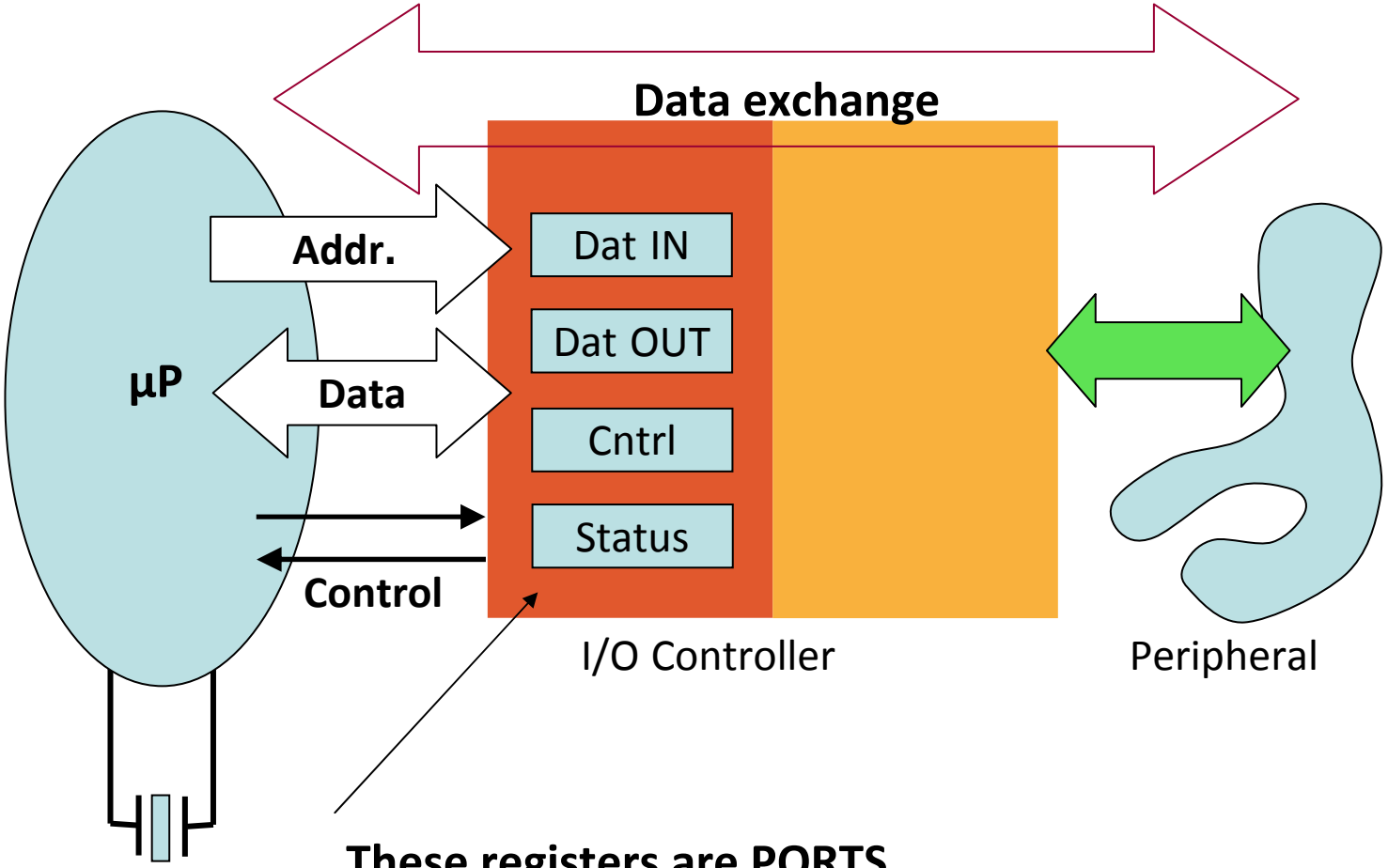
Microprocessor based digital Systems

Fundamentals of I/O

Guillermo Carpintero
Marta Ruiz
Universidad **Carlos III** de Madrid

Input/Output Interface

Input/Output Controller

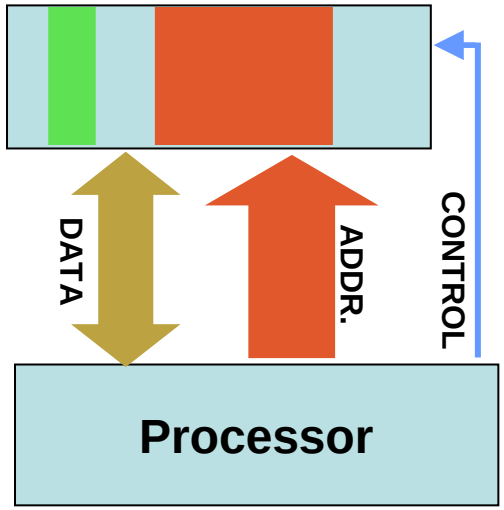


These registers are PORTS

Where are these registers located?

