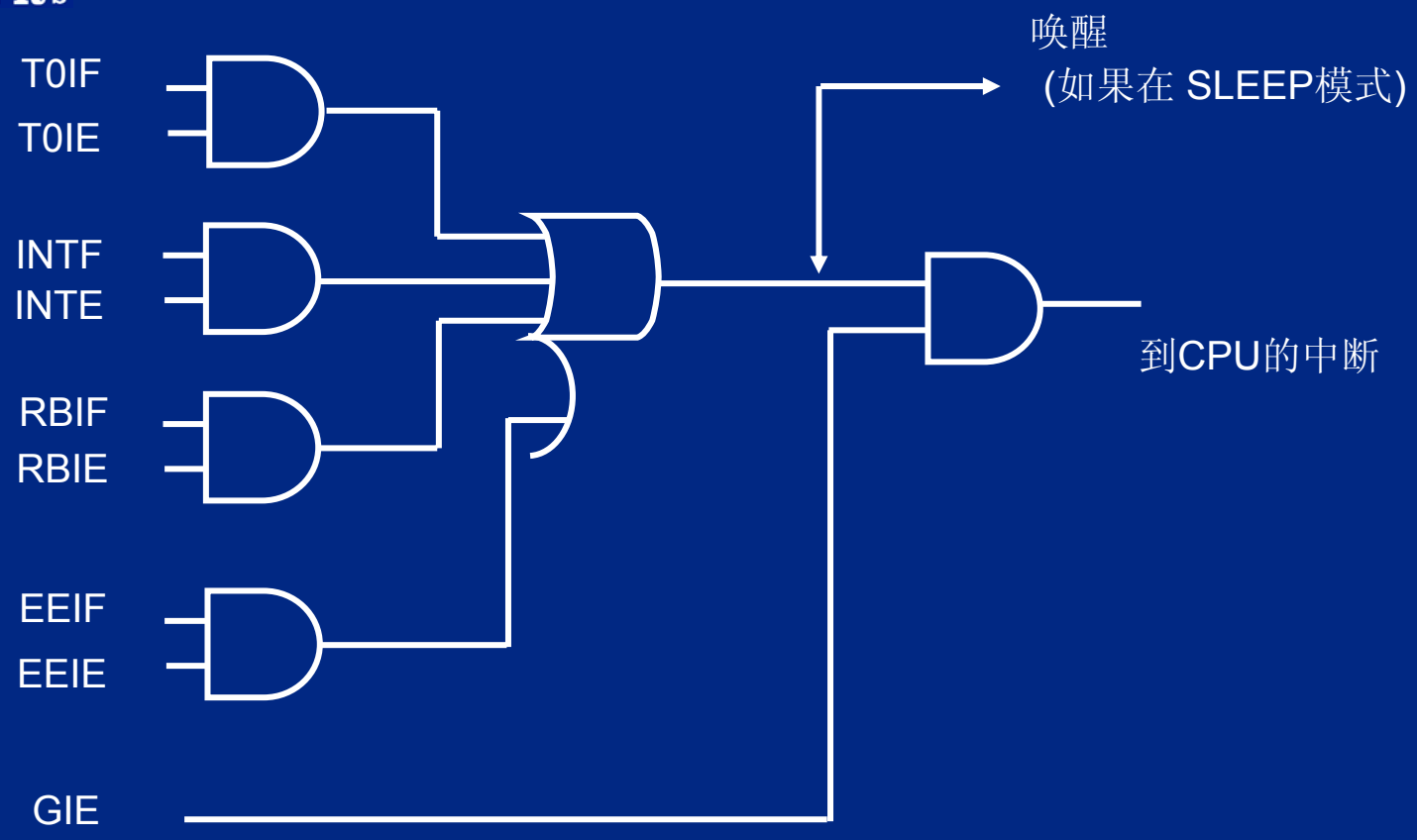
TMR0 中断

- 当定时器溢出时，会将标志位置1
  - 举个例子，TMR0 将TMR0IF置1
- 中断 标志位 xxIF产生标志时，既可用来查询，同时也是中断产生的基础
- 在您的软件中一旦检测到xxIF标志位，必须清零

Lab 5 - 中断

- 中断...
  - 使 CPU在执行主程序时转而执行其他任务
  - 中断源有多种:
    - 定时器
    - 外设
    - 外部信号
  - 配置方式
    - NTCON 寄存器
    - 其他寄存器(随芯片不同而不同 – 详见数据手册)

中断



Note: There is often a “PEIE” bit which enables/disable further 中断 sources

INTCON REG
PIC16F877A

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

R= Readable bit W= Writable bit
U= Unimplemented bit, read as '0'
- n= Value at POR reset

bit 7: **GIE:** Global 中断 Enable bit
1 = Enables all un-masked 中断
0 = Disables all 中断

bit 5: **T0IE** : TMR0 Overflow 中断 Enable bit
1 = Enables the TMR0 中断
0 = Disables the TMR0 中断

bit 2: **T0IF** : TMR0 Overflow 中断 Flag bit
1 = TMR0 has overflowed (must be cleared in software)
0 = TMR0 did not overflow

为什么要用 中断?

- 更好的利用 CPU 资源
- 快速响应
- 多任务
- 固定的或已知的间隔处理
- 异常处理

我需要中断吗?

- 许多程序不需中断
- 逻辑流程中不用中断
- 不需现场保护...

中断:
ISR 现场保护

Saving

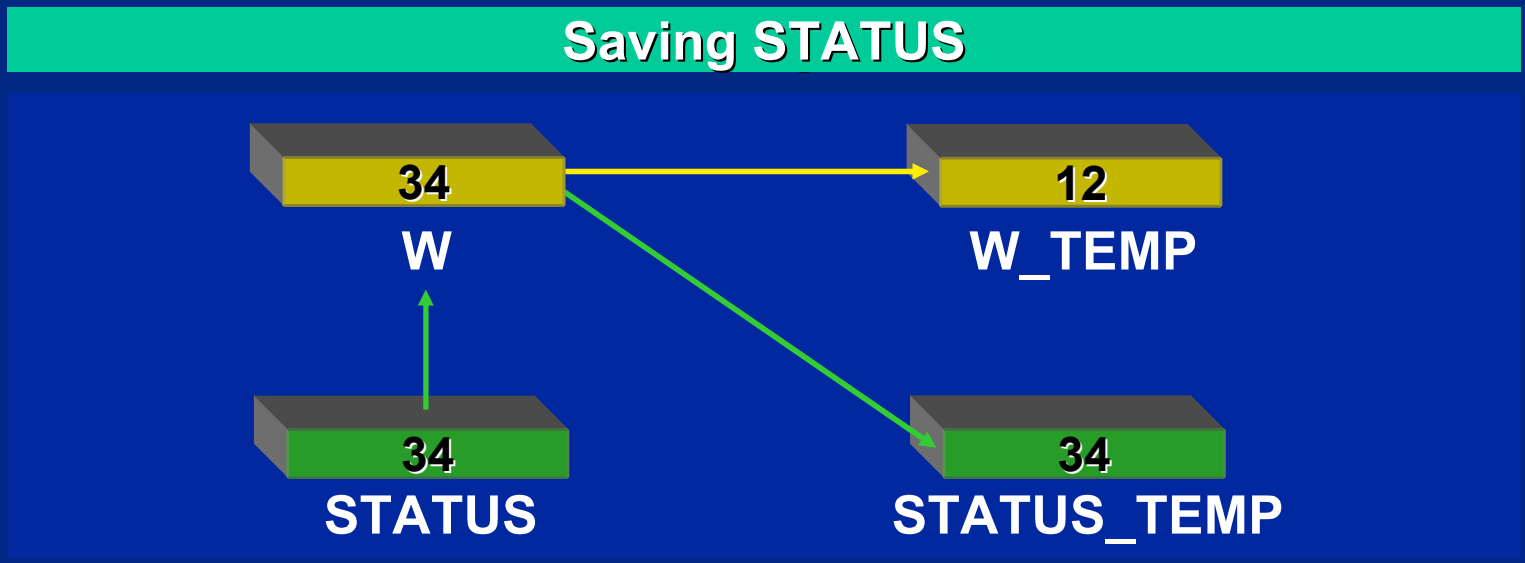
```
Push MACRO                          ; PUSH W and STATUS
    movwf     W_TEMP                ;
    movf      STATUS,W              ; (swapf will also work here)
    movwf     STATUS_TEMP           ;
    ENDM
```

