804 P18

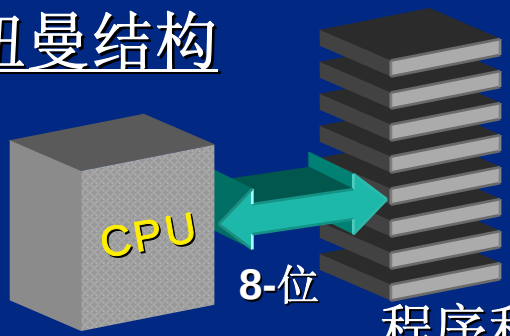
PIC18汇编程序学习

议题

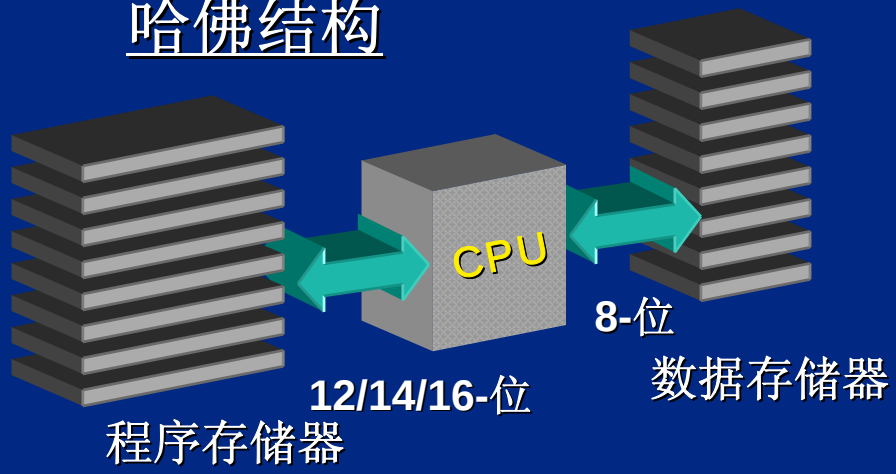
- 架构概述
- 休息
- Lab 1: USART 硬件操作
- Lab 2: 读取表格
- Lab 3: FSR 寄存器操作
- Lab 4: 字符串比较 (增强部分)

PICmicro®单片机架构
哈佛结构

冯-纽曼结构

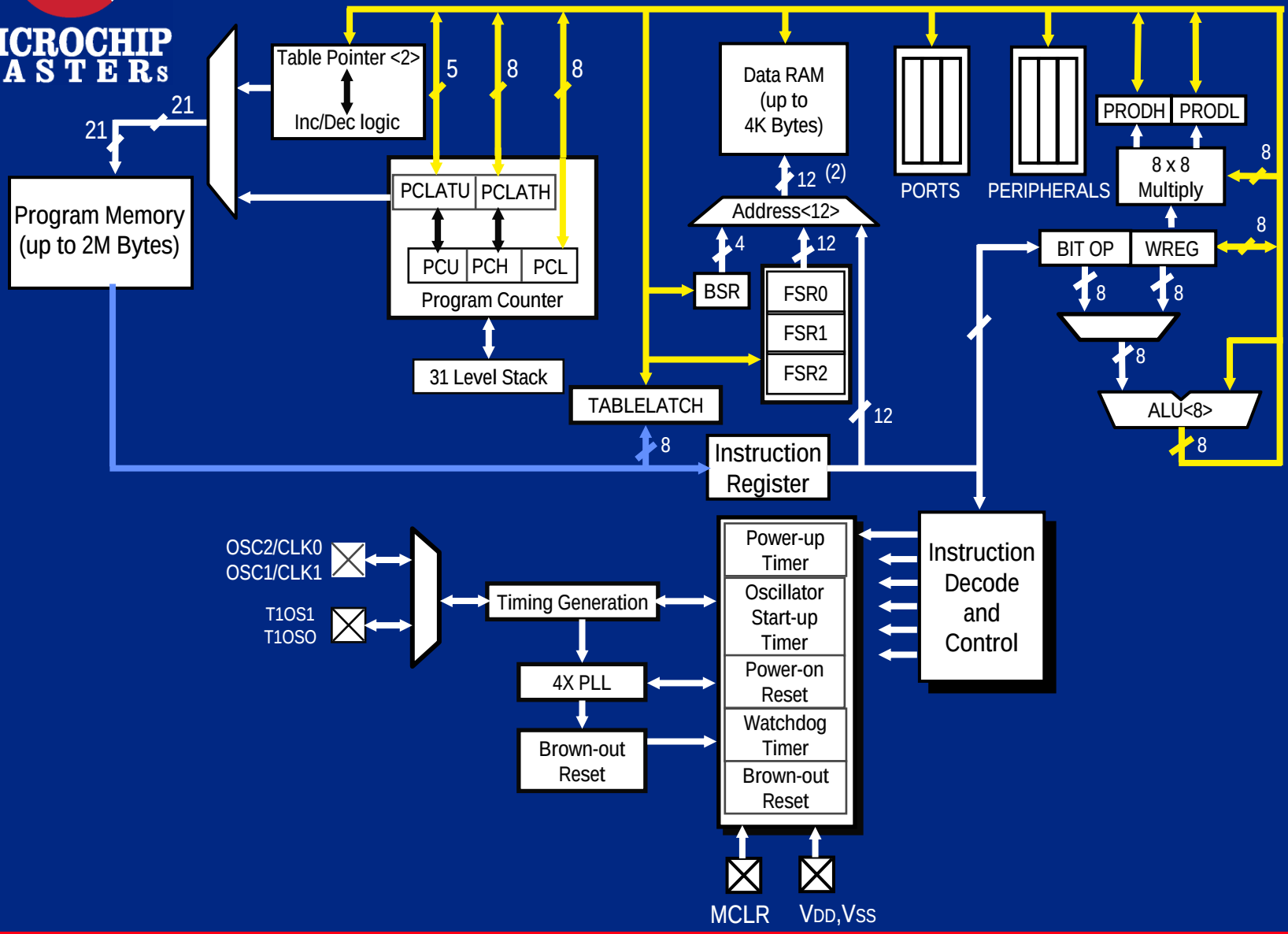


哈佛结构

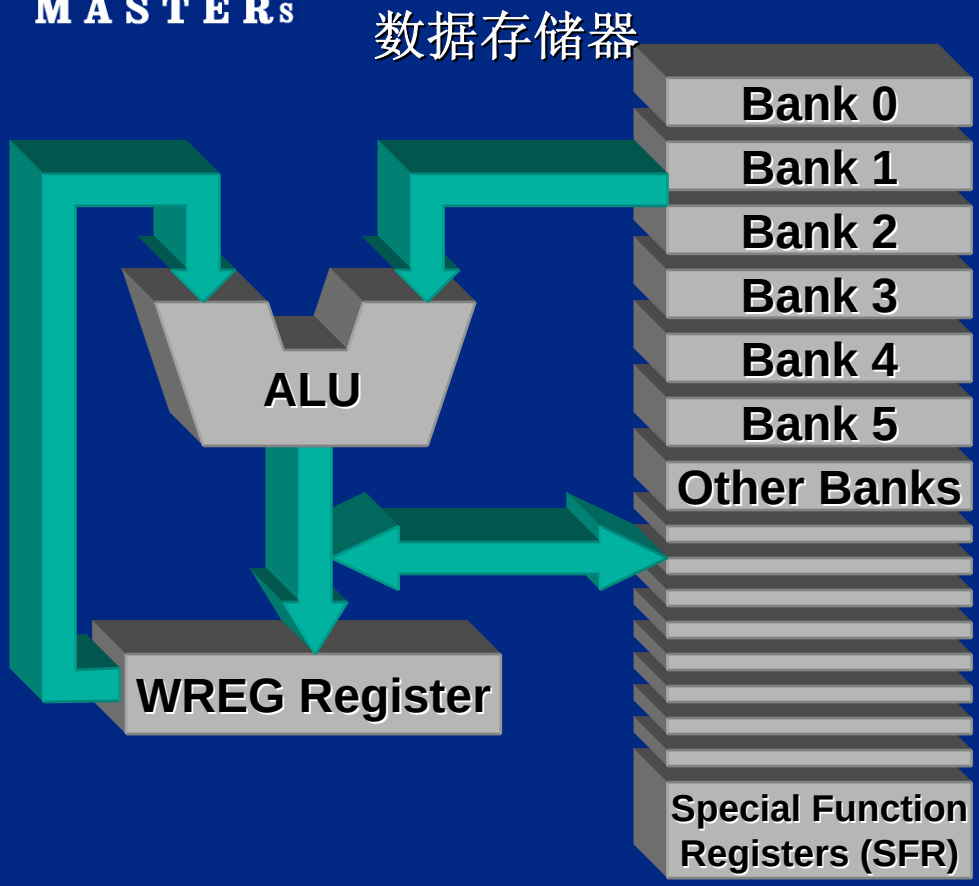


- 从同一存储器空间取指令和取操作数据.
  - 限制了数据流量
- 从两个独立的存储空间分别取指令和存取操作数.
  - 数据流量增加
  - 针对程序区和数据区可以设计不同的数据线宽度