Memory Mapped I/O

The PORTS are located within the memory address space



The Microchip PIC uses memory mapped solution

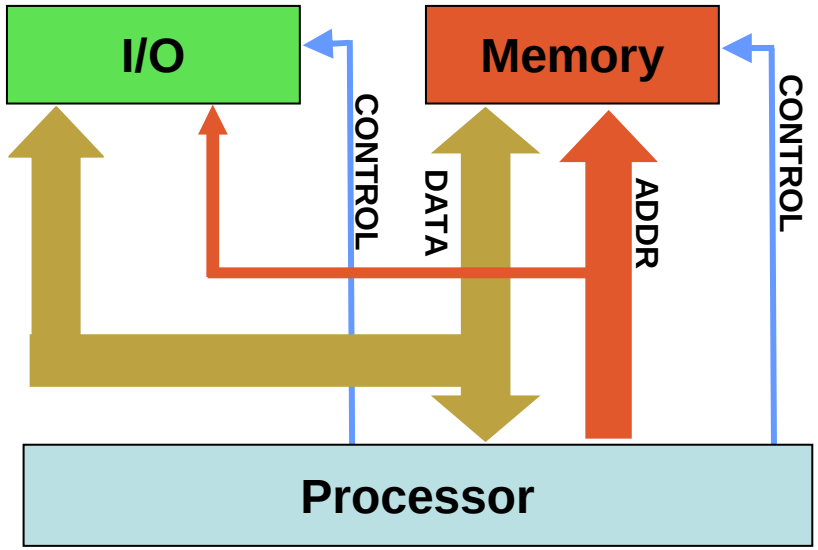
Memory Mapped I/O

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2(3)	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2(3)	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2(3)	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2(3)	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2(3)	FBBh	CCPR2L	F9Bh	—
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—
FF9h	PCL	FD9h	FSR2L	FB9h	—	F99h	—
FF8h	TBLPTRU	FD8h	STATUS	FB8h	—	F98h	—
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	—	F97h	—
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	—	F96h	TRISE(2)
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD(2)
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—
FF0h	INTCON3	FD0h	RCON	FB0h	—	F90h	—
FEFh	INDF0(3)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0(3)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0(3)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE(2)
FECh	PREINC0(3)	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD(2)
FEBh	PLUSW0(3)	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—
FE7h	INDF1(3)	FC7h	SSPSTAT	FA7h	EECON2	F87h	—
FE6h	POSTINC1(3)	FC6h	SSPCON1	FA6h	EECON1	F86h	—
FE5h	POSTDEC1(3)	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1(3)	FC4h	ADRESH	FA4h	—	F84h	PORTE(2)
FE3h	PLUSW1(3)	FC3h	ADRESL	FA3h	—	F83h	PORTD(2)
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