Restoring

```
Pop MACRO                           ; POP W and STATUS
    movf      STATUS_TEMP,W         ; (swapf will also work here)
    movwf     STATUS                ;
    swapf     W_TEMP,F              ; movf changes STATUS bit Z, but
    swapf     W_TEMP,W              ; swapf does not change STATUS
    ENDM
```

中断: ISR 现场保护

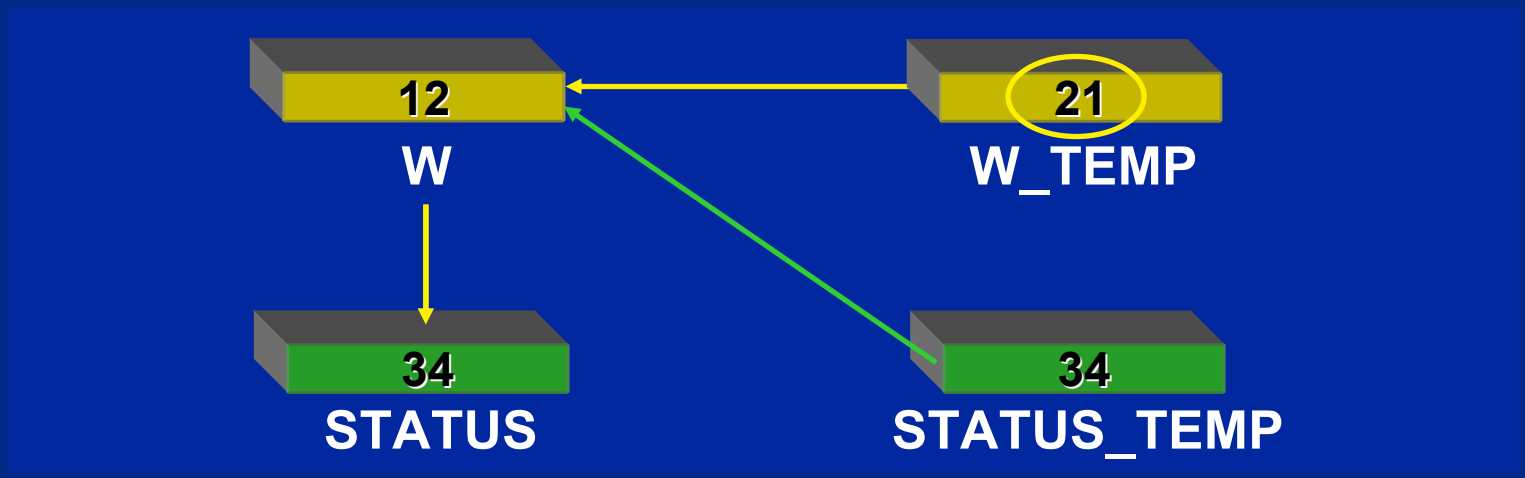
```
Push MACRO                          ; PUSH W and STATUS
                                    ;
        movwf   W_TEMP              ; (swapf will also work here)
        movf    STATUS,W            ;
        movwf   STATUS_TEMP
        ENDM
```



中断:
ISR现场保护

```
Pop MACRO                                ; POP W and STATUS
        movf    STATUS_TEMP,W            ; (swapf will also work here)
        movwf   STATUS                   ;
        swapf   W_TEMP,F                 ; movf changes STATUS bit Z, but
        swapf   W_TEMP,W                 ; swapf does not change STATUS
        ENDM
```

Restoring W



Lab 5: 内容

- TMR0 预分频 256, 每个循环计195 次以产生0.05 秒 (50000 us) 中断
  - 每四个循环 (0.2 seconds) 触发LED 0.
- LED 0 (RB0)以此速率闪烁
  - 在改变LED状态前你要读它的当前状态, 或采用其他方式
  - 对该口写一完整的值,不对该口寄存器使用 bsf/bcf指令
- 使用 MPLAB SIM, MPLAB ICD2 以及 PICDEM 2 Plus验证你的结果

LAB 5第一部分

```
#INCLUDE p16F877A.inc          ; Include the standard definitions
#INCLUDE 701PIC.inc            ; Include MACROs from MASTER's 03, PIC 102 class

ISRCtr      EQU      0x30      ; location for ISRCtr variable (countdown for ISR loops)

        ORG 0                  ; Reset Vector Location (0)
Start
        goto  Initialize       ; On start-up go to Initialization code

        ORG 4                  ; 中断 Vector Location (4)
ISR
                               ; Macro to context save in ISR
                               ; Clear the TMR0 中断 condition
                               ; Reload the Timer for accurate timing
                               ; Decrement the ISRCounter (from 4 to 0) and test for 0
                               ; if the counter is not yet 0, then toggling is not needed
                               ; if the counter is 0, run toggle code
```

LAB 5 第二部分

```
ToggleLED
                    ; put 4 into W to reload into the
                    ; 中断 counter (after 4 counts we need to reset it)

                    ; write 1 to W in order to,
                    ;  toggle LED 0 of PORTB - XOR with 1 will toggle the bit

SkipToggle          ; Done 中断, so restore and exit
                    ; Macro to context restore when leaving ISR
                    ; Return from 中断
```

LAB 5 第三部分

```
Initialize

                        ; Ensure LED 0 is off
                        ; MACRO to setup I/O port to drive LED RB0
                        ; Write 4 to W to
                        ;  preload the ISR counter with
                        ; Preload the timer for appropriate time delay
                        ; Turn on GIE and
                        ; TMR0 中断 (T0IE)
                        ; Set up banking to access OPTION_REG in bank 1
                        ; Set up TMR0 for 256 prescale and
                        ;  internal clock mode
                        ; return to accessing BANK 0

    goto  $             ; Wait forever for 中断
```

LAB 5 第四部分

```
; Subroutine to reload TMR0
DoReloadTimer
                                    ; Preload Timer to
                                    ;  count 195 times before roll-over
                                    ; return from subroutine

        END
```

Lab 5 注意点

- 定时器:
  - PIC 微控制器带预分频可选的至少 8位 (TMR0) 定时器
  - 需要重装初值以得到期望时间
  - 可以通过计几次溢出以得到期望时间.
- 中断:
  - 判断是否是您应用中所要的中断,如果是
    - 不要忘了保护和恢复现场
    - 置GIE, 及其他 xxIE 位来打开中断
    - 中断向量从程序段的0004h开始