PIC18 内部框图



寄存器文档概念

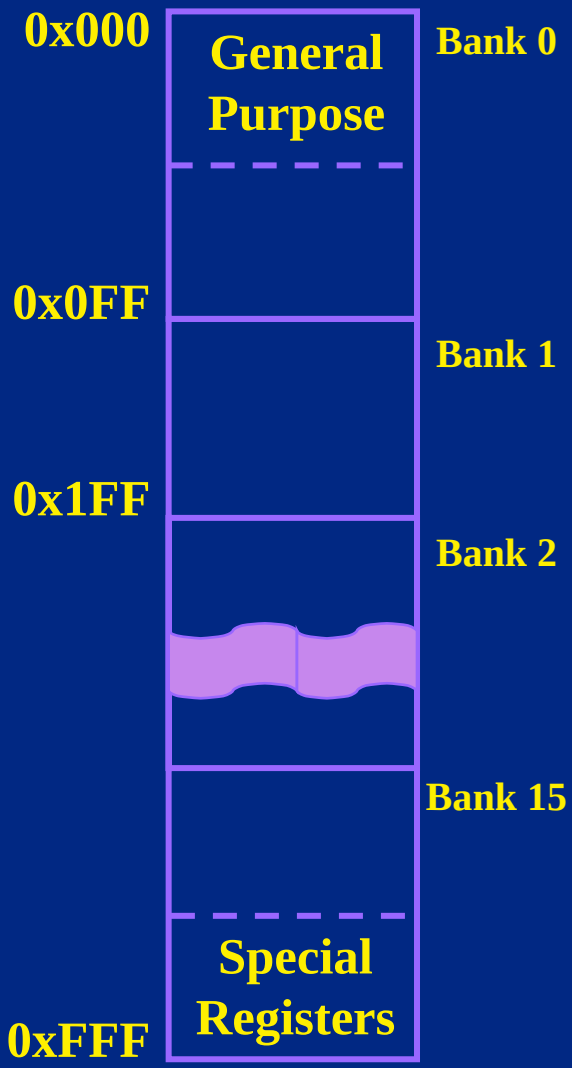


- SFRs 在Bank 15中
- 所有的指令操作都可针对所有的寄存器.
- 长字指令使指令可以直接寻址寄存器.
- 周边硬件寄存器和状态寄存器都可看作一个寄存器文档

16-位指令格式范例：

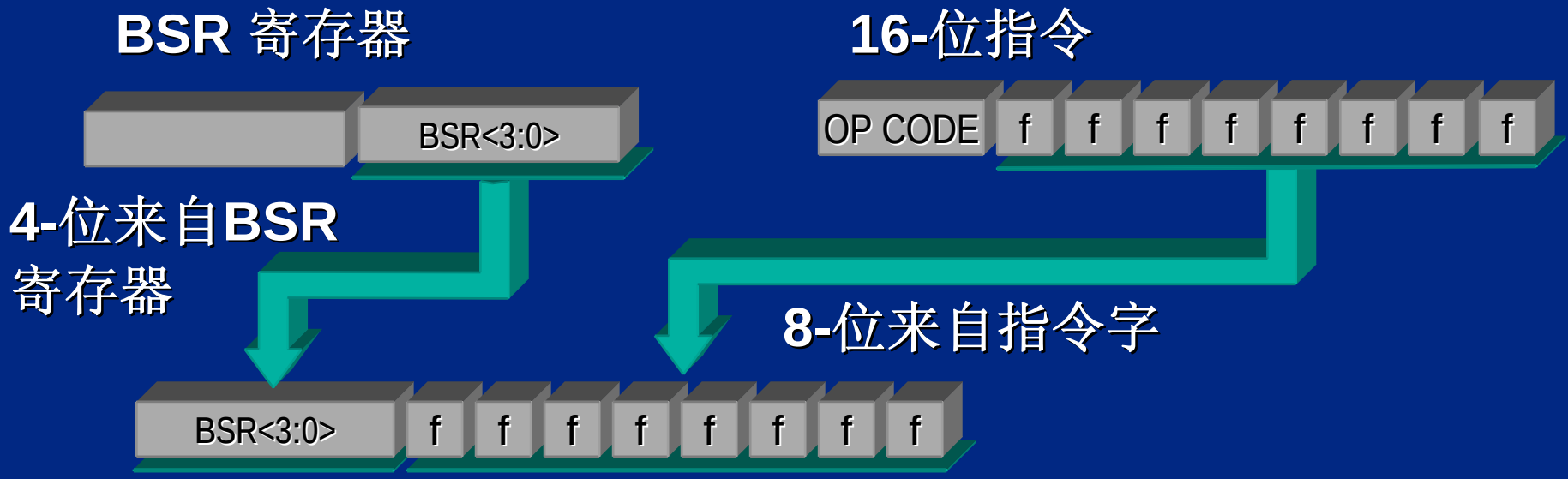
数据存储器结构

- RAM 数据存储器最大可到4K
- 分成为16个 组
  - 每组为256 个bytes
- 包括:
  - 特殊功能寄存器(SFR)
  - 通用寄存器RAM (GPR)
- SFRs 最高地址为0xFFFh
- GPR 地址从0x000h开始
- 之间包含了一些没有用到的存储空间



数据存储器: 直接寻址

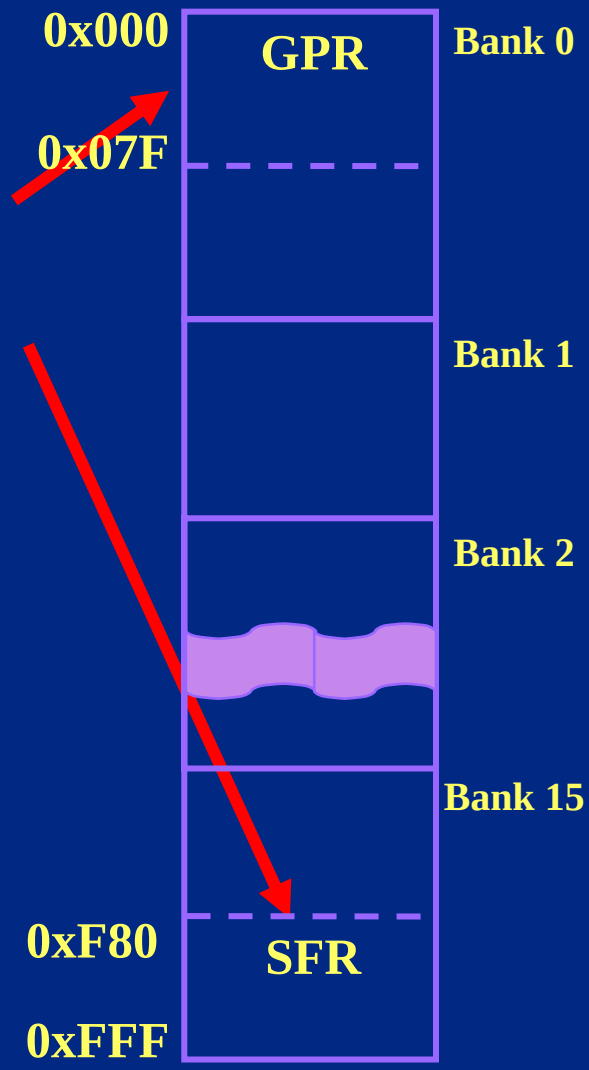
- 8-位直接地址从指令中得到
- 其余4-位由BSR寄存器指定
- **movlb** 指令将BSR进行加载



数据存储器: Access Bank

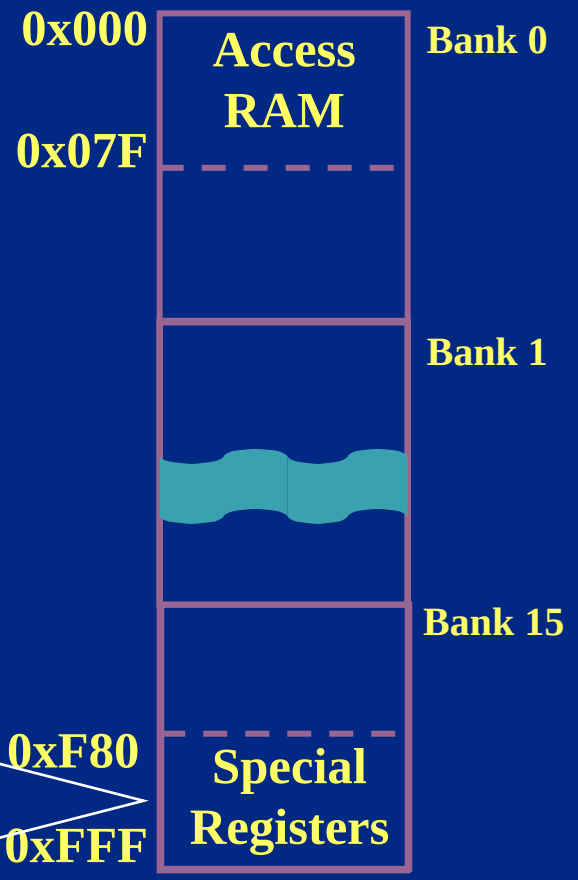
- “Access Bank”在数据存储器中的位置
  - Bank 0的首128个单元
  - Bank 15的后128个单元

