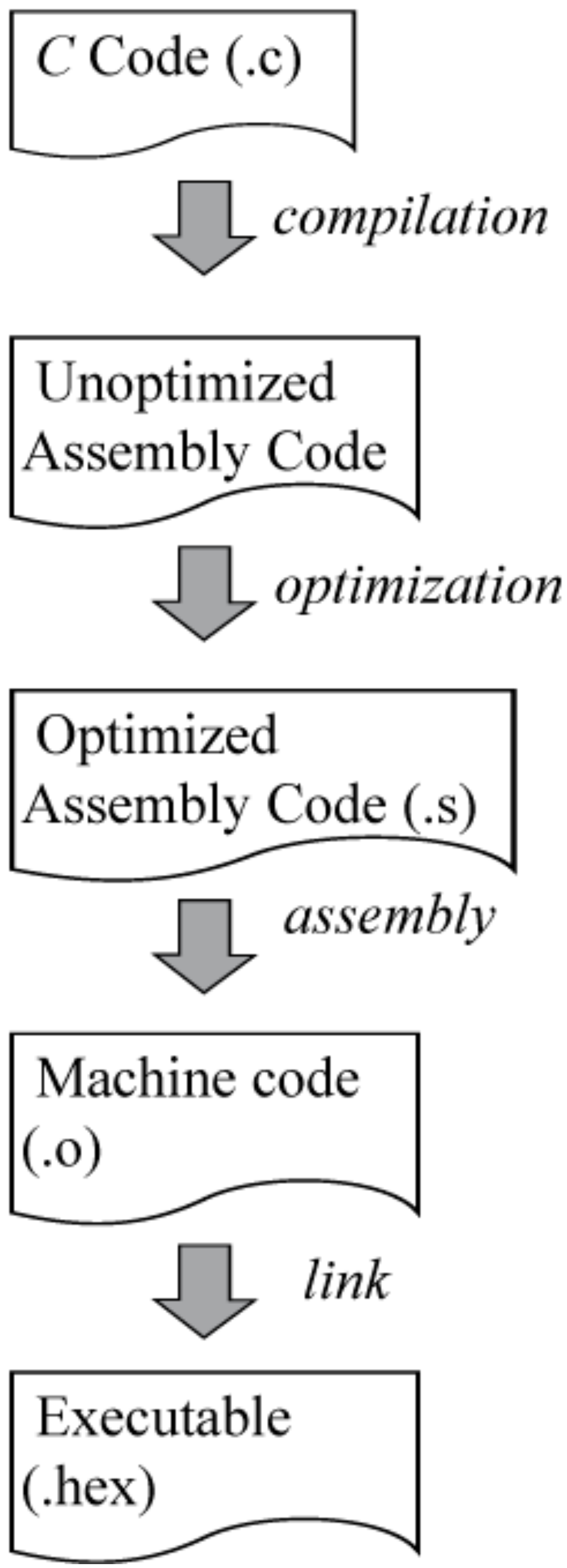
Chapter 7

I/O Ports, Reset, and Watchdog Timer

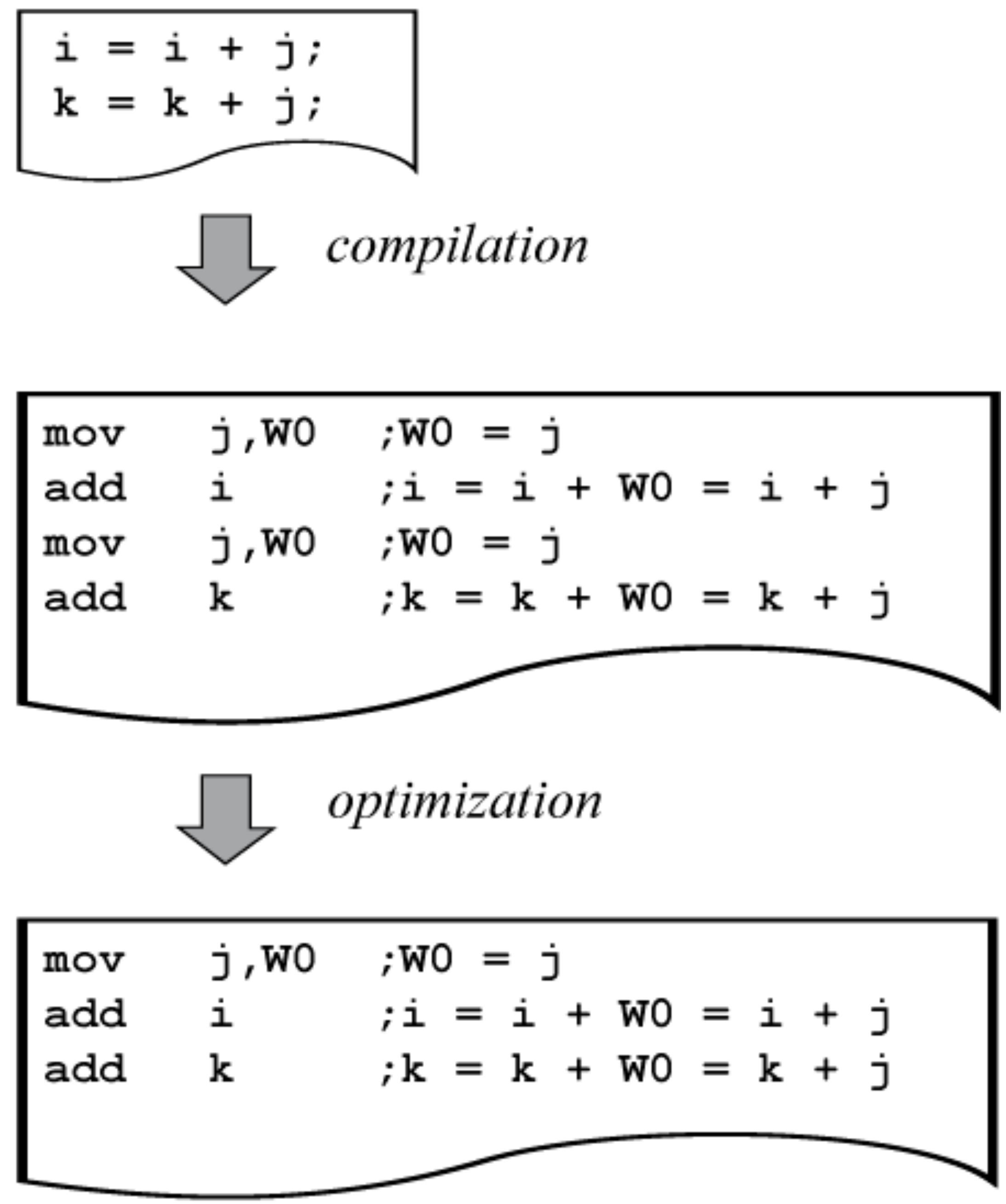
Yu Luo
Assistant Professor
Department of Electrical and Computer Engineering
Office: 239 Simrall Engineering Building
Email: yu.luo@ece.msstate.edu