MICROCHIP MASTERs

Lab 6

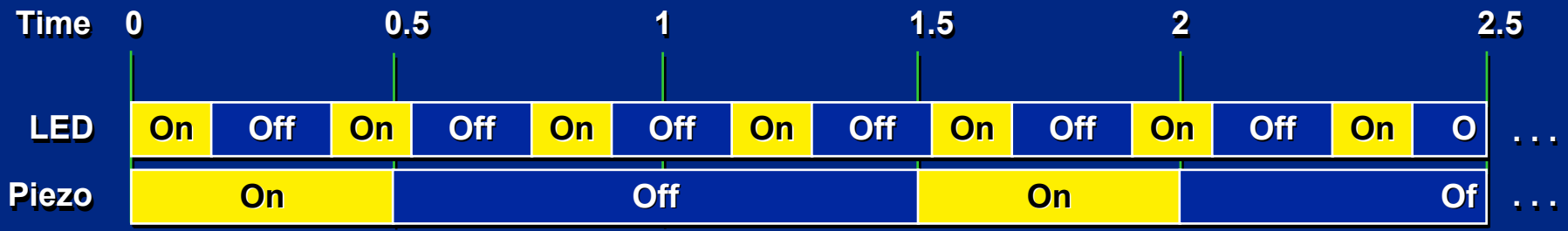
时间片系统
(实时多任务)

Lab6目标

- Lab6目标
  - 简单的时间片系统(Time -Slicing Systems)
  - 代码迅速处理两个 (或以上的) 任务!

Lab6: 任务

- LED 0以0.15 秒点亮, 0.25 秒熄灭来闪烁 ,并且蜂鸣器(及 LED 1闪烁) 以0.5秒打开, 1秒关闭:
  - 让我们看看是什么现象:



Lab6: 计算

- 分时
  - 找出最小的时间单元:
  - 秒数: *0.15, 0.25, 0.5, 1.0*
    - 0.05 秒贯穿所有这些整数时间段 (使用 GCF 来计算)
  - 计算任务数 (及变量): 2
  - 然后生成一计划来显示这两个软件计数器是怎样控制这些任务的.

Note: GCF stands for “Greatest Common Factor”

Lab6: 分时 (Animated)

TMR0 中断 每 0.05 s 在每个计数寄存器中产生一次计数

计数器
C
C
复

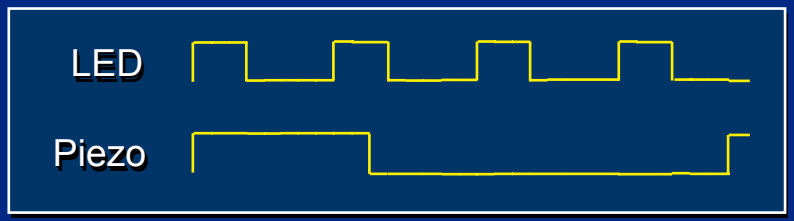
Piezo 计数器
0: ON
10: OFF 0
30: 复位到0

Time passes...

(计数器)
LEDCtr

OFF

ON



Lab6: 计划

- TMR0每0.05秒中断一次以产生时基
  - 该时基用于:
    - 点亮或熄灭 LED 0 及LED 1 (RB0 及 RB1)
    - 打开或关闭 Piezo
    - 清掉 LED 及 Piezo的时间计数器
- 点亮2个 LEDs 并使蜂鸣器发声 (以 500 Hz):
  - RB0 打开时间 0.15 秒, 关闭时间 0.25 秒 (共0.4s)
  - RB1 打开时间 0.5 秒, 关闭时间 1 秒 (共1.5 s)
  - Piezo 响 0.5 秒, 关闭1秒 (同 RB1)
- 使用宏指令来简化代码

Lab6第一部分

```
        #INCLUDE p16F877A.inc           ; Include the standard definitions
        #INCLUDE 701PIC.inc             ; Include MACROs from MASTER's 03, PIC 102 class
        #DEFINE DEVICE_FREQ_HZ D'4000000'  ; Needed by Piezo series macros

LEDCtr  EQU     0x30                    ; Counter for LED timing
PiezoCtr EQU    0x31                    ; Counter for Piezo timing

        ORG     H'00'                   ; Reset Vector
Start
        goto    Initialize              ; Jump to Initialization code

        ORG     H'04'                   ; 中断 Vector
ISR
                                        ; Macro to context save in ISR
                                        ; Clear the TMR0 中断 condition
                                        ; Add 1 to LEDCtr
                                        ; Add 1 to PiezoCtr
                                        ; Reload the Timer for accurate timing
                                        ; Macro to context restore when leaving ISR
                                        ; Return from 中断
```

Lab6第二部分

```
Initialize
        LEDOff    0             ; Turn off both LEDs
        LEDOff    1             ;  "
        clrf      LEDCtr        ; Initialze Task Couters to 0
                                ;  " (both LED and Piezo)
                                ; MACRO to setup I/O port to drive LEDs RB1 and RB0
                                ; 500 Hz Beep when PiezoOn is used
                                ; Preload the timer for appropriate time delay
                                ; set banking to access OPTION_REG in bank 1
                                ; Set up TMR0 for 256 prescale and
                                ;  internal clock mode
                                ; Return to accessing BANK 0

                                ; Turn on GIE and TMR0 中断
                                ;  "
```

Lab6第三部分

```
LEDTest
                        ; Test LED1 Counter Value for overflow
                        ;  by subtracting 8 from it.  Leave result in W
                        ; Test for 0 to determine if equal
                        ; If counter is not 8, test if LED should be on or off
                        ; If counter is 8, it is reset to 0
NotEqual1
                        ; Test LED counter against "On" time
                        ;  by subtracting 3 from it.  Leave result in W
                        ; Test for negative value to determine if less than 3
                        ; if 3 or more, goto LEDOff Code
TurnOnLED
                        ; else, Turn on LED 0 since counter is less than 3
                        ; LED Testing done, goto PiezoTest
TurnOffLED
                        ; Turn off LED since counter is 3 or more (up to 7)
                        ; LED Testing now done, so fall into PiezoTest
```

Lab6第四部分

```
PiezoTest
                                ; Test Piezo Counter Value
                                ;  by subtracting 30 from it.  Leave result in W
                                ; Test for 0 to determine if equal
                                ; If counter is not 30, test if LED should be on or off
                                ; If counter is 30, it is reset to 0
NotEqual2
                                ; Test Piezo Counter against "On" time
                                ;  by subtracting 10 from it.  Leave result in W
                                ; Test for negative value to determine if less than 3
                                ; if 10 or more, goto LEDOff Code
TurnOnPiezo
                                ; Turn on LED1
                                ; Turn on Piezo (by turning on CCP1)
                                ; manage tasks by returning to top of list
TurnOffPiezo
                                ; Turn off LED1
                                ; Turn off Piezo (by turning off CCP1)
        goto    LEDTest         ; manage tasks by returning to top of list
```

Lab6第五部分

```
; Subroutine to reload TMR0
DoReloadTimer
                                        ; Preload Timer to
                                        ; count 195 times before roll-over
                                        ; return from subroutine

            END
```

对您项目的
帮助

MICROCHIP MASTERs

Microchip 应用支持

www.microchip.com

数据手册, 应用笔记, 设计中心

选择最好的元件

申请免费样片

Microchip 应用支持

仍有困难?

免费的电话或**e-mail**帮助!