- Access Bank 用于:
  - 中间计算值
  - 全局变量
  - 子程序的局部变量
  - 快速存取



数据寄存器: Access Bank

- 程序中的特定位可用来指定使用特定的组
- 当 ‘a’ 位 = 0, Bank 在以下空间进行切换:
  - 特殊功能寄存器
  - 在任意的块中对Access RAM 寻址



BSR 寄存器 = 0001

Force access bank = 0000 1111

ADDWF f,d,a `0010 01dr ffff ffff`

a = 0

数据寄存器: 字节操作

字节操作指令格式:

a = Access Bit

a = 0 for access bank
a = 1 for BSR value

d = Destination Bit

d = 0 for destination W
d = 1 for destination f

f = 8-bit Register Address

Program Example:

```
movlw   5
iorwf   Var1,F,ACCESS
movlb   HIGH(Var2)
xorwf   Var2,F,BANKED
```

数据存储器: 位操作

位操作指令格式

b = 3-Bit Address
(Bit Number)

a = Access Bit

a = 0 for access bank
a = 1 for BSR value

f = 8-bit Register Address

Program Example:

```
bsf     Var1,0,ACCESS
movlb   HIGH(Var2)
btg     Var2,7,BANKED
```

数据存储器：Access Bank

- Access Bank使用诀窍
  - 所有的寄存器都定义为12-位的地址
  - 让MPASM 来决定access bit

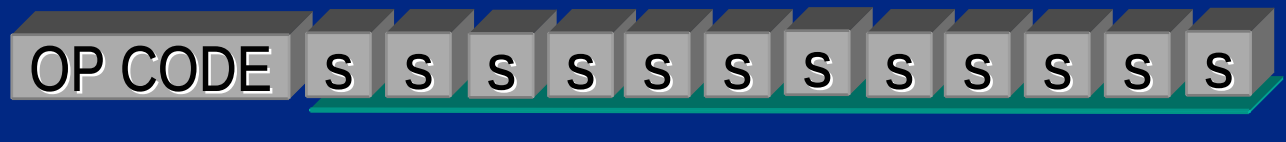
```
Program Example:
Var1    equ     0x001   ; Var1 is located in the access RAM
Var2    equ     0x100   ; Var2 is located in bank 1

org     0x000000        ; Reset vector location
movlw   5               ; Load 5 into WREG
addwf   Var1,F          ; Access bank used by MPASM
movlb   1               ; Set BSR for bank 1
addwf   Var2,F          ; BSR selected by MPASM
```

数据存储器: MOVFF 指令

- MOVFF指令允许数据在存储器和存储器间传输，而不必使用 WREG 或BSR
- 2 个指令字, 2 个指令周期

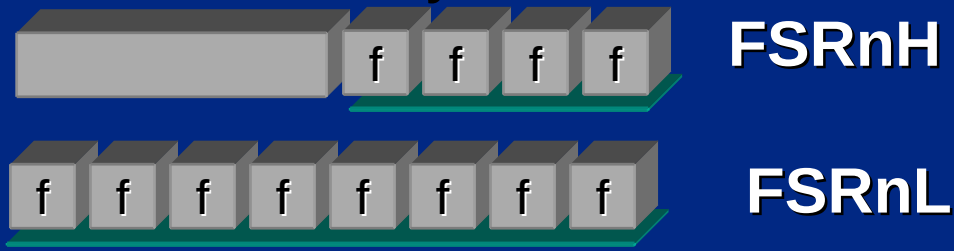
MOVFF 指令的格式



s, d = 12-bit Register Address

数据存储器: 间接寻址

- 三个12-位间接寻址寄存器 (FSRn)，称为文件选择寄存器
- 每个FSR分为两个8-位的寄存器
  - FSR0H, FSR0L
  - FSR1H, FSR1L
  - FSR2H, FSR2L
- 用LFSR 指令装载 FSR
  - 2 word, 2 cycle



数据存储器: 间接寻址

- 5 种模式来控制FSRxH ,FSRxL 的修改
- FSR 修改基于指令对模式寄存器的寻址

Register	C-equiv	Function
INDF0	*FSR0	间接寻址指针不变
POSTINC1	*FSR1++	间接寻址指针自动后增
POSTDEC0	*FSR0--	间接寻址指针自动后减
PREINC2	*(++FSR2)	间接寻址指针自动预增
PLUSW0	*(W+FSR0)	将W用作间接寻址索引

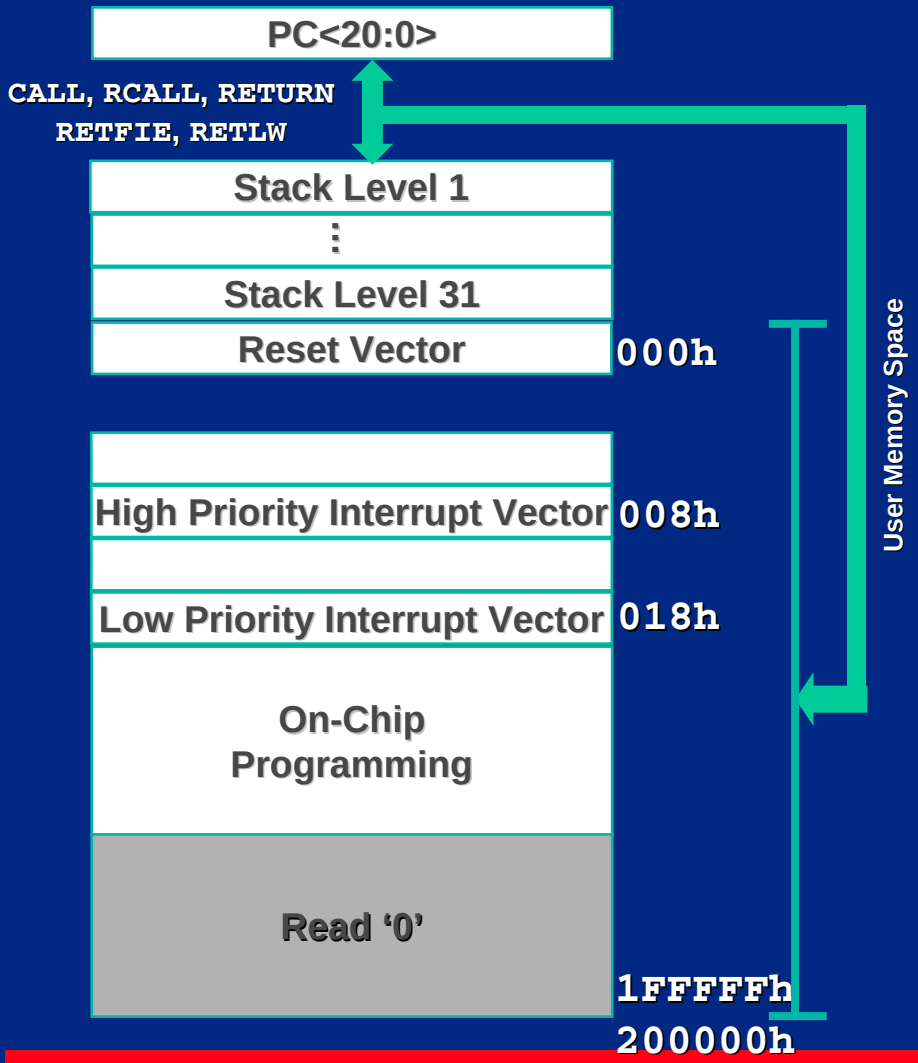
数据存储器: 间接寻址

- 清除RAM中地址0x000到0x5FF中的内容
  - 间接寻址的地址加载到FSRn中
  - INDFn被用作操作数,实际被使用的寄存器就是FSRn指向的位址

Program Example: Clearing RAM

```
        movlw   0x05            ; Value to compare
        lfsr    FSR1, 0x000     ; Set FSR to first RAM location
LOOP    clrf    POSTINC1        ; Clear location, increment FSR1
        cpfsgt  FSR1H           ; FSR1H > 5 ?
        goto    LOOP            ; NO, loop
        :                       ; YES, next instruction
```

PIC18 单片机架构
程序存储器



- 程序存储器空间最大可到2M字节(21 bits)
- 21-位的堆栈为31级深度
- 复位向量入口地址为0000h
- 中断向量的入口地址为0008h 和0018h

程序存储器中的字/字节

- 程序存储器以字节来定址
  - 是机器语言中默认的定址方式
- 指令的低有效位放在程序存储器定址对的低位中
- 程序存储器可以被看作最大为1 M的16位程序字
  - 和PIC17CXXX程序存储器类似