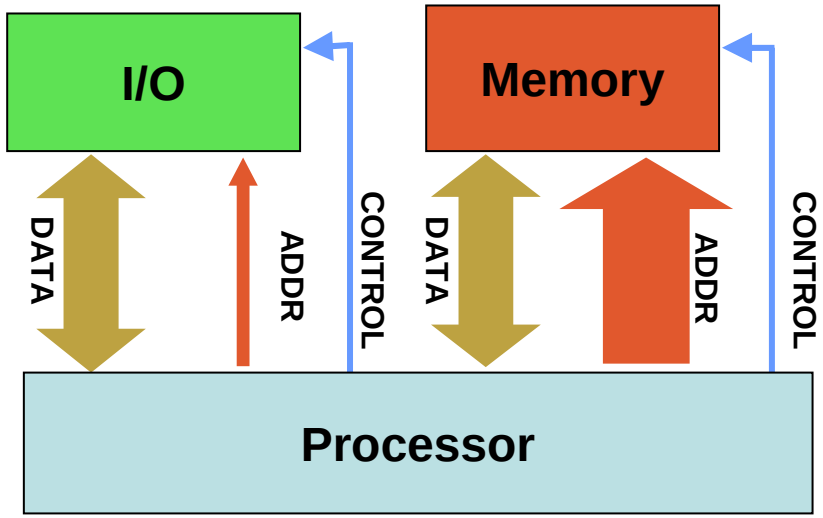
ports

Other Memory Models

Dedicated I/O
We must have I/O instructions

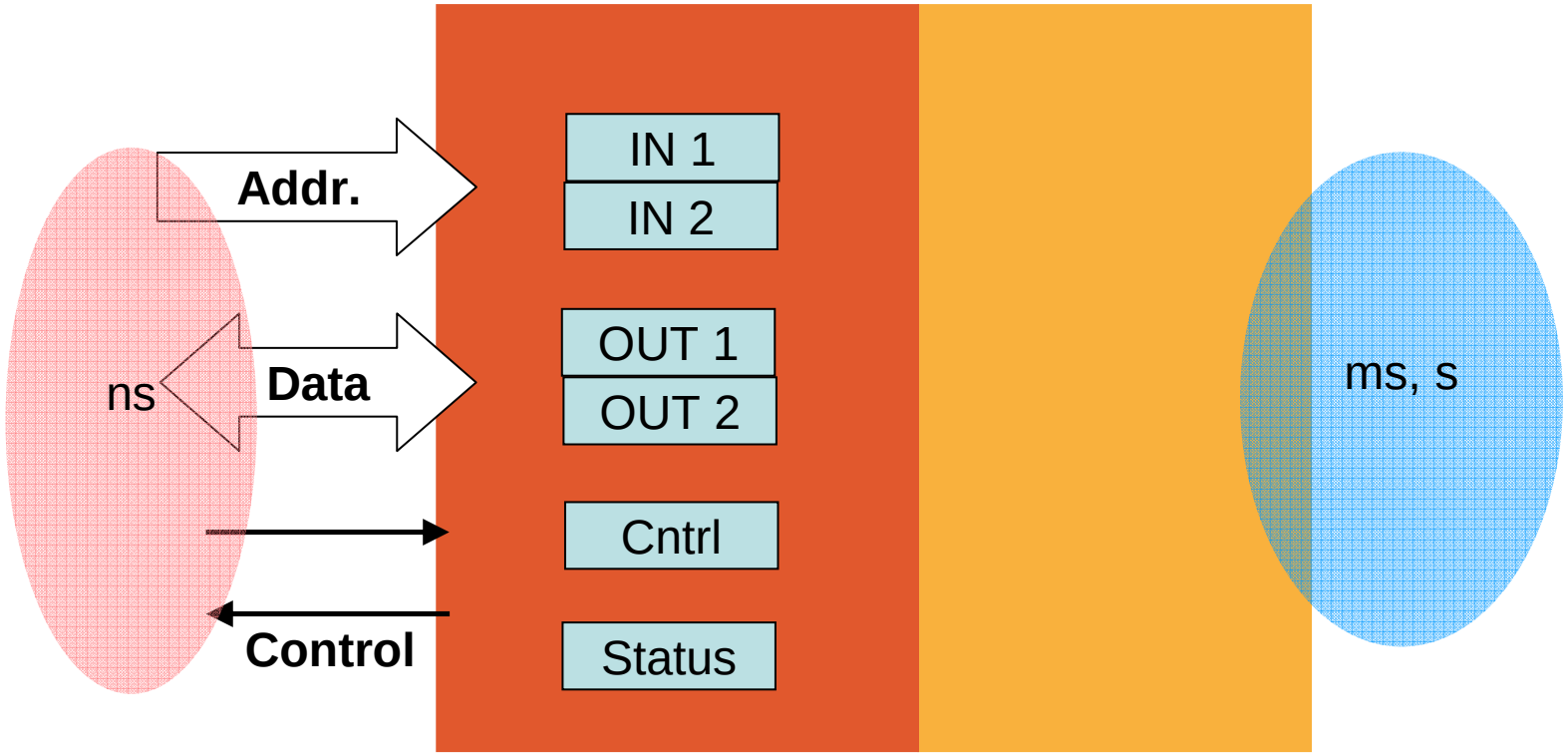


Segregated I/O
Separated I/O buses



I/O Fundamentals

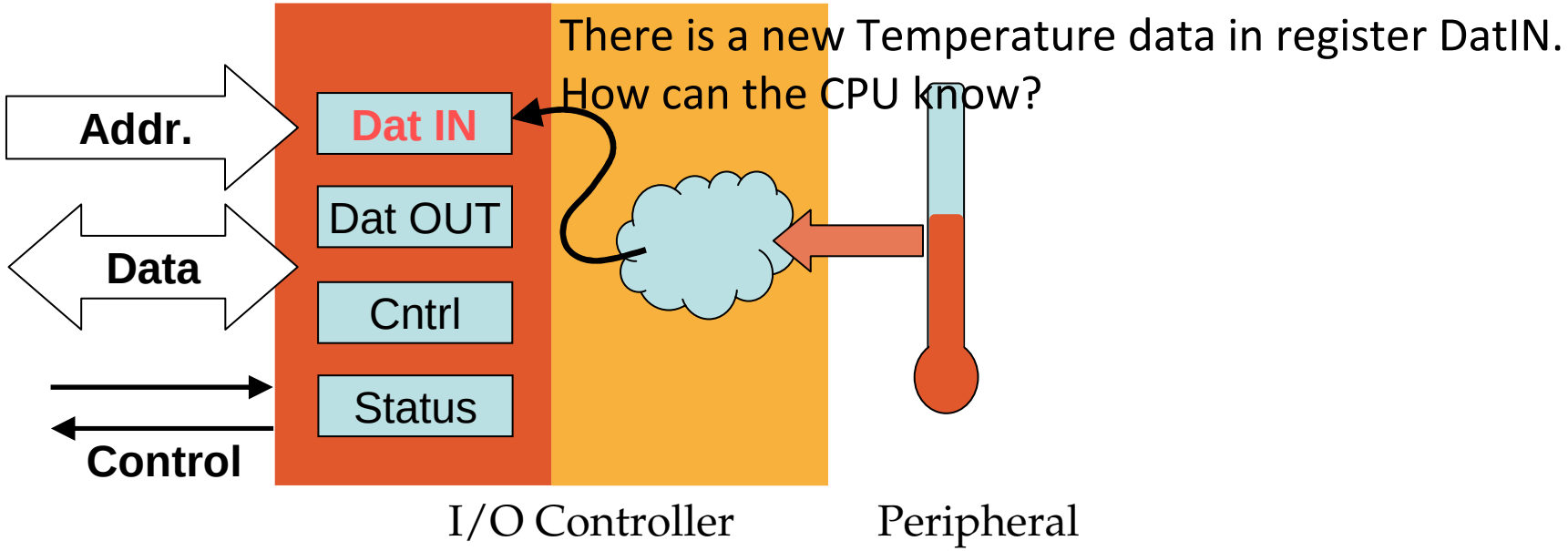
Buffering



I/O Controller

I/O Fundamentals