Development Systems Information Line

800-820-6247

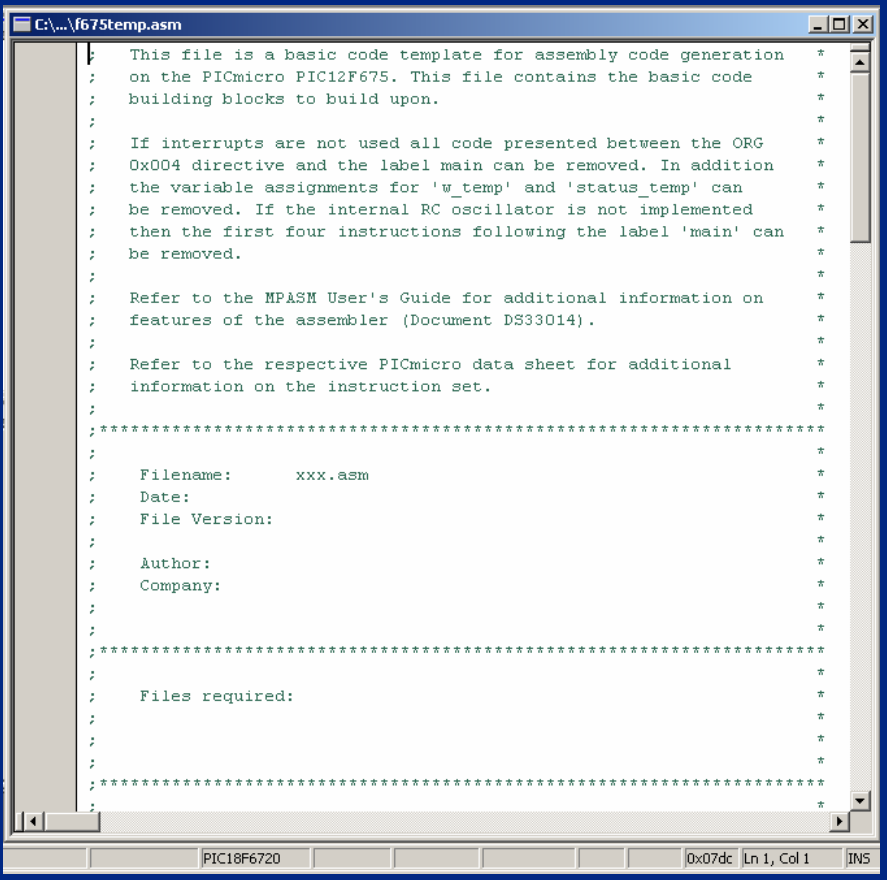
China.techhelp@microchip.com

800-820-6247

Thank You

附录

使用模板来建立文件



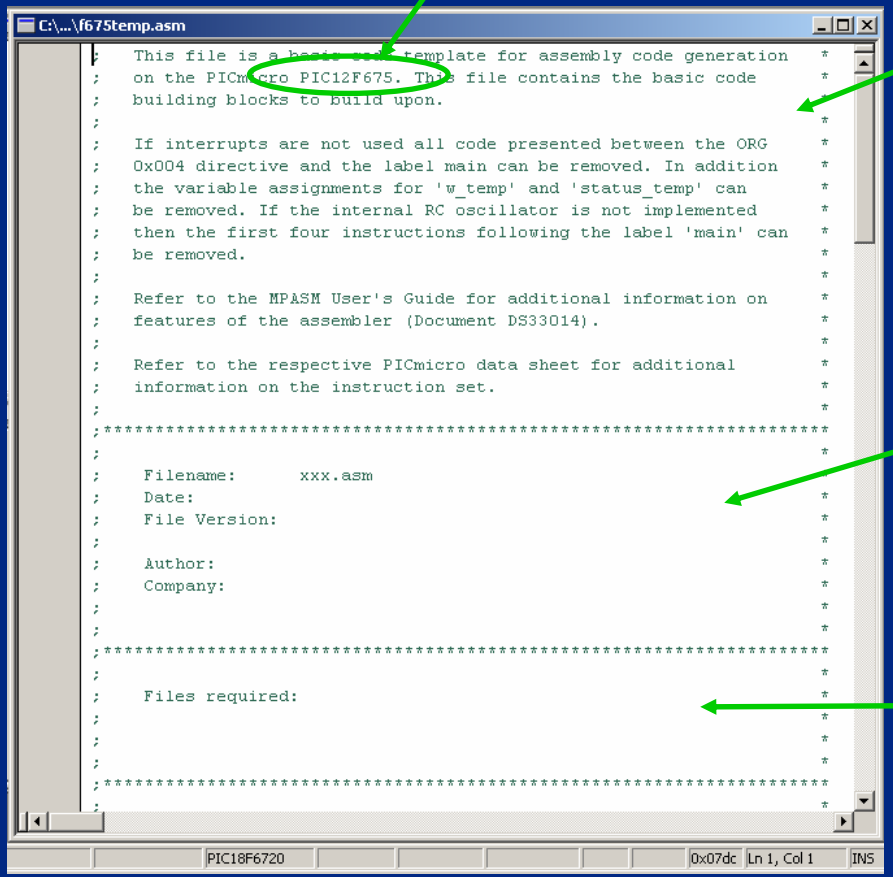
- 模板能帮助您快速建立文件
  - File -> Open
  - 进入模板路径:

C:\Program Files\MPLAB IDE\MCHIP_Tools\模板\CODE

- 找到您所需的文件 – 文件名中带有芯片型号的后几位字符
- 举例如 PIC12F675 或 (‘F675), 文件名 是f675temp.asm

模板中有些什么？

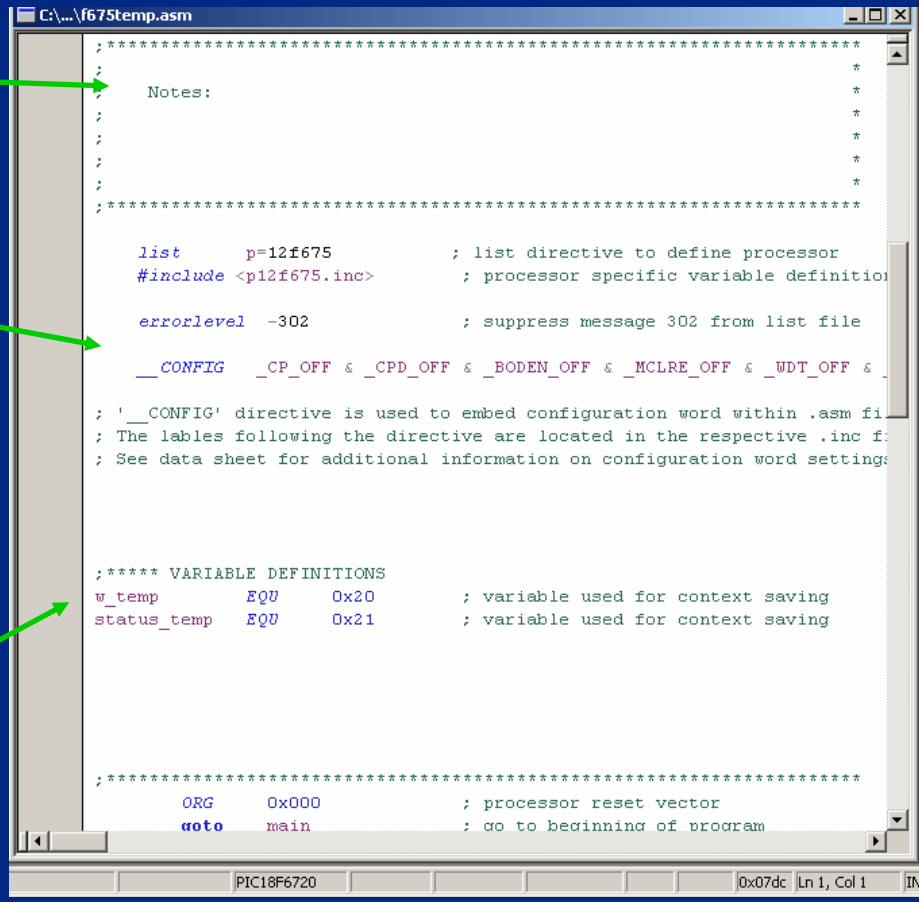
芯片名



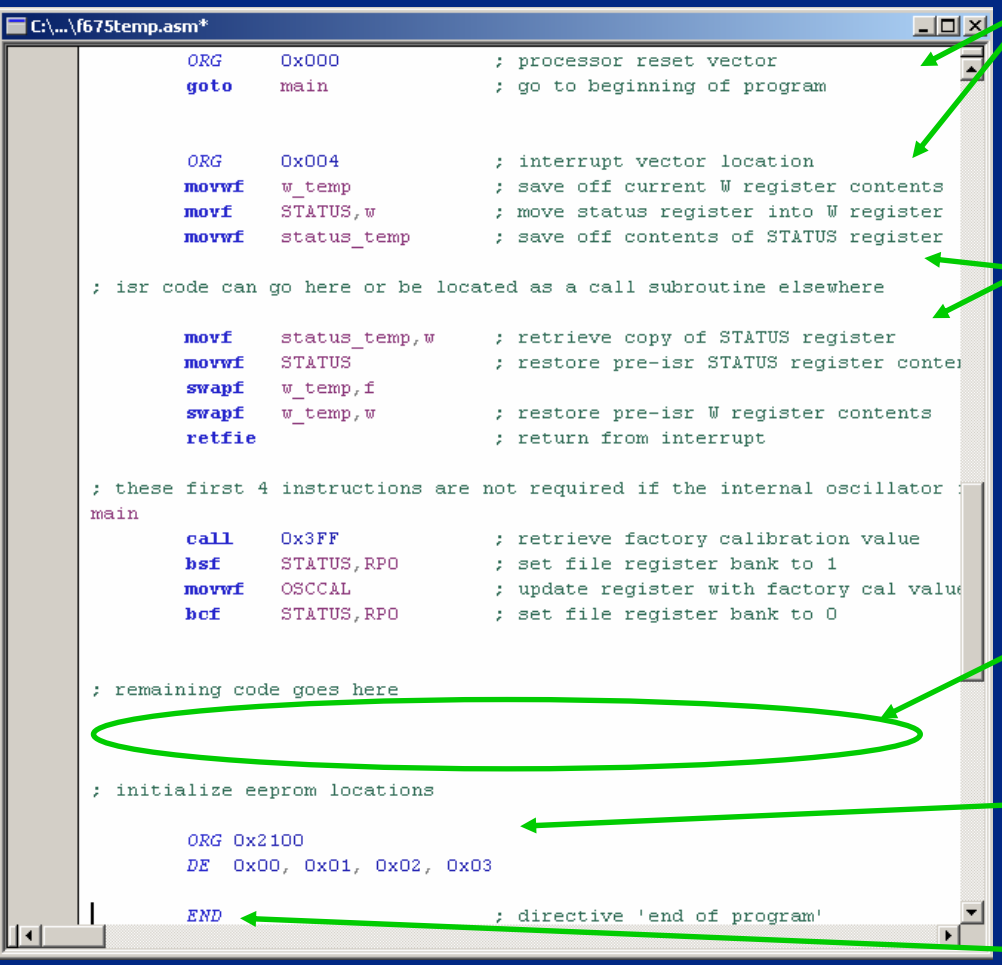
- 说明
  - 代码编写提示
- 文件信息
  - 文件名,时间, 版本等
- 所需的文件
  - 说明项目中所用到的其他文件

模板中有些什么?

- 笔记
  - 解释程序流程
- Include, List 及 __CONFIG 指令
  - Includes 标准定义
  - 设定配置位
- 定义变量
  - EQU 定义变量
  - 中断 中用的临时变量已有定义



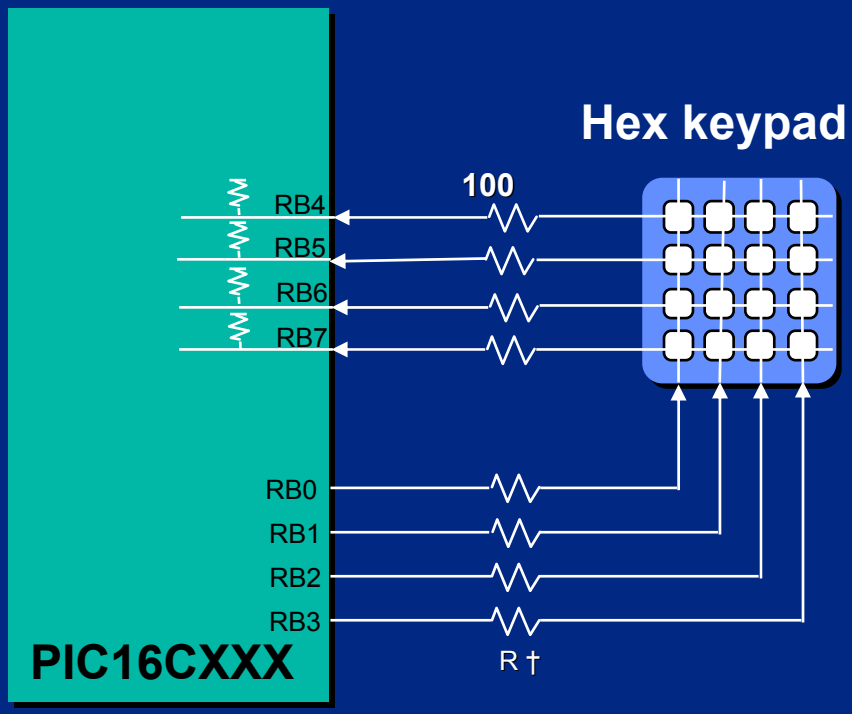
模板中有些什么？



- 向量
  - 复位，中断 向量及代码
- 中断 保护及恢复
  - 在中断保护代码及恢复寄存器
- 用户代码
  - *您的代码* 从这里开始
- EEPROM 初始化
- END 指令

外设:
PortB输入中断

键盘接口:



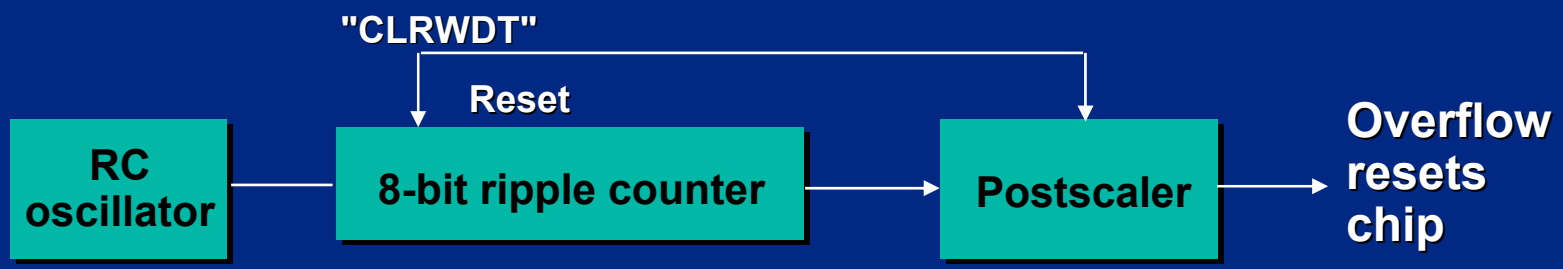
- 内部上拉(软件实现) 使 RB4-RB7 脚为高
- RB0-RB3 脚输出 ‘0’
- 按下任何建将使 RB 脚拉低并产生一中断
- 该中断能使处理器从SLEEP模式唤醒
- 节省了定时器资源