PIC18 单片机架构
程序计数器操作

- 21-位的PC
  - PCU, PCH, PCL
- 可直接操作的PC寄存器
  - PCLATU, PCLATH

- 指令总是以程序字的方式排列对齐
- PC的低有效位固定为‘0’
  - PC总是指向一个指令字

PIC18 单片机架构
硬件堆栈

- 可对栈顶进行访问的寄存器
  - TOSU, TOSH, TOSL
- 对堆栈操作的指令
  - PUSH, POP
- STKPTR 寄存器显示堆栈指针值，以及显示堆栈是否发生上溢或下溢

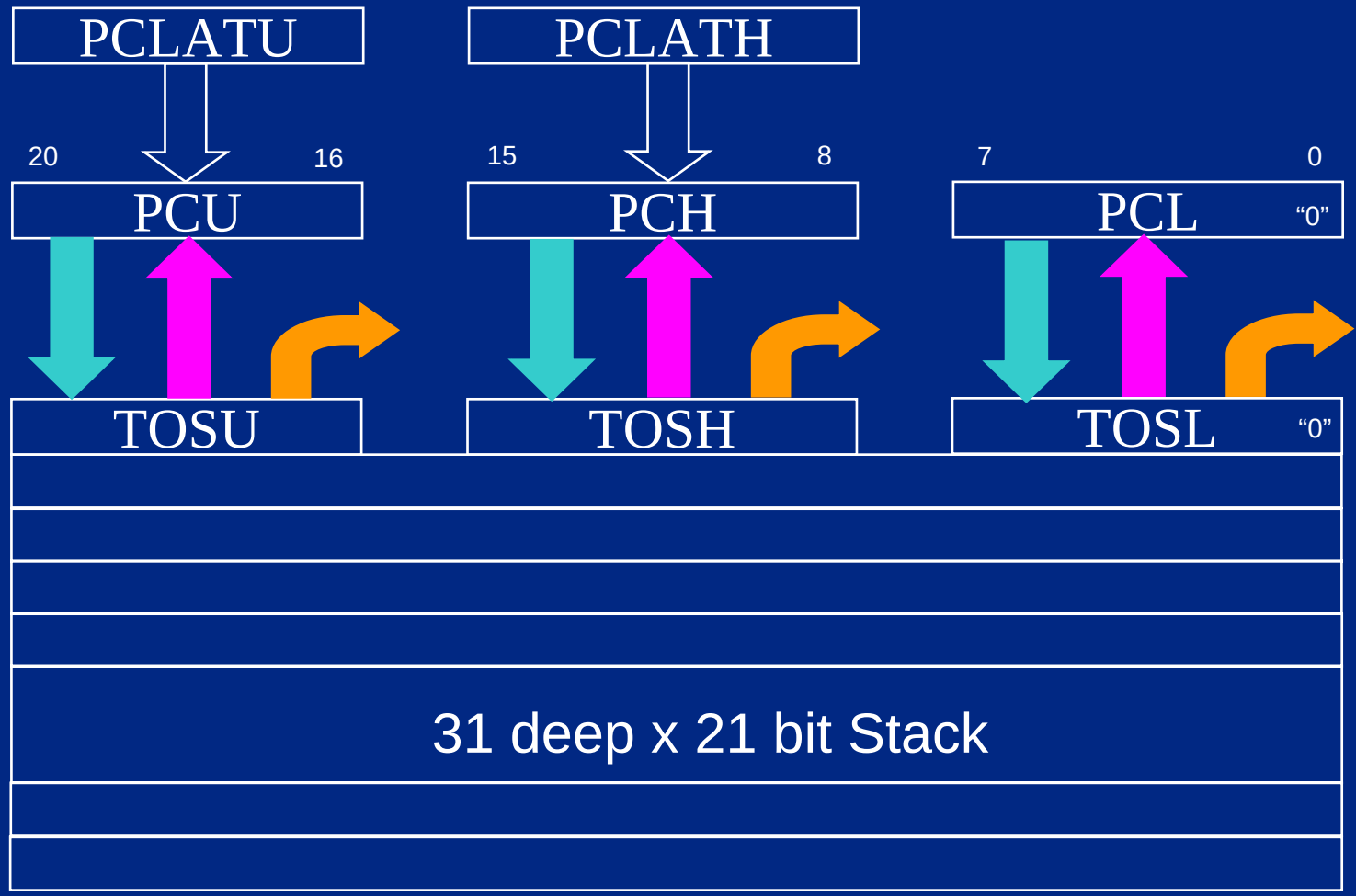
R/C-0	R/C-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STKOVF	STKUNF	-	SP4	SP3	SP2	SP1	SP0
bit7	6	5	4	3	2	1	0

PIC18 单片机架构
堆栈操作

CALL
RCALL
PUSH
INTERRUPT

RETURN
RETFIE
RETLW

POP



GOTO 指令

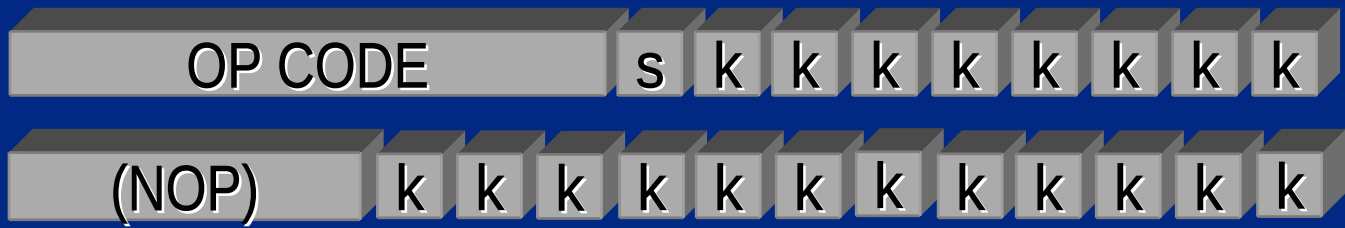
- *GOTO* 双字指令
  - 包含21-位 PC 值
- 不需要设置页
- 20-位“k” 值装载入PC<20:1>

GOTO Instruction Format

k = 20-bit Immediate Value

CALL 指令

- *CALL* 双字指令
  - 包含了21-位 PC
- 不需要设置页
- 20-位 “k” 值装载入 PC<20:1>
- 返回地址为(压栈值) PC+4



k = 20-bit Immediate Value

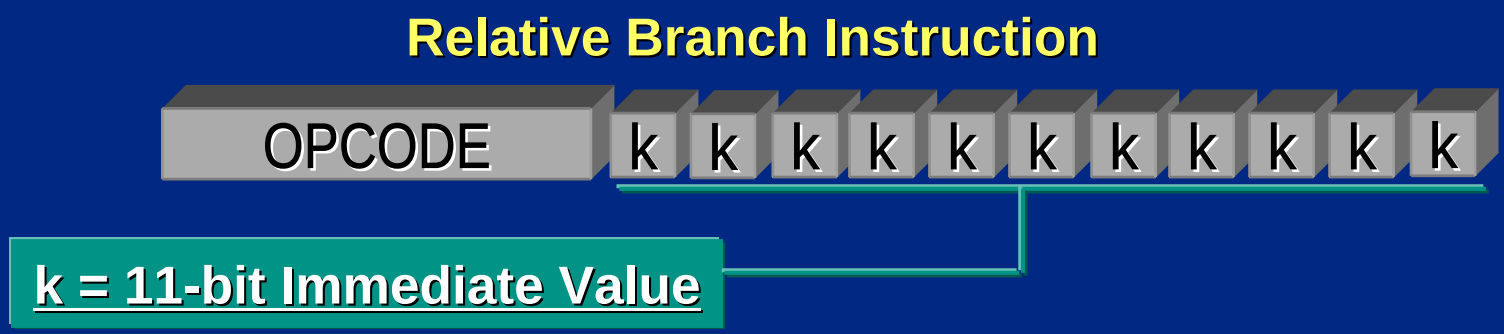
s = fast call/return mode select bit

相对分支指令

- 增加了两条相对分支指令: BRA,RCALL
  - 1 个指令字, 2个指令周期
  - 相对分支指令可以在以下范围跳转:

    -1024 指令 ($-2048)

    到 +1023 指令 ($+2046)