I/O Synchronization



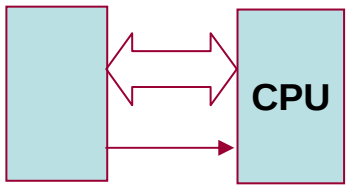
Data Exchange methods

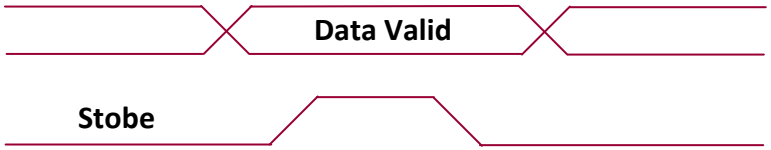
Synchronization with program execution

I/O Fundamentals

Data Exchange: Hardware

Open Loop Transfers

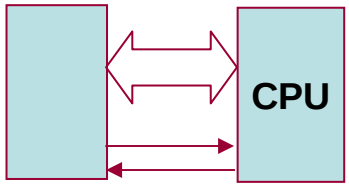


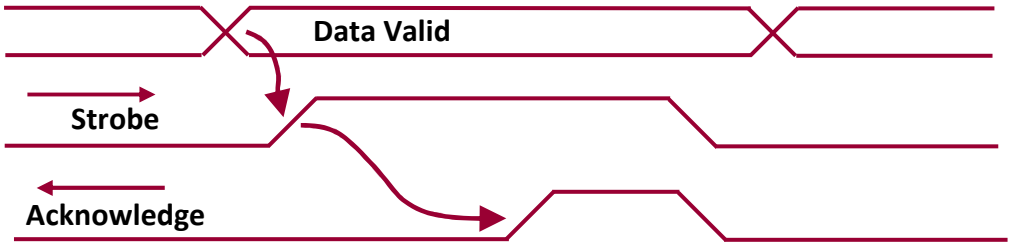