† Resistance for ESD protection

AN557

可靠性设计技术
硬件看门狗

- 有助于从软件故障中唤醒
- 使用芯片自带的 RC振荡
- WDT 不能由软件关掉
- WDT 溢出时使芯片复位
- CLRWDT 指令清 WDT
- 可编程的溢出周期: 18 ms 到 2.5 秒
- SLEEP模式时也工作
- 在 溢出时 唤醒 CPU



可靠性设计技术
软件看门狗

- 只用软件控制，看门狗 同样有效
- 整个程序中只使用 一个 CLRWDT 指令
- 将 CLRWDT 指令放在主循环中
- 不要将 CLRWDT指令放在 ISR 或其他子程序中
- 选择主循环能容许的最小WDT 溢出周期
- 以GOTO wdtreset指令(自循环) 初始化不用的内存来强制溢出

可靠性设计技术
填充未用的程序空间

- 当使用看门狗时，未用的区域填充GOTO wdtreset指令
- 该自循环正常情况不会执行
- 如果 PC 工作失常，就很有可能会执行这条指令
  - 通过 WDT 强制复位来初始化内核
- 在插入该指令时可使用FILL
  (假设<最末程序地址+1>是 400h):

- FILL (GOTO wdtreset), (400h-$)
- ORG 3FFh

可靠性设计技术
干净的 WDT 复位

- 上电时强制 WDT 复位:
  - 上电时检查 RAM模式
  - 如果模式不正确:
    - 初始化 RAM模式
    - 强制 WDT复位
- 如果代码跳到未用的程序区强制WDT复位

可靠性设计技术
子程序计数

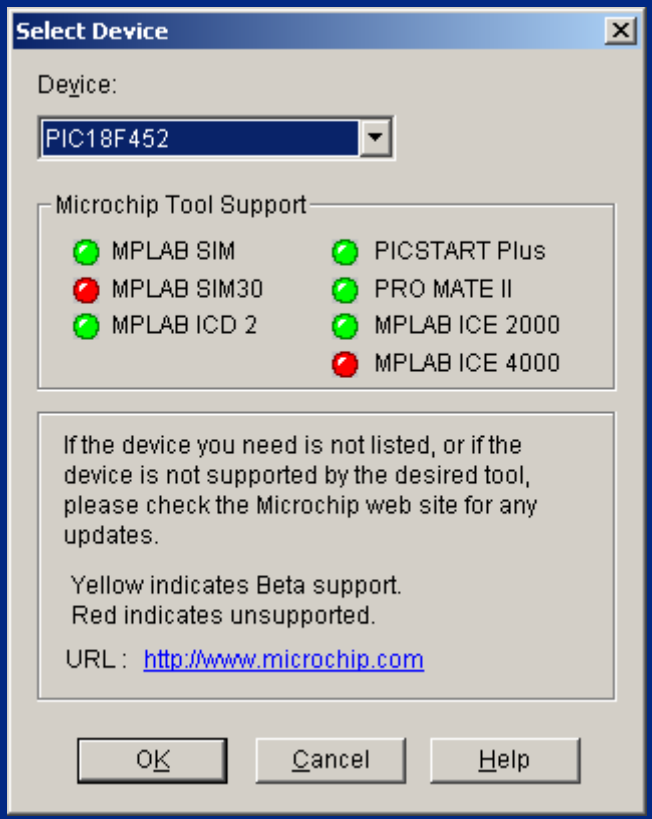
- 设定子程序调用计数器及子程序执行计数器
- 每调用一次子程序，子程序调用计数器自增1
- 在每个子程序开始使执行计数器自增1
- 在主循环开始，检查调用计数器和执行计数器是否相等
- 如果不等，强制 WDT 复位

术语表

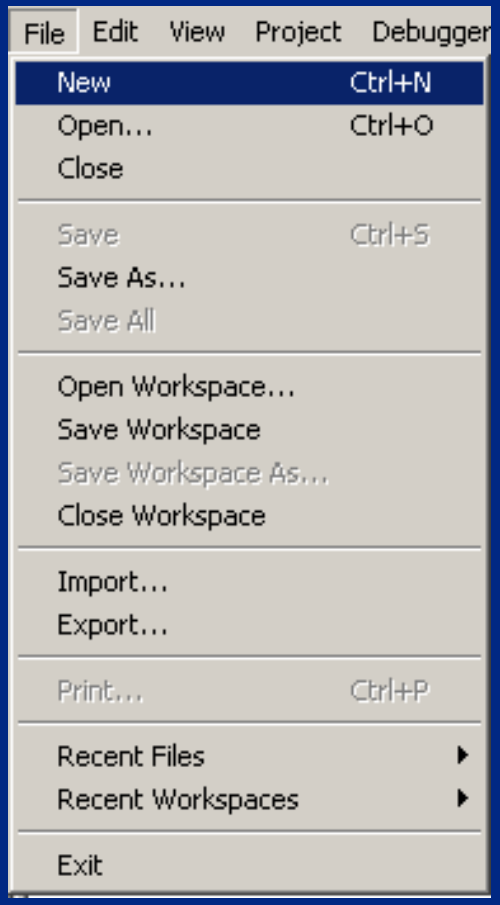
- TMR0 - 8-bit timer peripheral w/8 bit 预分频
- 预分频 - peripheral which delays the TMR0 count by a prescale ratio i.e. 1:1, 1:2, 1:4
- Page - memory organization for program memory
- Bank - memory organization for data memory
- Pipeline - hardware architecture for prefetching an opcode while executing the previous opcode
- Orthogonal instruction set - instructions work on ports and registers the same way
- OPTION_REG - 寄存器of control bits for configuring tmr0, INT, and pull ups
- INTCON - 寄存器of control bits for configuring peripheral 中断
- STATUS - 寄存器of bits indicating the results of an operation
- RP1&RP0 - direct addressing bank selection control bits
- IRP - indirect addressing bank selection control bit
- PORTB - I/O port B
- GPIO - I/O port on the 8 pin parts

Lab 1
Projects the Old Way

- 使用Configure > Select Device 来选择处理器
  - 注意有不同的开发工具来支持
  - 使用最新的 MPLAB IDE 确保得到最好支持

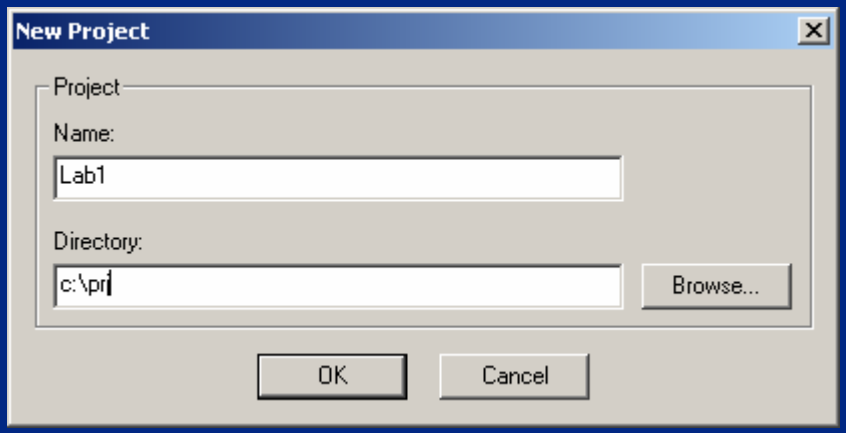


Lab 1
项目建立



- 点击 File > New 产生新的项目文件.
- 也可以通过 File > Open 打开文件完成此操作.
- 可以在此处保存workspace以及 import 和 export 文件供后面使用.