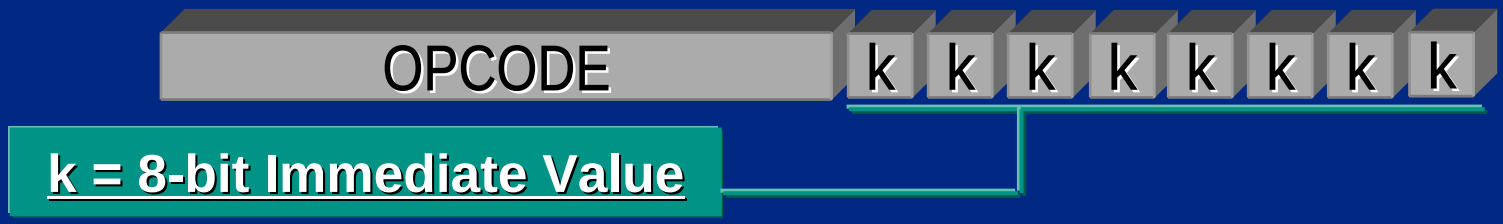


相对条件分支

- 增加了八条相对条件跳转指令:
  - 全部指令都是1 指令字 (1 到 2 指令周期)
  - 相对条件分支指令可在以下范围跳转:
    -128 指令 ($-256) 到+127 指令 ($+254)

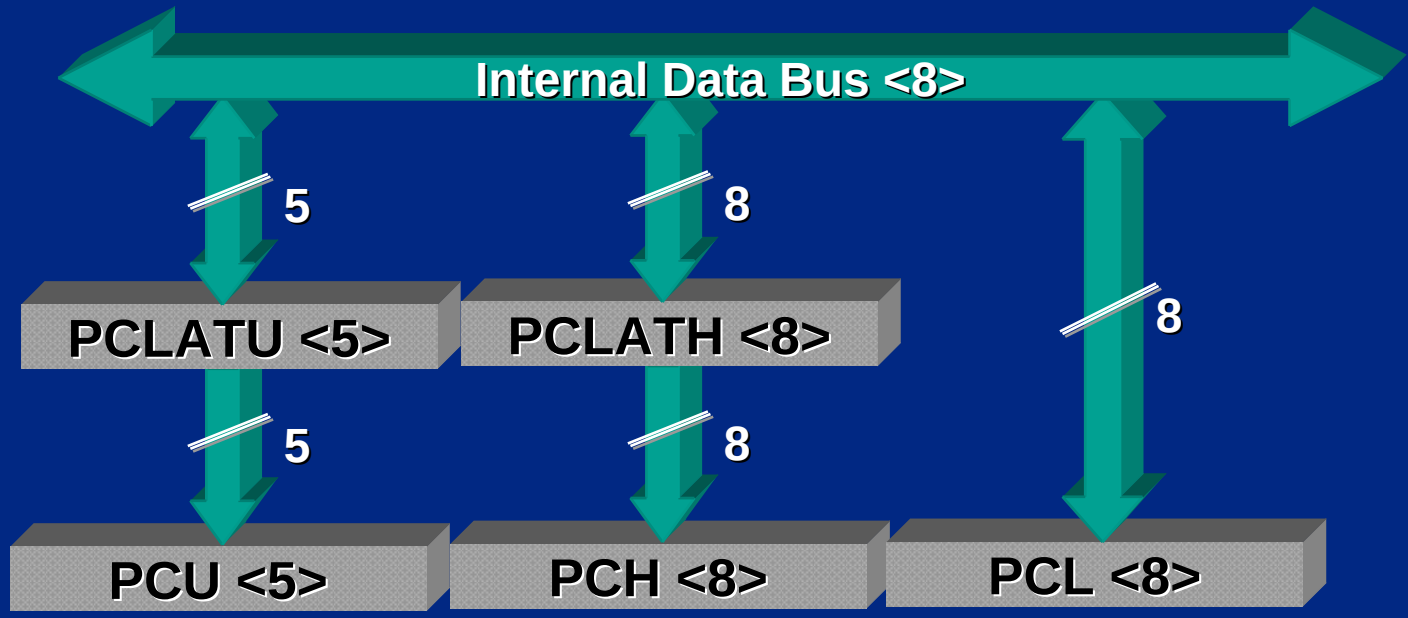
BC	**label**	**branch if carry (C=1)**	**BNC**
BZ	**label**	**branch if zero (Z=1)**	**BNZ**
BOV	**label**	**branch if overflow (OV=1)**	**BNOV**
BN	**label**	**branch if negative (N=1)**	**BNN**

Conditional Branch Instruction



程序存储器: PC 相对寻址

- 先改写 寄存器PCLATU 和PCLATH
- 然后改写寄存器 PCL,装载全部 21-位数值到程序计数器中



程序存储器: 立即数操作

- 8-位立即数包含在指令字中
- 立即数操作指令范例:

  `movlw`, `addlw`, `retlw`, etc.

16-bit Instruction for Literal Instructions

读/写程序存储器

- 专用的指令可以方便的读写程序存储器
  - TBLRD, TBLWT
- 允许对内部或外部存储器进行读/写
  - SRAM
  - FLASH
  - EPROM
  - 外设 (FPGA, USART, A/D…)

读程序存储器

- TBLRD 指令
- 用于:
  - 读取存储器内容
  - 计算存储器校验值
  - 从外设读取数据
- 需要两条指令来传输数据

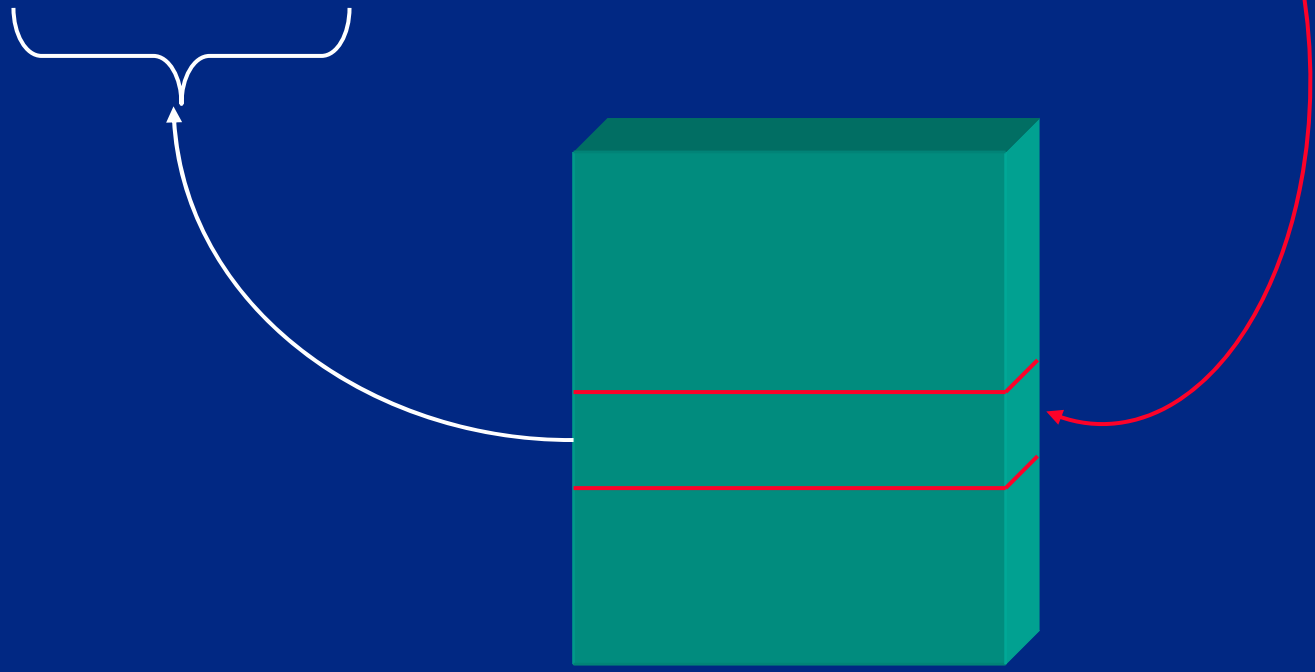
```
tblrd*          ;read PROGMEM(TBLPTR)->TABLAT
movff TABLAT,A;move TABLAT to location
```

tblrd 指令

Table Pointer | TBLPTRU | TBLPTRH | TBLPTRL |

Table Latch | TABLAT |

tblrd*



程序存储器

读表范例

```
movlw   UPPER(TBL_ADDR) ;Load the
movwf   TBLPTRU         ;table address
movlw   HIGH(TBL_ADDR)  ;upper, high and low
movwf   TBLPTRH
movlw   LOW(TBL_ADDR)
mowvf   TBLPTRL
tblrd*                  ;read PROGMEM(TBLPTR)->TABLAT
movff   TABLAT,LABEL    ;move the data byte out of
                        ;TABLAT
```