C Compilation

From .c to .hex



Example Optimization



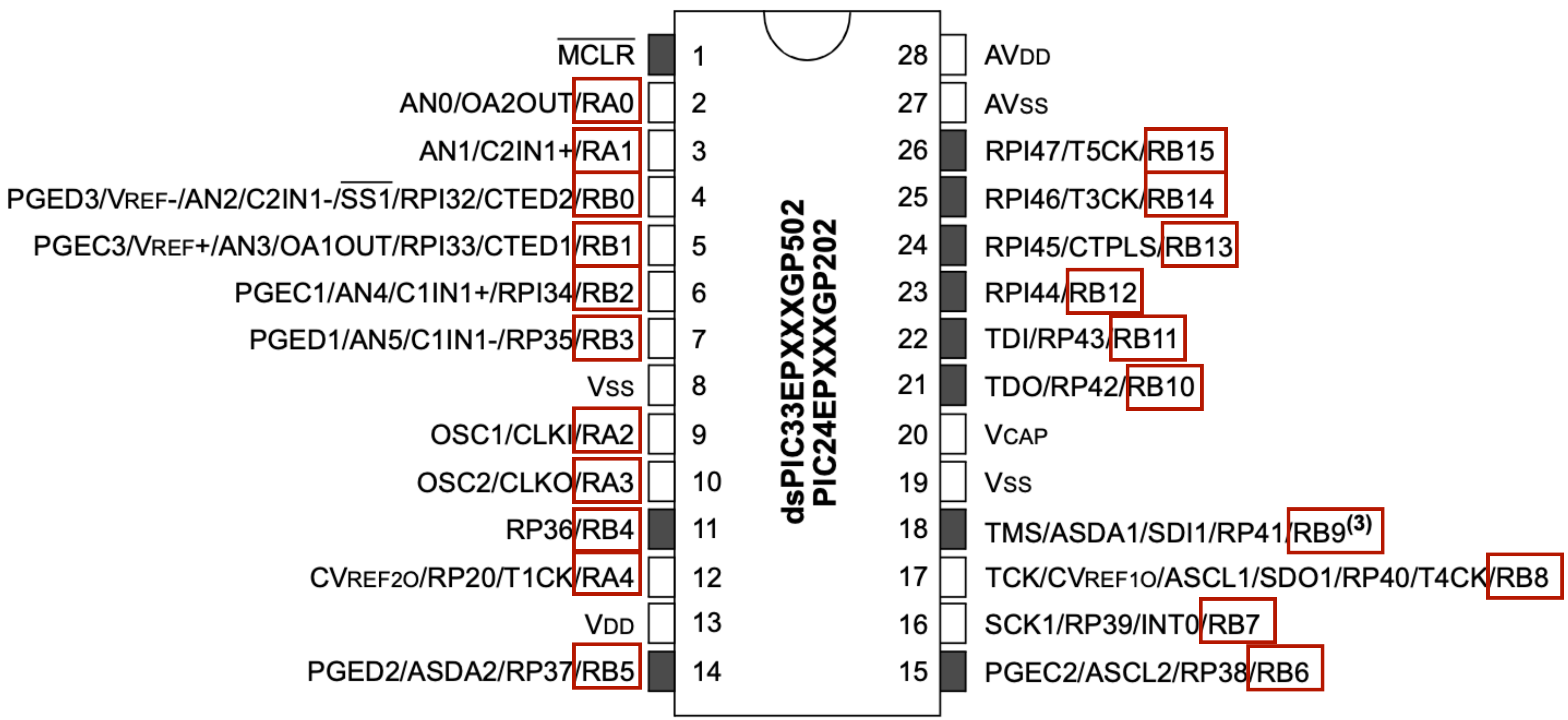
W0 already contains `j`,
remove second `mov` instruction

General-purpose I/O (GPIO)

- A PIC µC can have multiple I/O ports
- Each I/O port can have multiple independent GPIO pins

A. PORTA: pins RA0 to RA4

B. PORTB: pins RB0 to RB15



Input and Output (1)