I/O Fundamentals

Data Exchange: Hardware

Handshaking

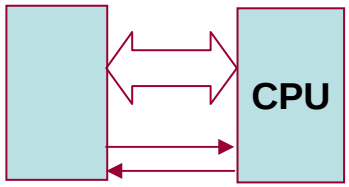


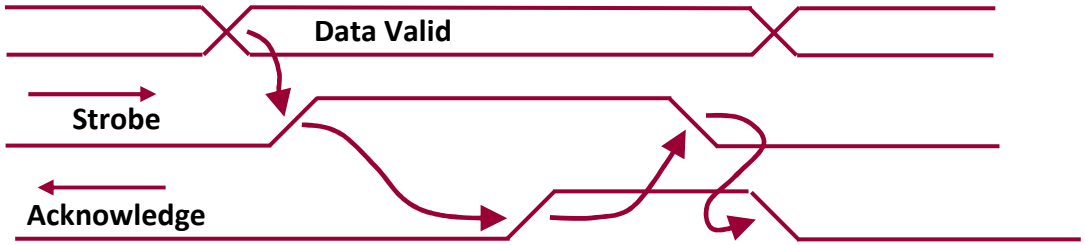


I/O Fundamentals

Data Exchange: Hardware

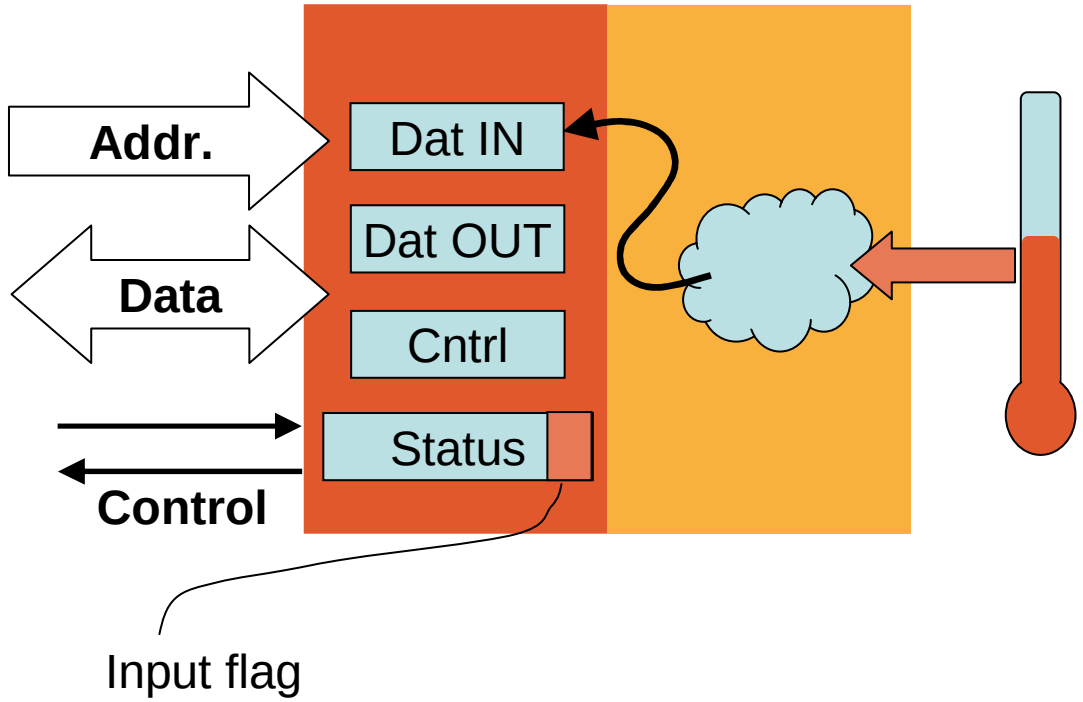
Fully-Interlocked Handshaking





I/O Fundamentals