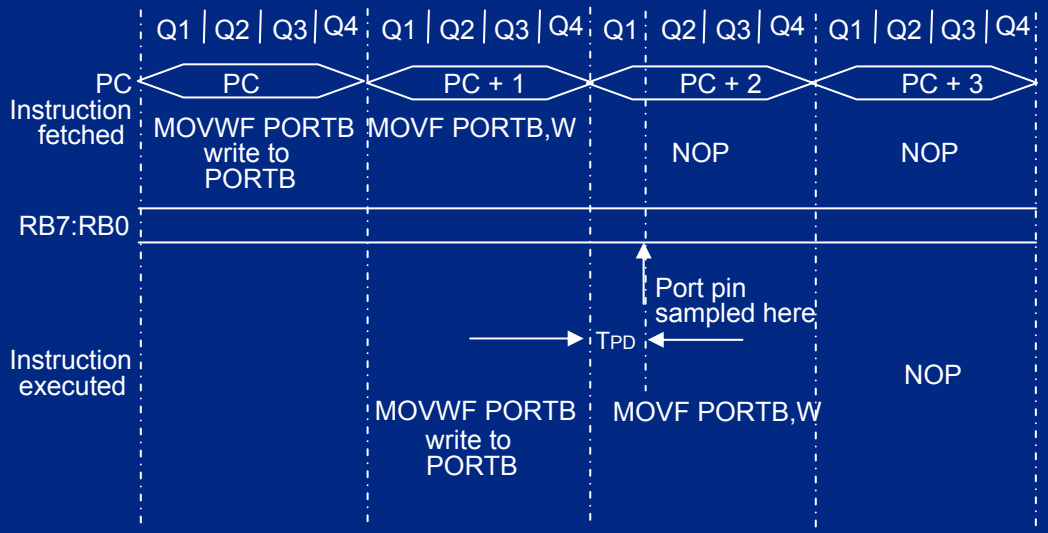
Lab 1
项目建立



- 点击 Project > New Project 在 Workspace 中建立新的项目文件
- 输入项目名及 *路径*

Lab 注意点:
读-修改-写

- Qcy = Fosc
- Instruction cycle = Fosc/4



注意:

本例显示对 PORTB一次写指令后紧跟一条读指令.

注意到:
数据建立时间 = (0. 25TCY-TPD)

这里 TCY= 指令周期
TPD = 传播延时

因此, 在较高时钟频率时,
写操作后紧跟读操作可能会出现问题.

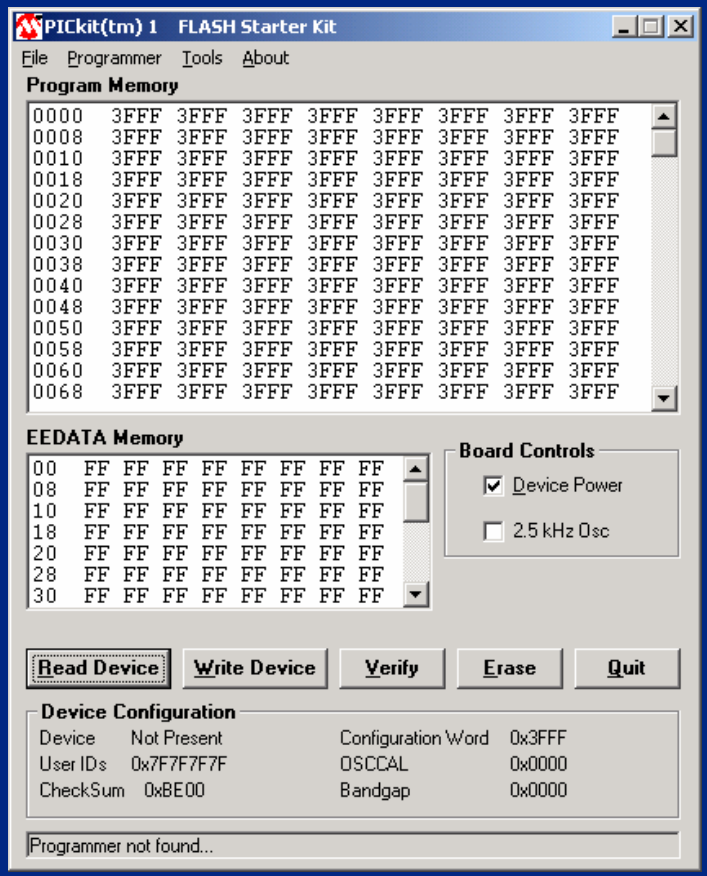
- 在反复读写口时要考虑口的负载

按钮/按键去抖或噪声输入

```
Check_key
                                        ;poll for key press
        btfsc   PORTB,1                         ;
        goto    Check_key               ;
;       call    Debounce_delay          ;debounce signal
        btfsc   PORTB,1                         ;check for bouncing input
        goto    Check_key               ;
        return

; Routine to give delay for key bounce routine
Debounce_delay
        .
        .
        delay routine here
        .
        return
```

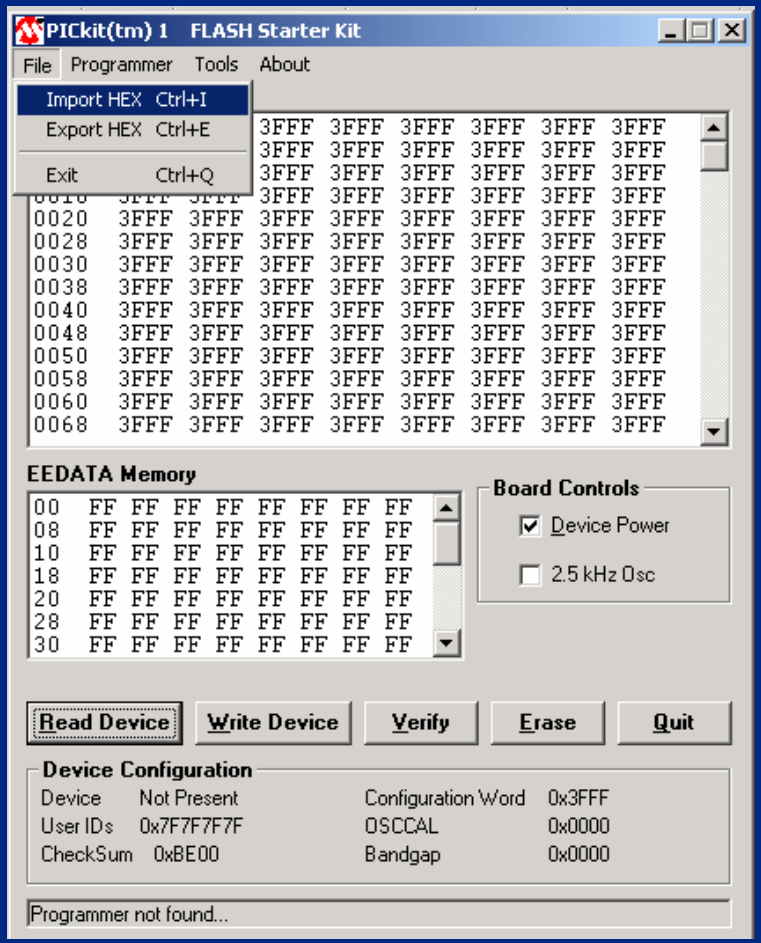
AN 566 contains debounce

Exercise #1:
启动 PICkit™ 1 软件



- Step 1: 启动软件
  - **Start->Programs-> PICkit™ 1 FLASH Start Kit-> PICkit™ 1 FLASH Starter Kit**