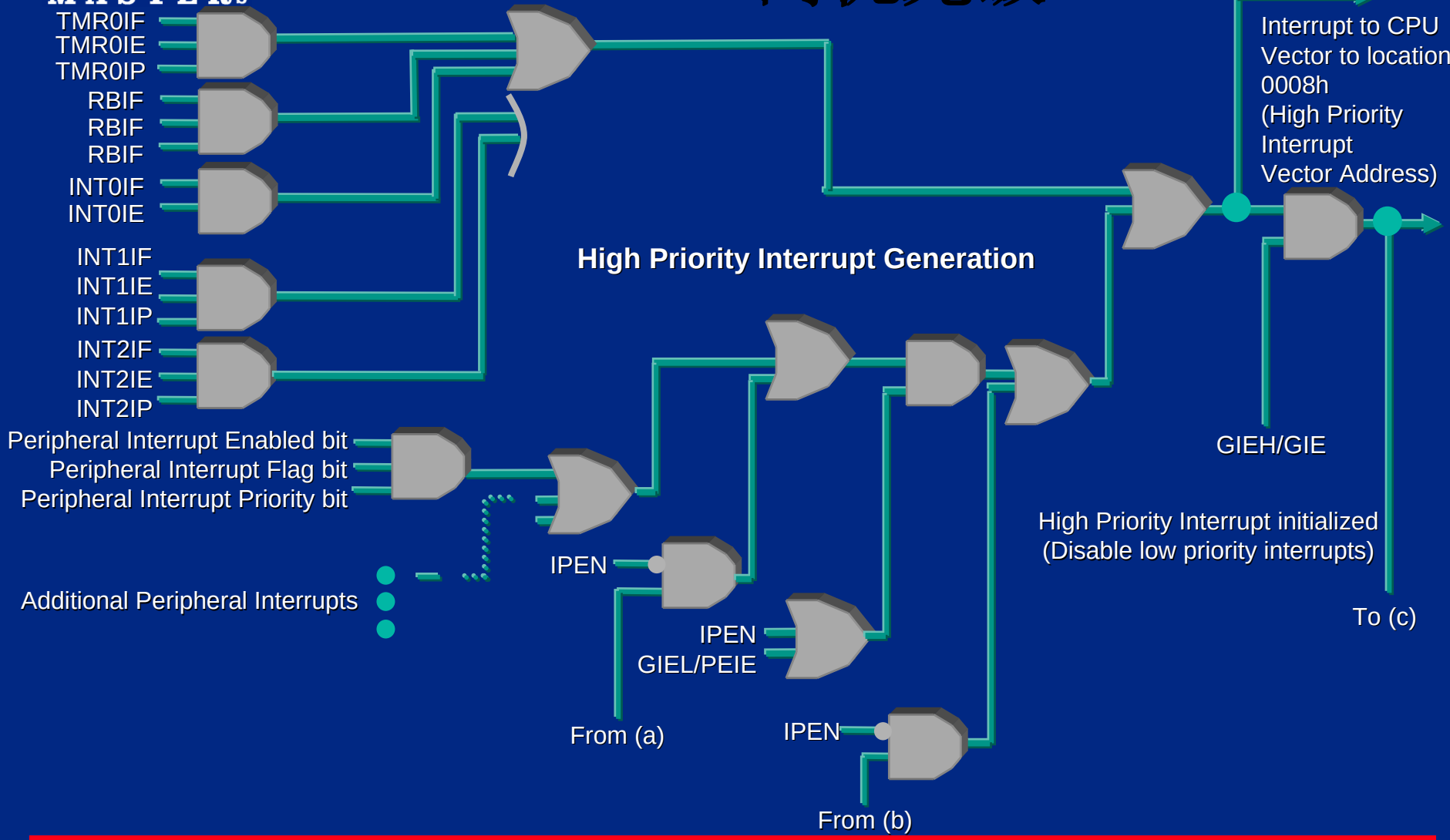
表指针操作

- 4种不同的模式增强了表操作的灵活性
- 模式操作数类似于C语言的后缀
- TBLRD* 表指针不变
- TBLRD*+ 表指针自动后增
- TBLRD*- 表指针自动后减
- TBLRD+* 表指针自动预增

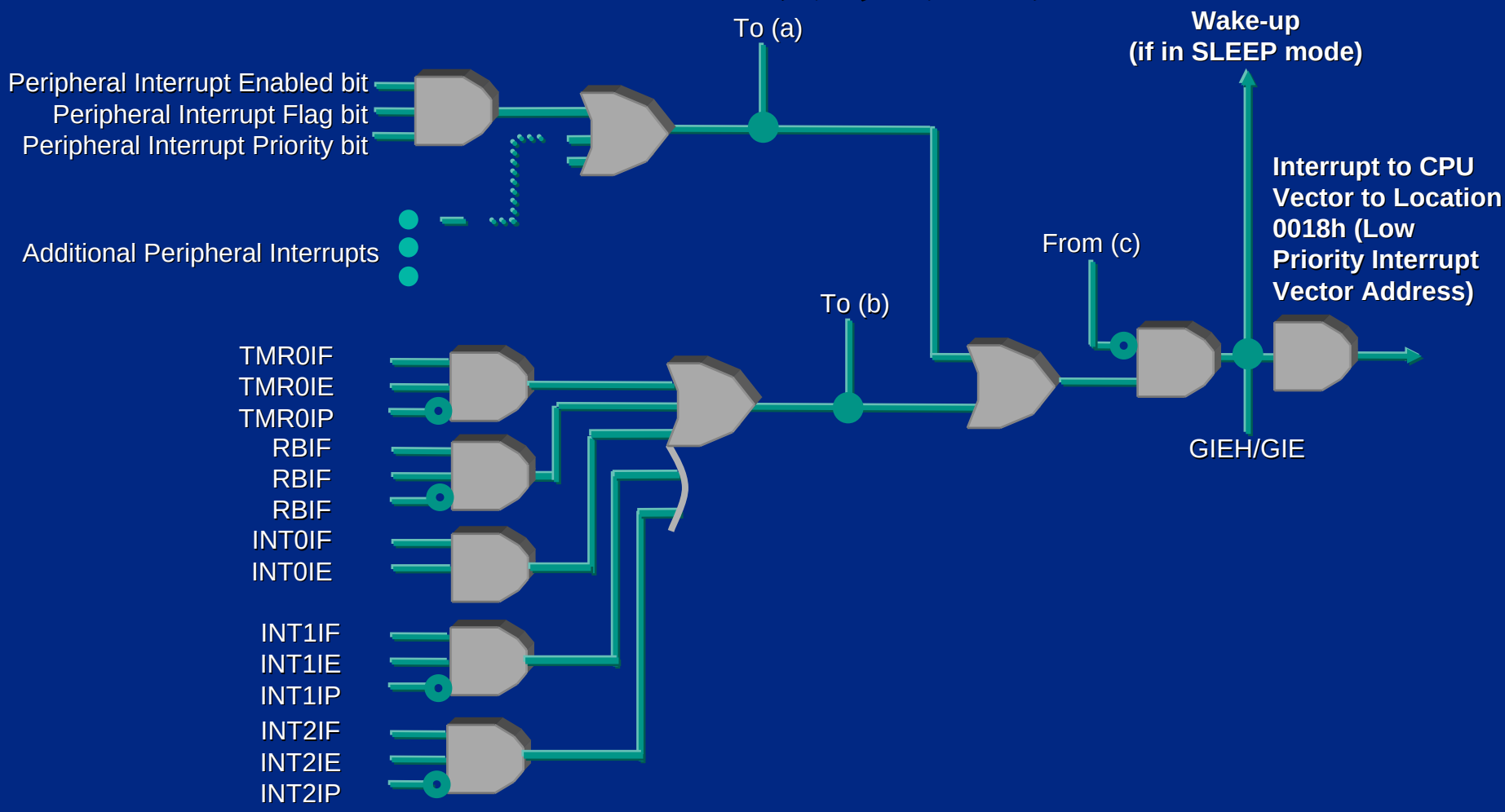
PIC18 单片机架构
中断概述

- 多个内部和外部中断源
- 每个外设中断都可以被配置成高/低优先级
- 在高/低优先级内的中断优先级别由软件设定
- 全局和单独的中断使能
- 每个中断有单独的中断标志
- 大多数中断可以唤醒单片机
- 固定的中断延时为3个指令周期

中断逻辑: 高优先级



中断逻辑: 低优先级



中断优先级使能

- IPEN 位用作中断模式
- RCON 寄存器:

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	-	RI	TO	PD	POR	BOR
bit7	6	5	4	3	2	1	

- 使能/禁止中断优先级
- 使能/禁止和PIC16CXXX 兼容的中断模式

PIC18 中断: 兼容模式

- 当IPEN=0， 兼容模式

- INTCON<7> 为GIE
- INTCON<6> 为PEIE

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE/GIEH	PEIE/GIEL	T0IE	INT0E	RBIE	T0IF	INT0F	RBIF
bit7	6	5	4	3	2	1	

- 中断类型与PIC16CXXX 家族兼容

PIC18 中断:
可选优先级中断模式

- 当IPEN=1时，可选优先级中断模式
  - INTCON<7> 为GIEH, INTCON<6> 为GIEL

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE/GIEH	PEIE/GIEL	T0IE	INT0E	RBIE	T0IF	INT0F	RBIF
bit7	6	5	4	3	2	1	

- 高优先级使能 GIEH 代替 GIE
- 低优先级使能 GIEL 代替 PEIE

优先级位

- INTCON2 寄存器包含了(例如) TMR0 优先级位

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG0	INTEDG1	INTEDG2	INTEDG3	T0IP	INT3P	RBIP
bit7	6	5	4	3	2	1	