- Each I/O pin can either be an input (read) or output (write)
- Input or output is controlled from the corresponding bit in the TRIS register
  A. Set the bit #n in TRIS*x* to 1 —> pin #n of PORT*x* will be input
  B. Clear the bit #n in TRIS*x* to 0 —> pin #n of PORT*x* will be output

Example 1

Set RB3 and RB5 as inputs, other pins in PORTB as outputs

↓

Set bit #3 and #5 in TRISB to 1 ⟶ TRISB = 0b0000 0000 0010 1000

bit #5 bit #3

Example 2

Set RA0 and RA2 as inputs, other pins in PORTA as outputs

↓

Set bit #0 and #2 in TRISA to 1 ⟶ TRISA = 0b0000 0000 0000 0101

bit #2 bit #0

Input and Output (2)

- If a GPIO is configured as an output, output 0 (low voltage) or 1 (high voltage) is controlled from the corresponding bit in the PORT register
  A. Set the bit #n in PORT*x* to 1 —> pin #n of PORT*x* will output 1
  B. Clear the bit #n in PORT*x* to 0 —> pin #n of PORT*x* will output 0

Example

Configure PIC to let (a) RB15 and RB13 output 1; and (b) other GPIO as input

Step 1

Configure RB15 and RB13 as output, other GPIO as input

↓

Set bit #13 and #15 in TRISB to 0 ⟶ TRISB = 0b0101 1111 1111 1111

bit #15 bit #13

Step 2

Let RB15 and RB13 output 1

↓

Configure RB15 and RB13 as output, other GPIO as input

↓

Set bit #13 and #15 in PORTB to 1 ⟶ PORTB = 0b1010 0000 0000 0000

bit #15 bit #13

Input and Output (3)

- If a GPIO pin is configured as an input, corresponding bit in the PORT register will be the value read from the pin

Example

0b0000 0000 0000 0001

A. Test if RB0 is high (i.e., input = 1): if ((PORTB & 0x0001) == 1)

B. Test if RA3 is low (i.e., input = 0): if ((PORTA & 0x0008) == 0)

0b0000 0000 0000 1000

C. Test if RB0 to RB3 are all high: if ((PORTB & 0x000F) == 0x000F)

0b0000 0000 0000 1111

Input and Output (3)

- Using PORT register is convenient to write multiple output pins or read multiple input pins at the same time

- If we just want to read or write a single output pin, then we can directly use the name of the pin in the PORT register, e.g., _RA2, _RB3, to do it

Example

A. Assume all GPIOs in PORTB are set as output, then
   _RB5 = 1 —> Let RB5 output 1 (high voltage)

B. Assume all GPIOs in PORTA are set as input, then
   _RA0, means read input value from RA0 pin

LAT*x* vs PORT*x* (1)

- LATx register holds the last value written to PORTx

- If the GPIO is configured as **output**, then writing LATx is the same as writing PORTx

Example

Assume RB0 has been configured as output, then either _RB0 = 1 or LATB0 = 1 can let RB0 pin output 1 (high voltage)

- If the GPIO is configured as **input**, then:

  A. Reading LATx is reading the last value written to PORTx

  B. Reading PORTx is reading the voltage at the physical pin

LAT*x* vs PORT*x* (2)

About difference between LATx and PORTx in the input mode. Assume RB3 is tied to ground, and then we do following things:

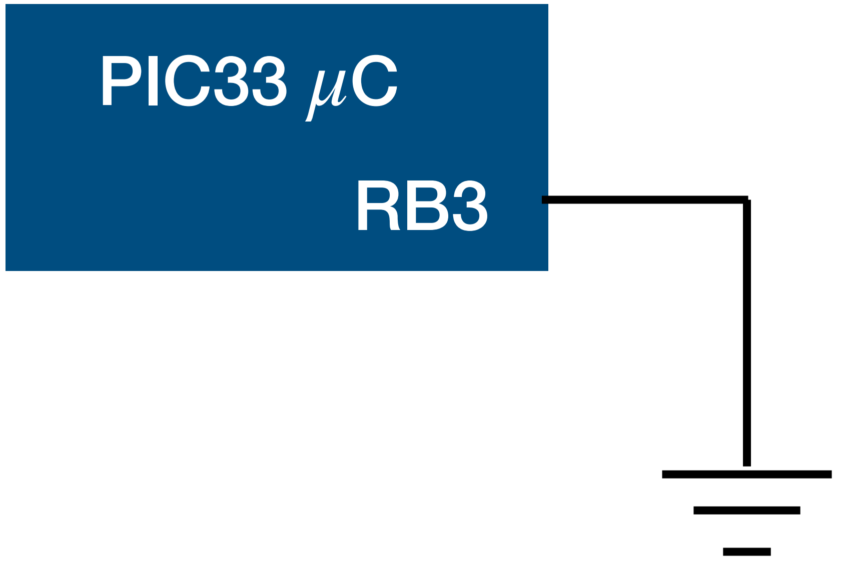
Step 1: Configure RB3 as output, then let it output 1

```
TRISB = 0b1111 1111 1111 0111;  // Configure RB3 as output
_RB3 = 1;                       // Let RB3 output 1
```