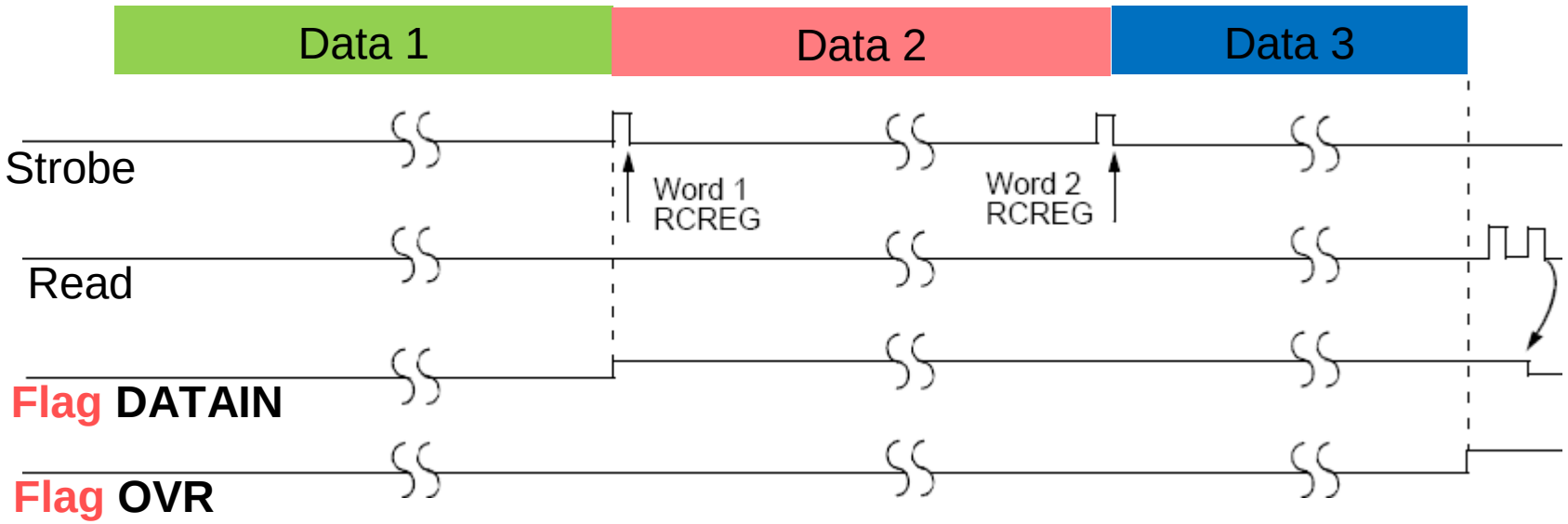
Data Exchange: Hardware



I/O Fundamentals

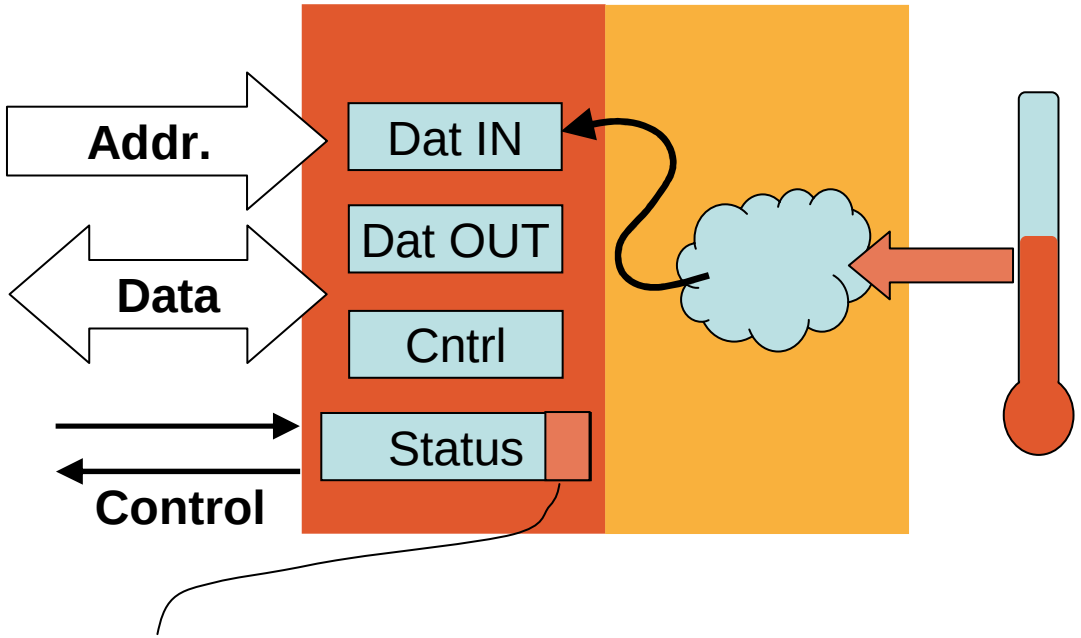
Data Exchange: Hardware

FLAGS



I/O Fundamentals

I/O Synchronization with the Program execution

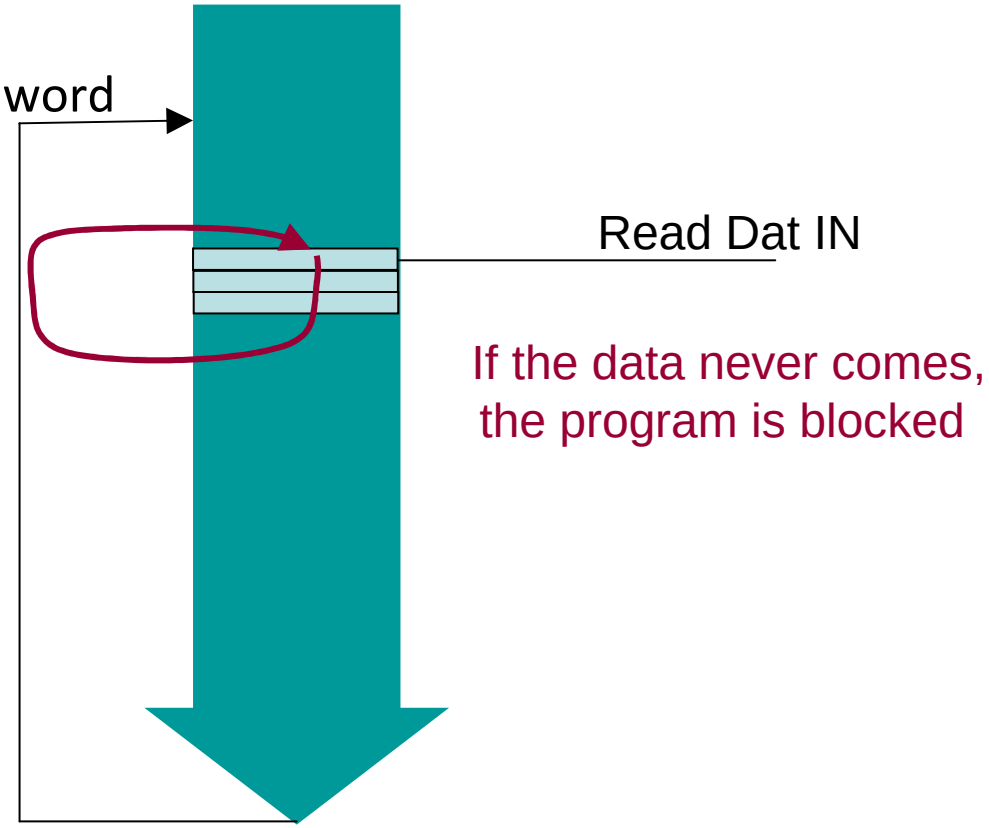


OK! An Input flag is set, but . . . Who cares?