- 如果T0IP=1 (复位状态), 那么T0IF为高优先级
- 如果T0IP=0, 那么T0IF为低优先级

- 注意 INTCON2 还有 RBPU 和INTEDG位
  - 兼容 PIC16CXXX OPTION寄存器

中断寄存器

	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
RCON	IPEN	LWRT	-	RI	TO	PD	POR	BOR
	bit7	6	5	4	3	2	1	

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INTCON	GIE/GIEH	PEIE/GIEL	T0IE	INT0E	RBIE	T0IF	INT0F	RBIF
	bit7	6	5	4	3	2	1	

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	INTEDG3	T0IP	INT3P	RBIP
	bit7	6	5	4	3	2	1	

	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
INTCON3	INT2P	INT1P	INT3E	INT2E	INT1E	INT3F	INT2F	INT1F
	bit7	6	5	4	3	2	1	

中断寄存器

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
	bit7	6	5	4	3	2	1	

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
	bit7	6	5	4	3	2	1	

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
IPR1	PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
	bit7	6	5	4	3	2	1	

	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
PIR2	-	-	-	-	BCLIF	LVDIF	TMR3IF	CCP2IF
	bit7	6	5	4	3	2	1	

	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
PIE2	-	-	-	-	BCLIE	LVDIE	TMR3IE	CCP2IE
	bit7	6	5	4	3	2	1	

	U-0	U-0	U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1
IPR2	-	-	-	-	BCLIP	LVDIP	TMR3IP	CCP2IP
	bit7	6	5	4	3	2	1	

影子寄存器

- 任何一个中断都可以用一级深度的影子寄存器来存储 WREG, BSR, 和 STATUS
  - 允许对存储器内容的快速处理

- 影子寄存器可用 RETFIE 指令进行恢复

```
RETFIE    FAST      ; Fast restore
```

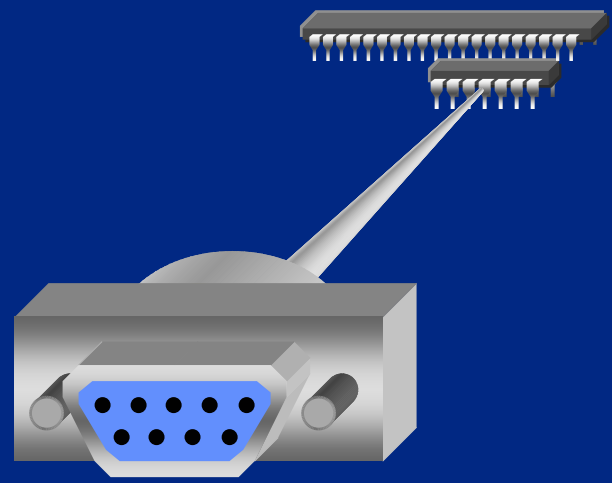
- 如果使用影子寄存器，中断处理子程序(ISR)禁止被其他中断打断
  - 可能会对低优先级中断处理子程序造成影响

影子寄存器

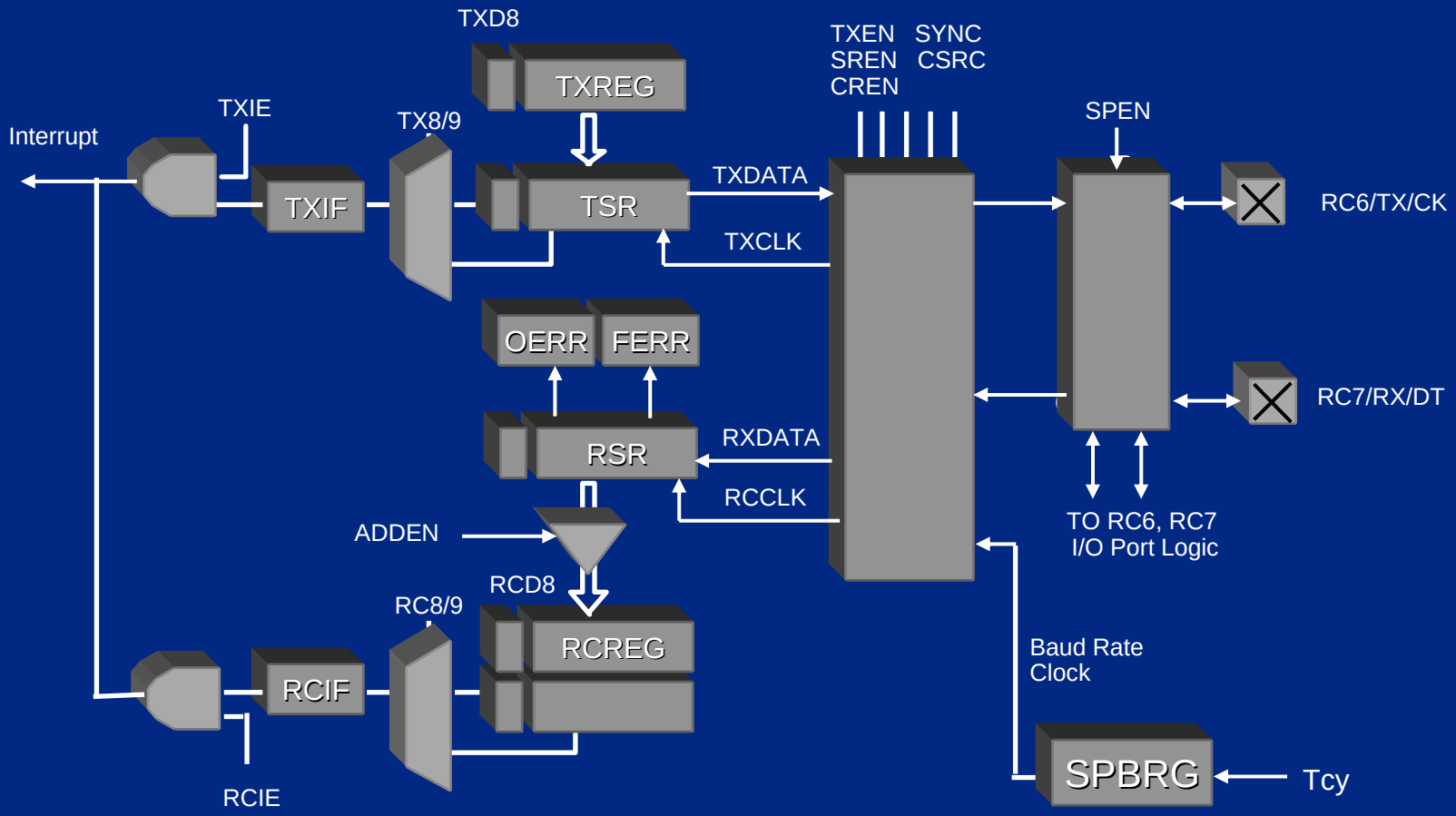
- CALL指令中的“s”位可以使能‘快速’CALL/RETURN操作
- 如果s = 1, WREG, STATUS, BSR 寄存器存入影子寄存器
- 指令中 s = 1 可以将影子寄存器中的内容恢复到 WREG, STATUS, BSR 寄存器
  - RETURN FAST
  - RETFIE FAST

串行通讯接口 (SCI/USART)

- 全双工异步方式或半双工同步方式
- 8- 或9-bit 数据
- 传送和发送双缓冲寄存器
- 发送和接收产生单独的中断
- 传送和接收从低有效位开始
- 专用的波特率发生器
- 最大波特率@ 40 MHz
  - 同步: 10 Mbaud
  - 异步: 625 Kbaud/2.5 Mbaud
- 9-位定址模式

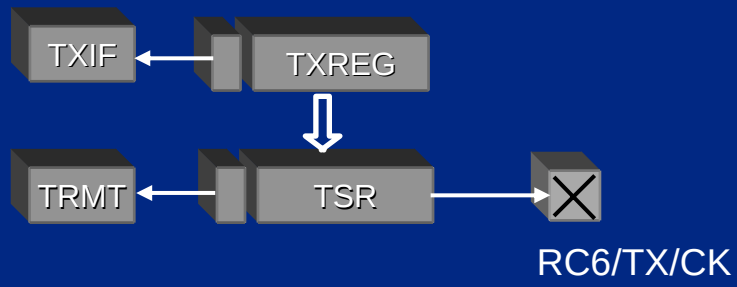


串行通讯接口
USART 框图



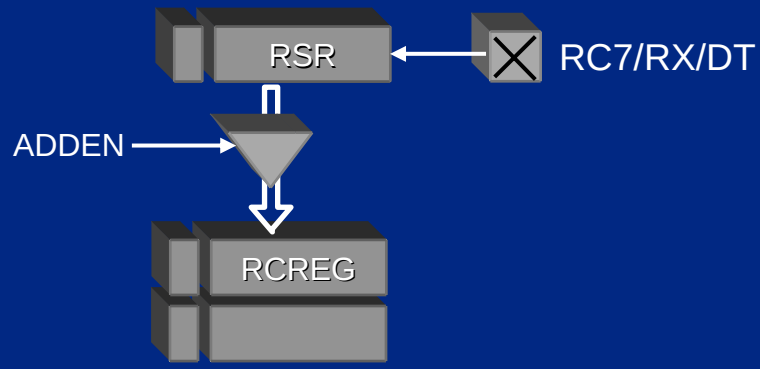
串行通讯接口
TXIF 和 TRMT 操作

- 如果TXREG 为空，TXIF = 1
- 装载TXREG 使 TXIF = 0
- 如果TSR 为空，TRMT = 1
- 装载 TSR使TRMT = 0
- 如果TXREG被装载而且 TRMT = 1, 数据立即被载入TSR. 串行数据开始进行移位，同时 TXIF = 1