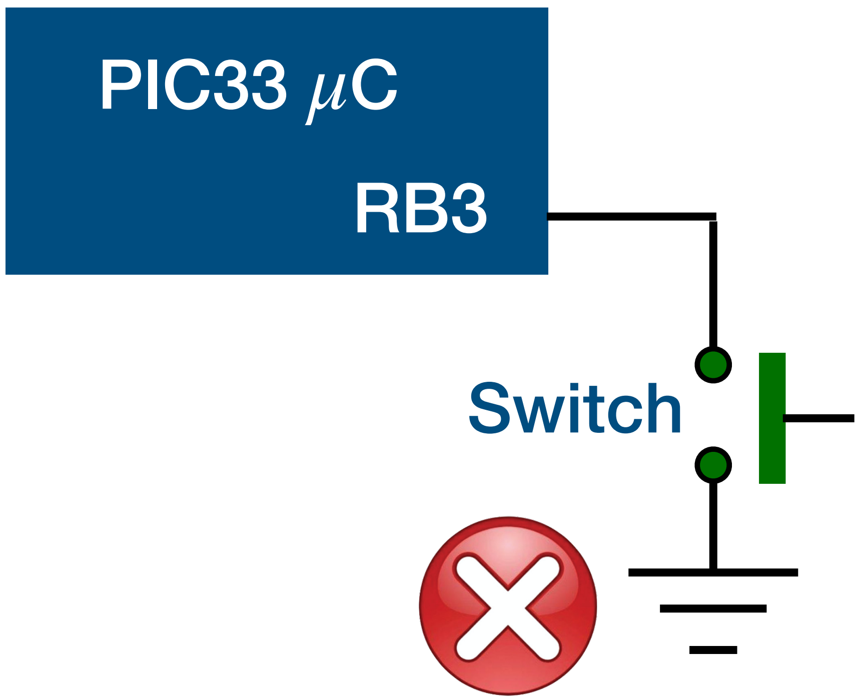
This will set bit #3 in PORTB to 1

Step 2: Reading LATB3 —> reading the last value write to the bit #3 in PORTB —> return 1

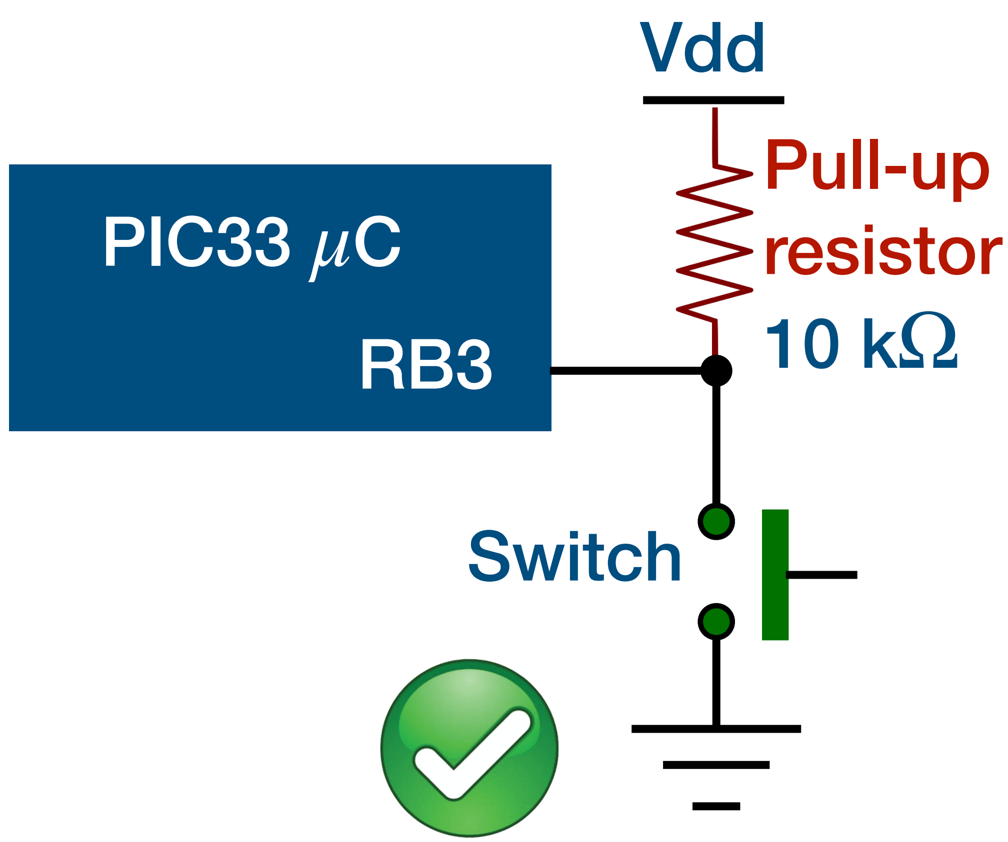
Step 3: Reading _RB3 —> reading the voltage at the physical pin RB3 —> return 0 because RB3 is tied to ground