I/O Fundamentals

Programmed driven I/O - Polling

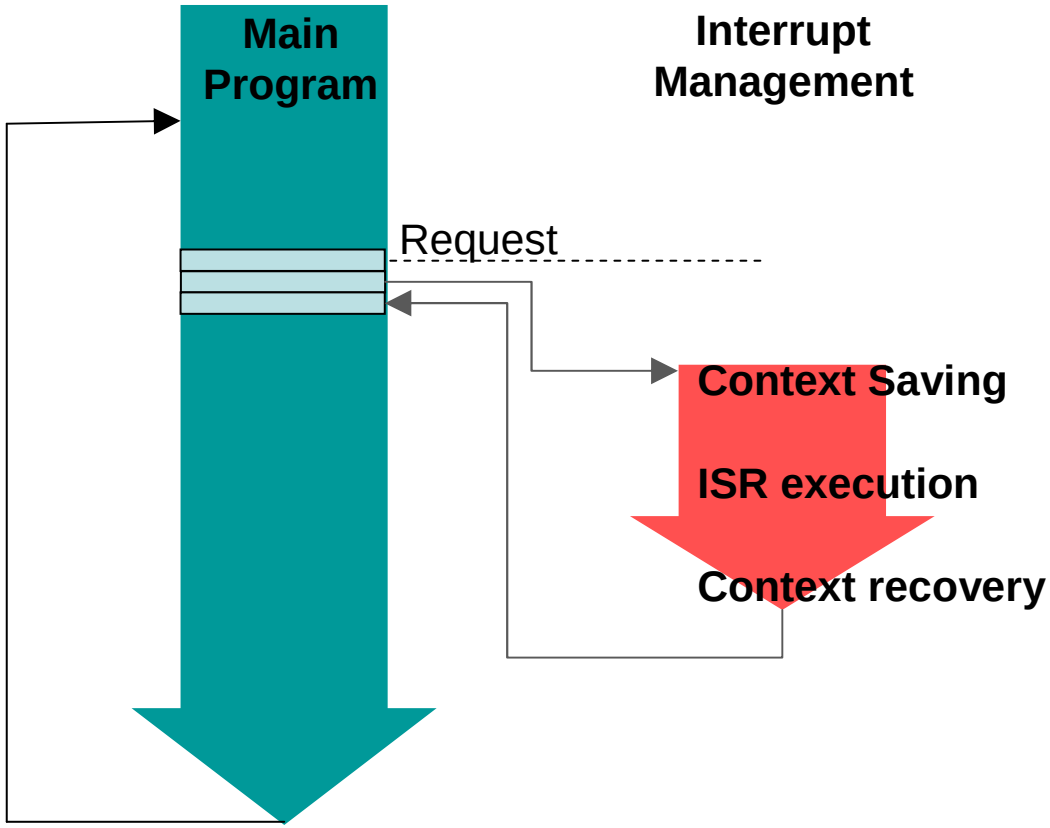
The CPU continously finds
the state of the flag
by reading the STATUS word



I/O Fundamentals

Interrupt driven I/O

The I/O controller sends a request to the CPU for attention (Interrupt)
The CPU executes a particular code (Interrupt Service Routine) in response to the interrupt



I/O Fundamentals

Interrupt driven I/O

Resources for the interrupt management:

Temporal storage for the PC	Stack
Location of the appropriate ISR code	Vector Area
Return Instruction	RTFIE