串行通讯接口
RCIF 操作

- RSR 接收有效数据
- 数据载入 RCREG FIFO，
  RCIF = 1
- 如果第一个数据还没被处理时第二个数据被接收到，新接收的数据放入FIFO的第二个位置
- 当处理完第一个数据，如果FIFO中还有数据，那么第二个RCIF 中断产生



2 级深度 FIFO

动手试验

Lab 1: USART 硬件

- 目的:
  - 使用MPLAB® IDE
    - 生成一个项目
    - 在项目中添加文件
    - 添加指令和完成汇编
    - 用ICD 2来调试
  - 编写中断处理程序
  - 对SFRs进行初始化设置

Lab 1: USART 硬件

- Lab 1代码:
  - 在‘Startup’中配置USART
    - 19,200 波特率. @ 4 MHz
    - 传送使能，连续接收
    - 地址检测禁止
    - 设置为高优先级中断源
  - 在‘HighISR’中添加代码：
    - 检测USART 接收中断
    - 回送数据到PC终端
  - 用超级终端来校验结果

Lab 2: 读表

- Lab 2的目的:
  - 用汇编语言来使用程序存储器的数据.
  - 使用程序存储器内的数据
  - 使用条件判断指令

Lab 2: 读表

- Lab 2目的:
  - 在 ‘HighISR’中加入代码
    - 读取从USART接收到的字符
    - 如果字符是 <CR> (0x0d)
      - 将程序存储器内的信息发送出去
      - 直到‘0’找到为止

Lab 3: FSR 寄存器

- Lab 3目的:
  - 生成和调用功能块
  - 用FSRs操作数据存储器

Lab 3: FSR 寄存器

- Lab 3目的:
  - 生成功能块 ‘WriteUSART’
    - 等待直到TXREG为空
    - 发送WREG内的数值
  - 在‘Startup’中添加代码
    - 初始化FSR0，使它指向 ‘RXBuffer’

Lab 3: FSR 寄存器

- Lab 3目的:
  - 在 'HighISR'中添加代码
    - 将接收到的字符存到'RXBuffer'中
    - 如果 <CR> 接收到, 将收到的字符以反序传回PC终端
    - 同时在传送的字符串后附加上<CR> (0x0d)和<LF> (0x0f)

Lab 4: 字符串比较

- Lab 4目的:
  - 在前一个试验的基础上添加代码
    - 预先在程序存储器中定义字符: “Start”
    - 将接送到的字符回传
    - 将接收到的字符存储在‘RXBuffer’中
    - 当收到 <CR> ,
      - 将RXBuffer中的内容和程序存储器中的字符串相比较
      - 将RXBuffer 的指针值复位
    - 如果两者吻合，开启 PORTB 的LED

附录

PIC18 单片机指令集

Byte and Bit Oriented Operations

ADDWF	f,d,a	Add W to f
ADDWFC	f,d,a	Add W and Carry to f
ANDWF	f,d,a	AND W and f
CLRF	f,a	Clear f
COMF	f,d,a	Bit-wise complement f
CPFSEQ	f,a	Compare W to f, skip if =
CPFSGT	f,a	Compare W to f, skip if >
CPFSLT	f,a	Compare W to f, skip if <
DECF	f,d,a	Decrement f
DECFSZ	f,d,a	Decrement f, skip if 0
DECFSNZ	f,d,a	Decrement f, skip if not 0
INCF	f,d,a	Increment f
INCFSZ	f,d,a	Increment f, skip if 0
INCFSNZ	f,d,a	Increment f, skip if not 0
IORWF	f,d,a	Inclusive OR W and f
MOVF	f,d,a	Move f
MOVFF	f1,f2	Move source f1 to dest. f2
MOVWF	f,a	Move W to f
NEGF	f,a	Negate f
RLCF	f,d,a	Rotate left through carry
RLNCF	f,d,a	Rotate left (no carry)
RRCF	f,d,a	Rotate right through carry
RRNCF	f,d,a	Rotate right (no carry)
SETF	f,a	Set all bits in f
SUBFWB	f,d,a	Subtract f, Borrow from W
SUBWF	f,d,a	Subtract W from f
SWAPF	f,d,a	Swap nibbles of f
TSTFSZ	f,a	Test f, skip if 0
XORWF	f,d,a	Exclusive OR W and f
BCF	f,b,a	Bit clear f
BSF	f,b,a	Bit set f
BTFSC	f,b,a	Bit test f, skip if clear
BTFSS	f,b,a	Bit test f, skip if set
BTG	f,b,a	Bit toggle

f = File Register, d = destination (0=f, 1=W), a = access bank usage (0 = access, 1= BSR)

PIC18 单片机指令集

Control Operations

BC	k	Branch if Carry
BNC	k	Branch if Not Carry
BN	k	Branch if Negative
BNN	k	Branch if Not Negative
BOV	k	Branch if Overflow
BNOV	k	Branch if Not Overflow
BZ	k	Branch if Zero
BNZ	k	Branch if Not Zero
BRA	k	Branch unconditionally
CALL	k	Call subroutine
CLRWDT	-	Clear watchdog timer
GOTO	k	Go to address
NOP	-	No Operation
POP	-	Pop top of return stack
PUSH	-	Push top of return stack
RCALL	-	Relative Call
RESET	-	Software device RESET
RETFIE	s	Return from interrupt
RETURN	s	Return from subroutine

Literal and Control Operations

LFSR	n,k	Load 12-bit literal to FSRn
RETLW	k	Return with literal in W
SLEEP	-	Go into standby mode
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, b = bit address <0,7>, d = destination (0=f, 1=W)

PIC18 单片机指令集

Program Memory Operations		
TBLRD*	-	Table Read
TBLRD+*	-	Table Read, Pre-increment
TBLRD*+	-	Table Read, Post-increment
TBLRD*-	-	Table Read, Post-decrement
TBLWT*	-	Table Write
TBLWT+*	-	Table Write, Pre-increment
TBLWT*+	-	Table Write, Post-increment
TBLWT*-	-	Table Write, Post-decrement

PIC18 代码范例

```
; Polling a bit
Poll    btfss   PIR1, TMR1IF    ; Has Timer1 rolled over yet?
        bra     Poll            ; No, keep checking
        •                       ; Yes, continue code
        •
```

```
; 16-bit Addition/Subtraction
movlw   LOW(Constant)  ; Get low byte to be added
addwf   LowByte,F      ; Add the low byte
movlw   HIGH(Constant) ; Get high byte to be added
addwfc  HighByte,F     ; Add the high byte + carry

movlw   LOW(Constant)  ; Get low byte to subtract
subwf   LowByte,F      ; Subtract the low byte
movlw   HIGH(Constant) ; Get high byte to subtract
subwfb  HighByte,F     ; Subtract low byte - borrow
```

PIC18 代码范例

```
; Comparing a value
        movlw  ConstantValue ; Put compare value in WREG
        cpfseq INDF0         ; Is value in INDF0 equal?
        bra    NotEqu        ; No, go to appropriate code
IsEqu           •           ; This code executed when
                •           ; equal
        return
NotEqu  •                   ; This code executed when
        •                   ; not equal
        return
```

PIC18代码范例

```
; Copy 10 bytes of data memory and reverse order of the bytes

        lfsr    FSR0,Buffer1            ; Point to source buffer
        lfsr    FSR1,(Buffer2+10)       ; Point to end of dest. buffer
        movlw   .10                     ;
Loop    movff   POSTINC0,POSTDEC1       ; Move bytes and update FSRs
        decfsz  WREG,F                  ; Ten bytes moved yet?
        Bra     Loop                    ; No, keep looping
        •                               ; Yes, continue execution
        •                               ;
```

PIC18代码范例

```
; Reading a value from Program Memory

movlw   UPPER(Address)          ; Load the
movwf   TBLPTRU                 ; table address
movlw   HIGH(Address)           ; upper, high and low
movwf   TBLPTRH
movlw   LOW(Address)
mowvf   TBLPTRL
tblrd*                          ; read PROGMEM(TBLPTR)->TABLAT
movf   TABLAT,W                 ; move the data byte out
                                ; of TABLAT
```