Switch Input



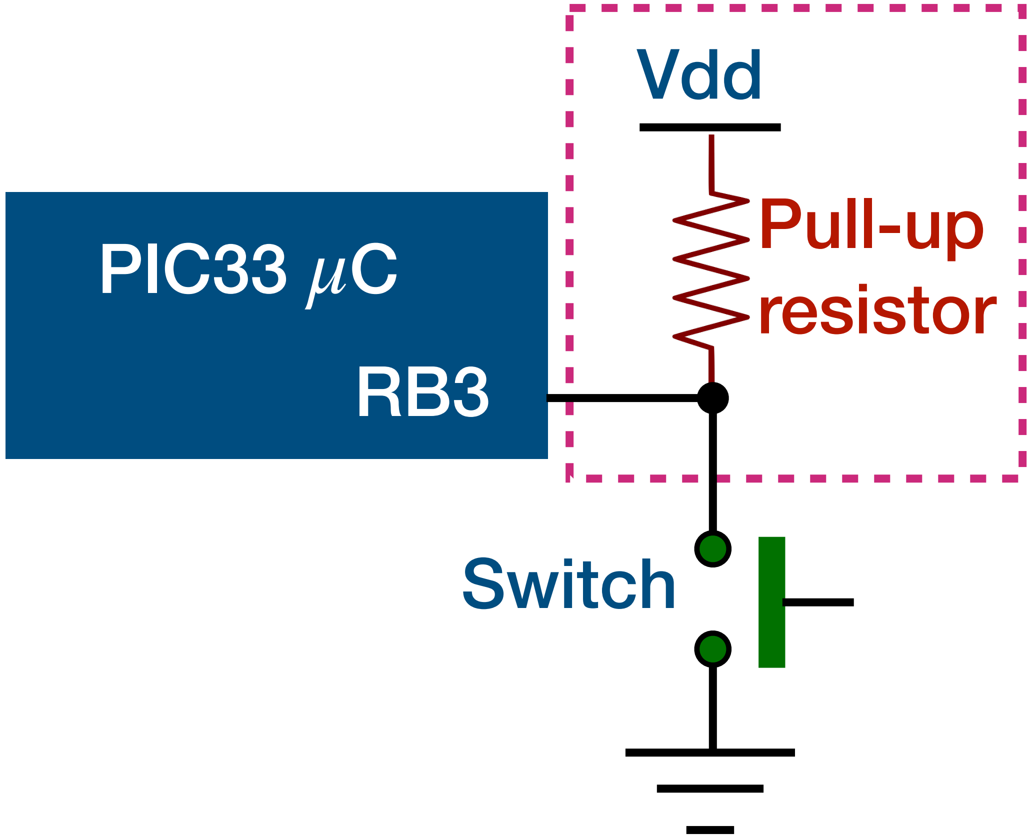
- When switch is pressed —> RB3 is tied to ground —> _RB3 reads as 0
- When switch is released —> RB3 floats —> _RB3 reads as a random value



- When switch is pressed —> RB3 is tied to ground —> _RB3 reads as 0
- When switch is released —> RB3 is connected to VCC through pull-up resistor —> _RB3 reads as 1

It prevents shortcut between Vdd and ground when switch is pressed

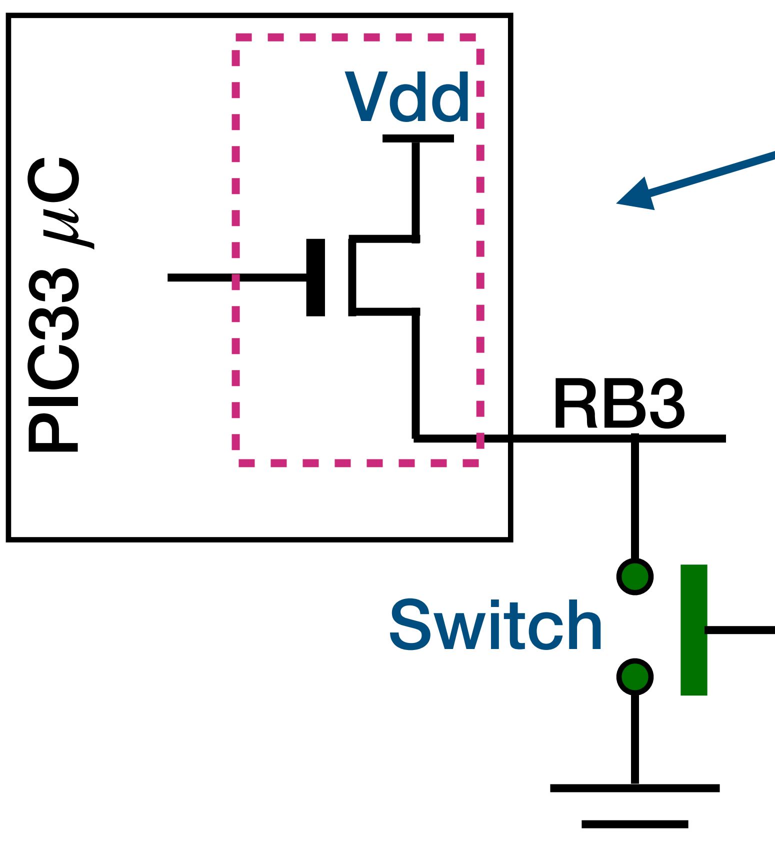
Internal Weak Pull-Up Inputs



- To avoid pin float in the air, we need to connect pin through an external pull-up resistor and external power supply (Vdd)