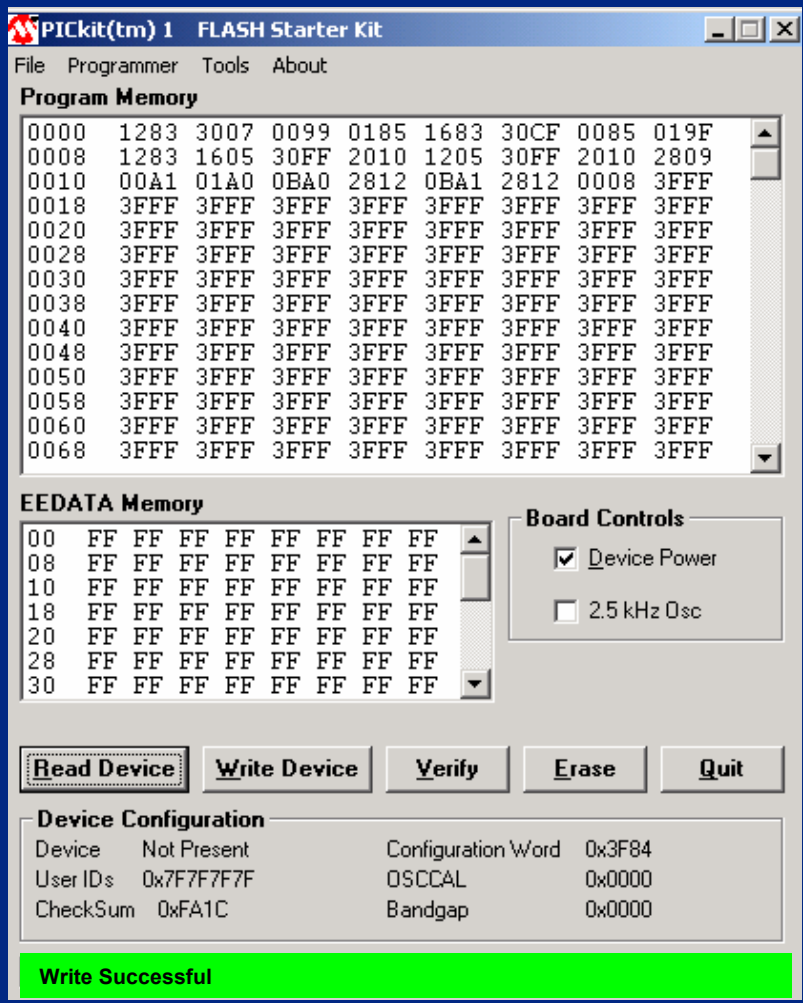
装载 Exercise1.hex 到缓冲



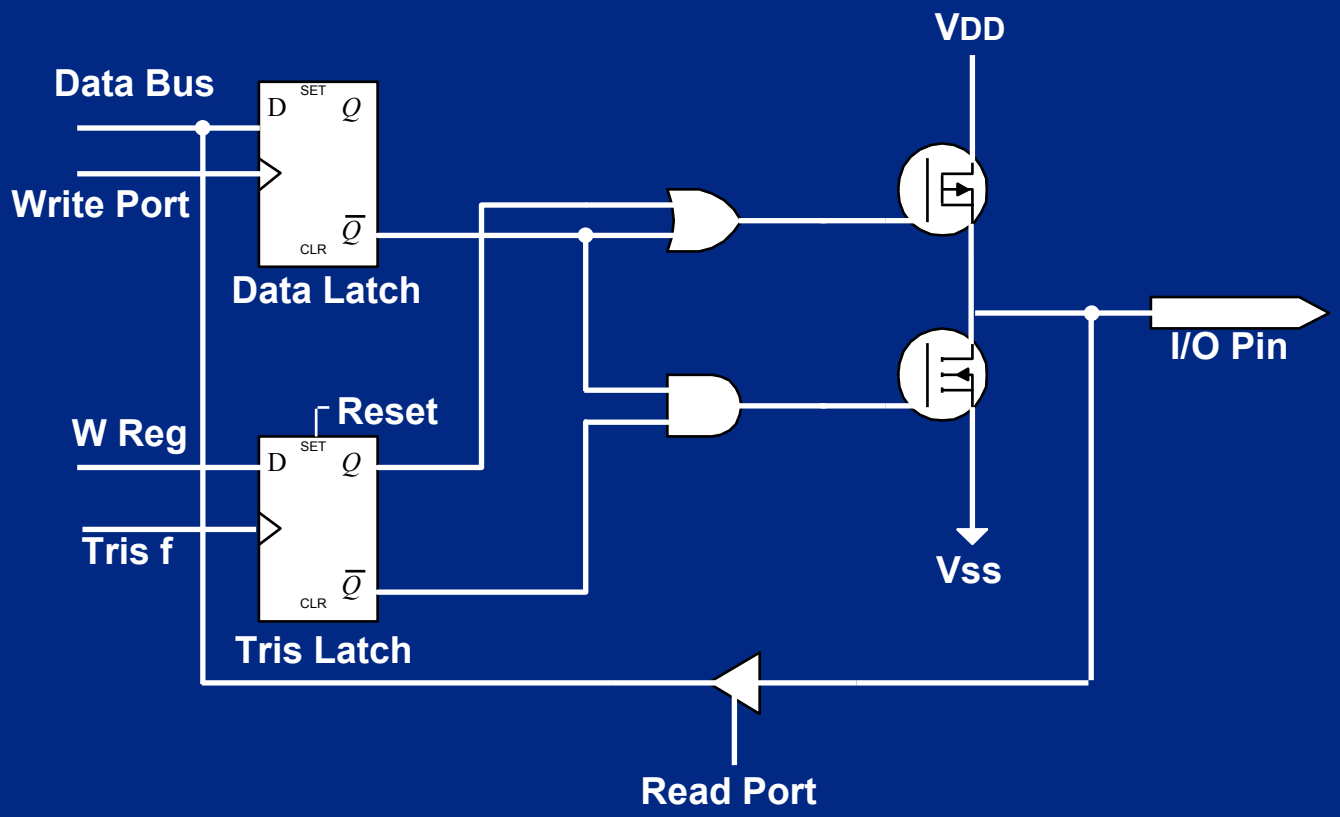
- Step 2:
  - 找到 Exercise1.hex 并装入缓冲
  - ***File->Import HEX***
    - E:/PICkit1/Exercise/
    - C:/Program Files/Microchip/PICkit1/Exercise/

Exercise1: 成功写入

- Step 4: 状态窗口
  - 成功写入
- Step 5: 成功装载后观察 PICkit™ 1
  - LED **D0** 应以200ms的速度闪烁



外设:
数字I/O框图


VDD
Data Bus
SET
D
Q
Write Port
CLR
Data Latch
I/O Pin
Reset
W Reg
SET
D
Q
Tris f
CLR
Tris Latch
Vss
Read Port