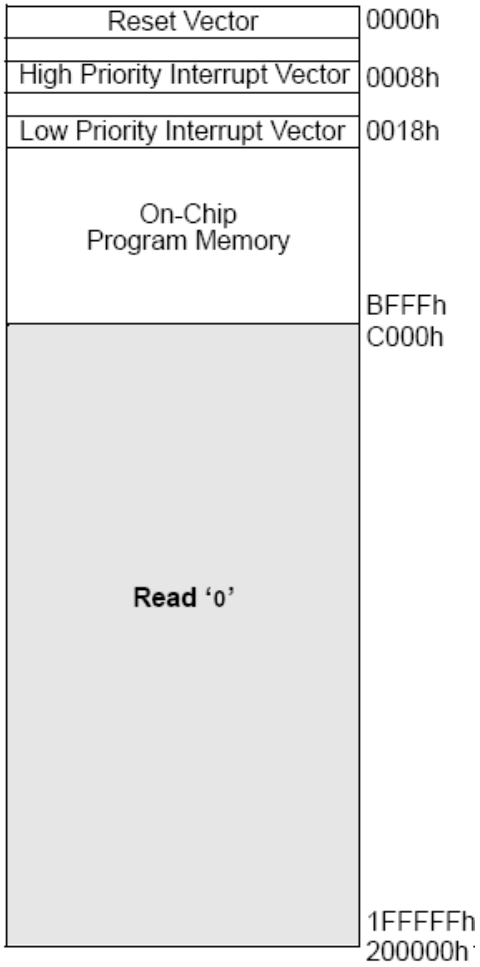
For external interrupts

Request Line	INT / IRQ

Figura del “PIC18F2525/2620/4525/4620 Data Sheet”
Con permiso de MICROCHIP

PIC Interrupts

PIC Interrupt management



Multiple interrupt sources

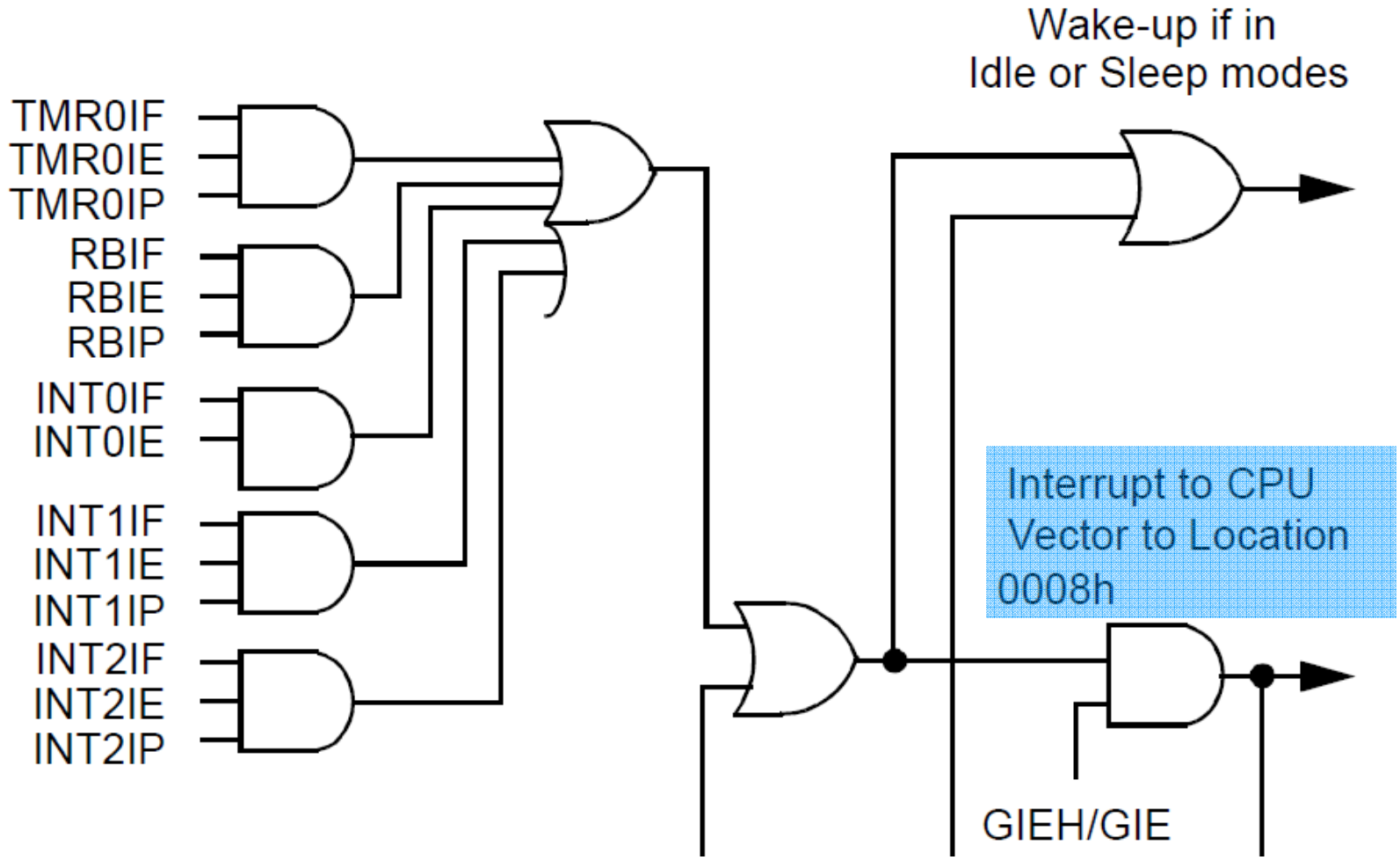
Interrupts with priority
IPEN bit (RCON<7>) = 1

We can assign either High (high priority, interrupt vector in 0008h) or Low low priority level, Interrupt vector is at 0018h) priority to each source.

There are 10 registers related to the interrupt control

PIC Interrupts

PIC Interrupt management



PIC Interrupts

PIC Interrupt management

Each Interrupt source has three bits associated to them:

- **Flag bit** saves the information that there has been an int request
- **Enable bit** which enables the interrupt to reach the CPU
- **Priority bit** to assign the priority level to the interrupt (high o low).

In addition to the above, there are General Enable bits

GIEH bit (INTCON<7>)=1 enable high priority interrupts.

GIEL bit (INTCON<6>)=1 enable low priority interrupts.

Figura del “PIC18F2525/2620/4525/4620 Data Sheet”
Con permiso de MICROCHIP

Atención a Interrupción en el PIC18

REGISTER 10-1: INTCON: INTERRUPT CONTROL REGISTER

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/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts

When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Note: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Interrupt Service Routines (ISR)

1

```
// definition of functions
void low_isr(void);
void high_isr(void);
```

2

```
// load the special address in program memory
#pragma code low_vector=0x18
void interrupt_at_low_vector(void)
{
_asm GOTO low_isr _endasm
}

#pragma code /* return to the default code section */
```

3

```
// write the ISR functions
#pragma interruptlow low_isr
void low_isr (void)
{
/* ... */
}
```