Not convenient

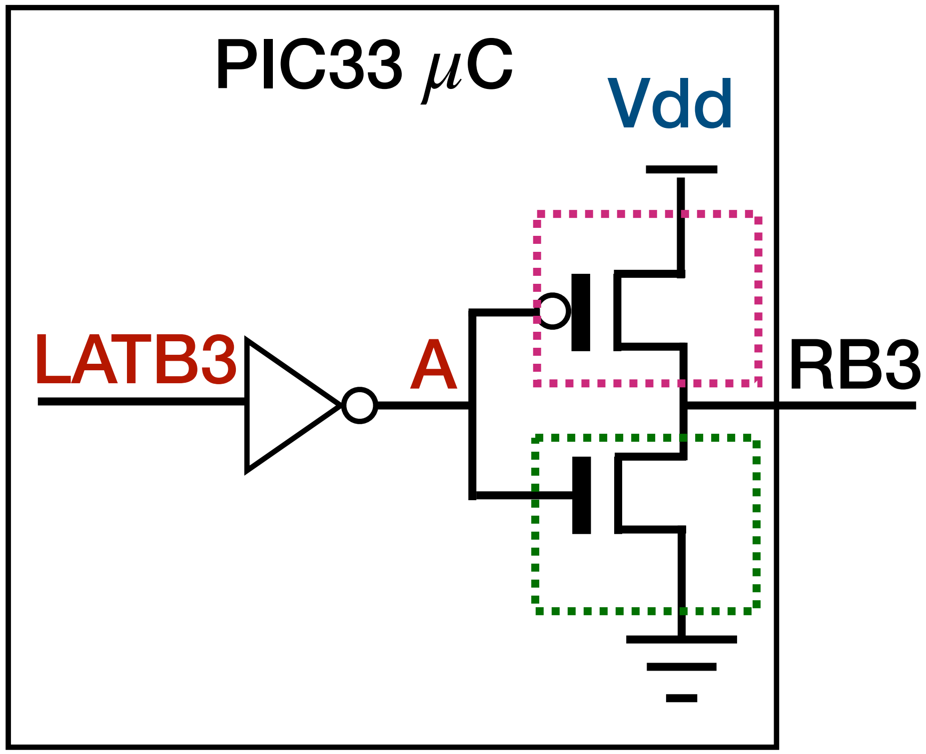
- GPIO pins in PIC33 supports internal weak pull-up, which integrates the internal pull-up resistor and internal power supply in the chip



It is called weak pull-ups because PMOS transistor with high resistance is used as the internal pull-up resistor, which draw only a little current through the pin.

Output circuit

- When a GPIO is configured as common output, its inside looks like this:



A. LATB3 = 0 —> Point A = 1 —> Upper transistor is open and lower transistor is closed —> RB3 is tied to ground through lower transistor —> RB3 outputs low voltage (0)

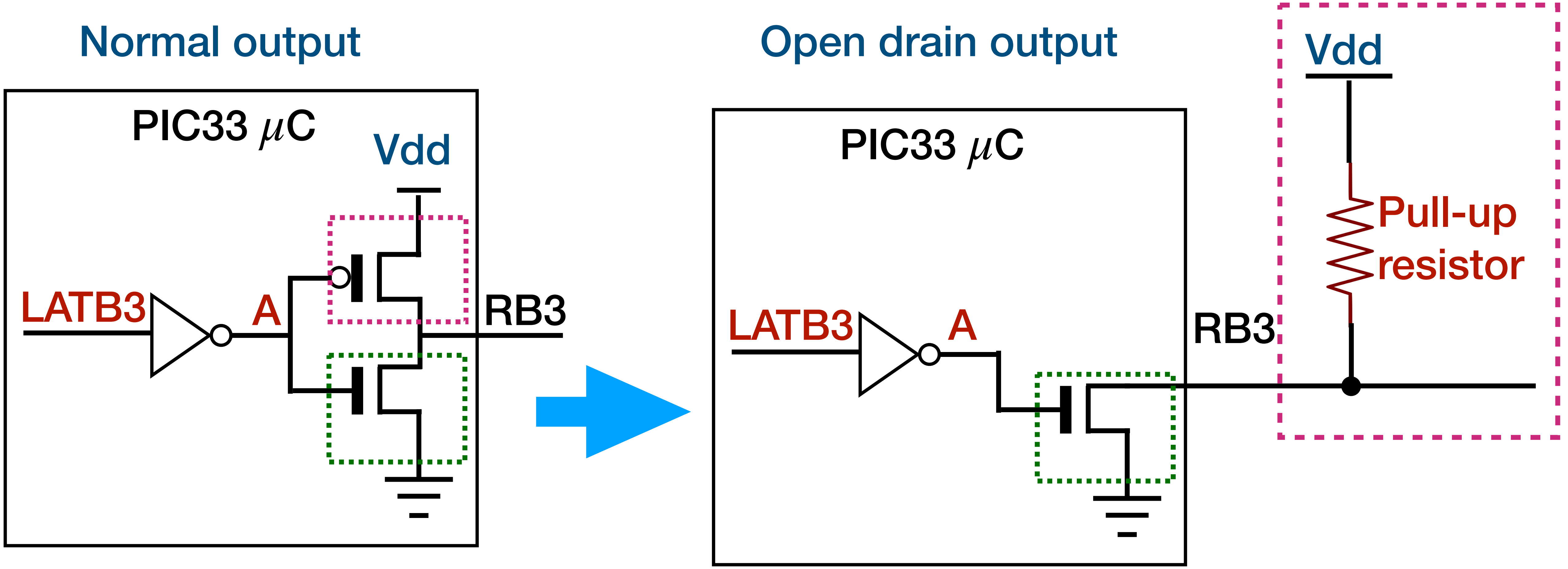
B. LATB3 = 1 —> Point A = 0 —> Upper transistor is closed and lower transistor is open —> RB3 connects to Vdd through upper transistor —> RB3 outputs high voltage (1)

Open Drain Outputs (1)

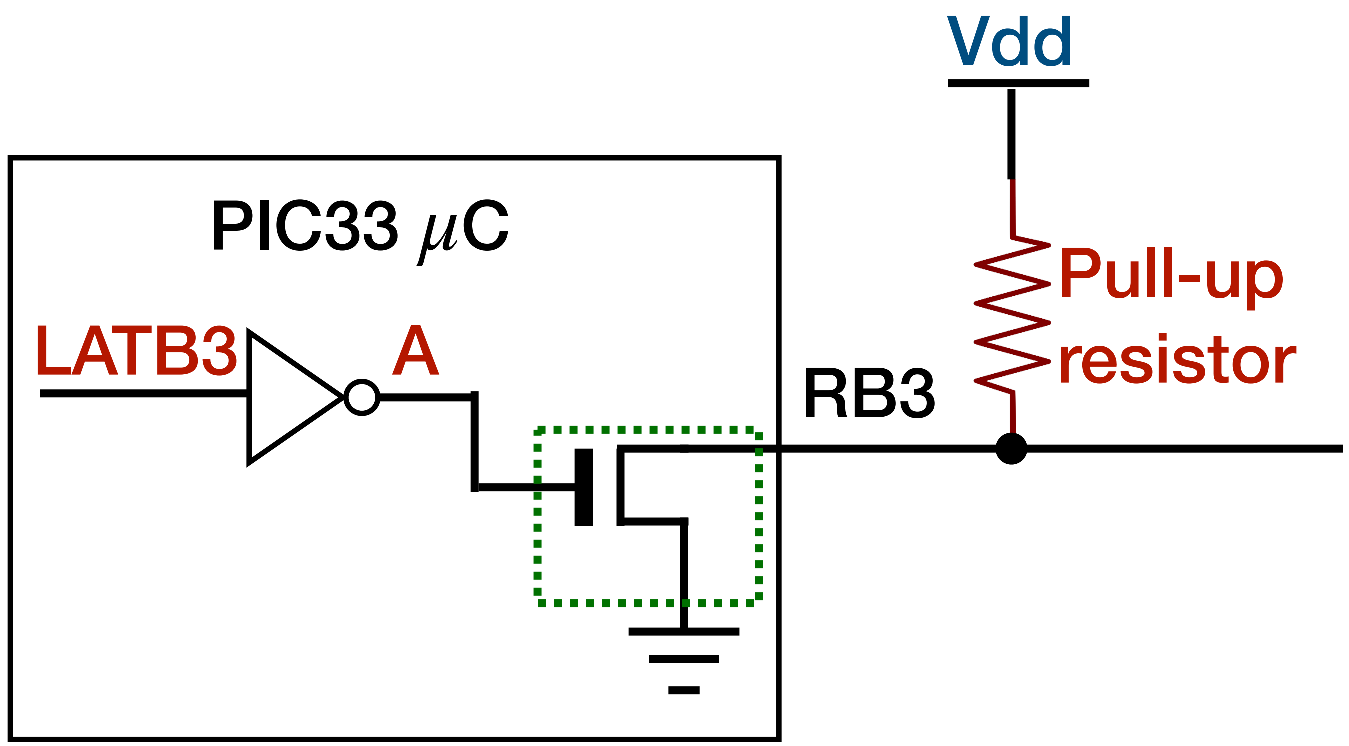
- PIC also supports another output mode, called open drain output
- In open drain output, internal power supply (Vdd) and internal pull-up resistor (upper transistor) is disabled

↓

We need to provide external power supply and external pull-up resistor



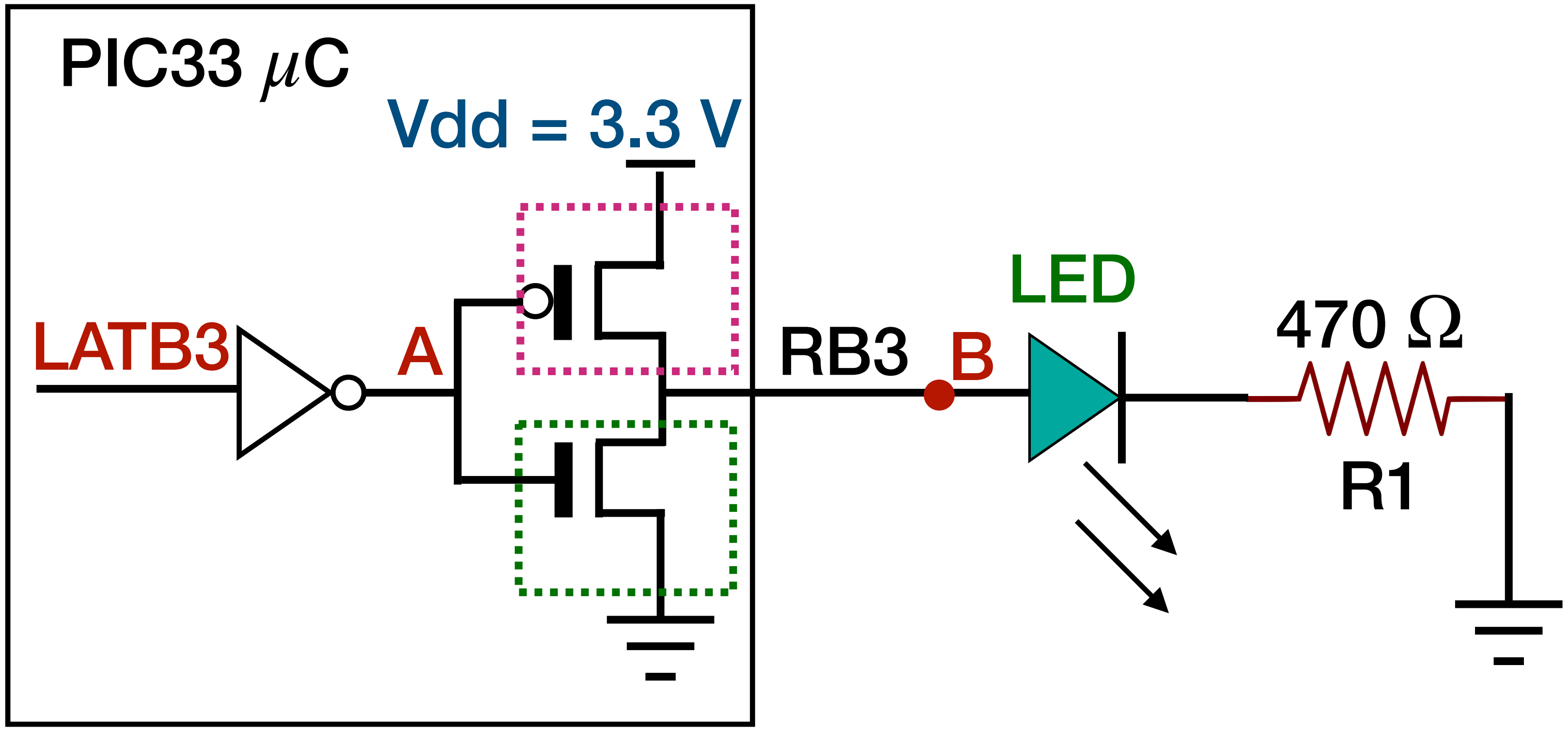
Open Drain Outputs (2)



A. LATB3 = 0 —> Point A = 1 —> Transistor is open —> RB3 is tied to ground through the transistor —> RB3 output low voltage (0)

B. LATB3 = 1 —> Point A = 0 —> Transistor is closed —> RB3 connects to the external Vdd through external pull-up resistor —> RB3 output high voltage (1)

Driving LED in Normal Output Mode

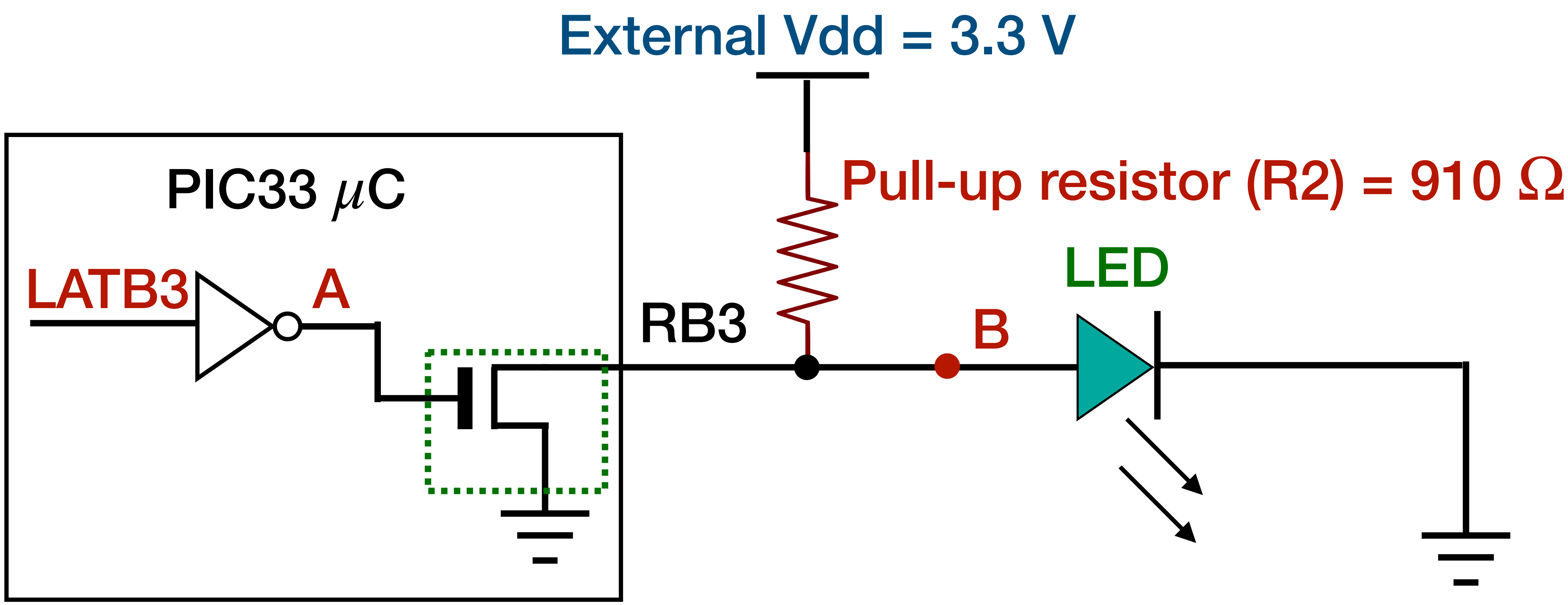


A. LATB3 = 0 —> Point A = 1 —> Upper transistor is open and lower transistor is closed —> RB3 is tied to ground through lower transistor —> Voltage at point B = 0 —> LED is turned off

B. LATB3 = 1 —> Point A = 0 —> Upper transistor is closed and lower transistor is open —> RB3 connects to Vdd through upper transistor —> Voltage at point B = Vdd (3.3 V) —> LED is turned on

C. The resistor R1 is applied to limit the current went through the LED; otherwise LED may be damaged.

Driving LED in Open Drain Mode

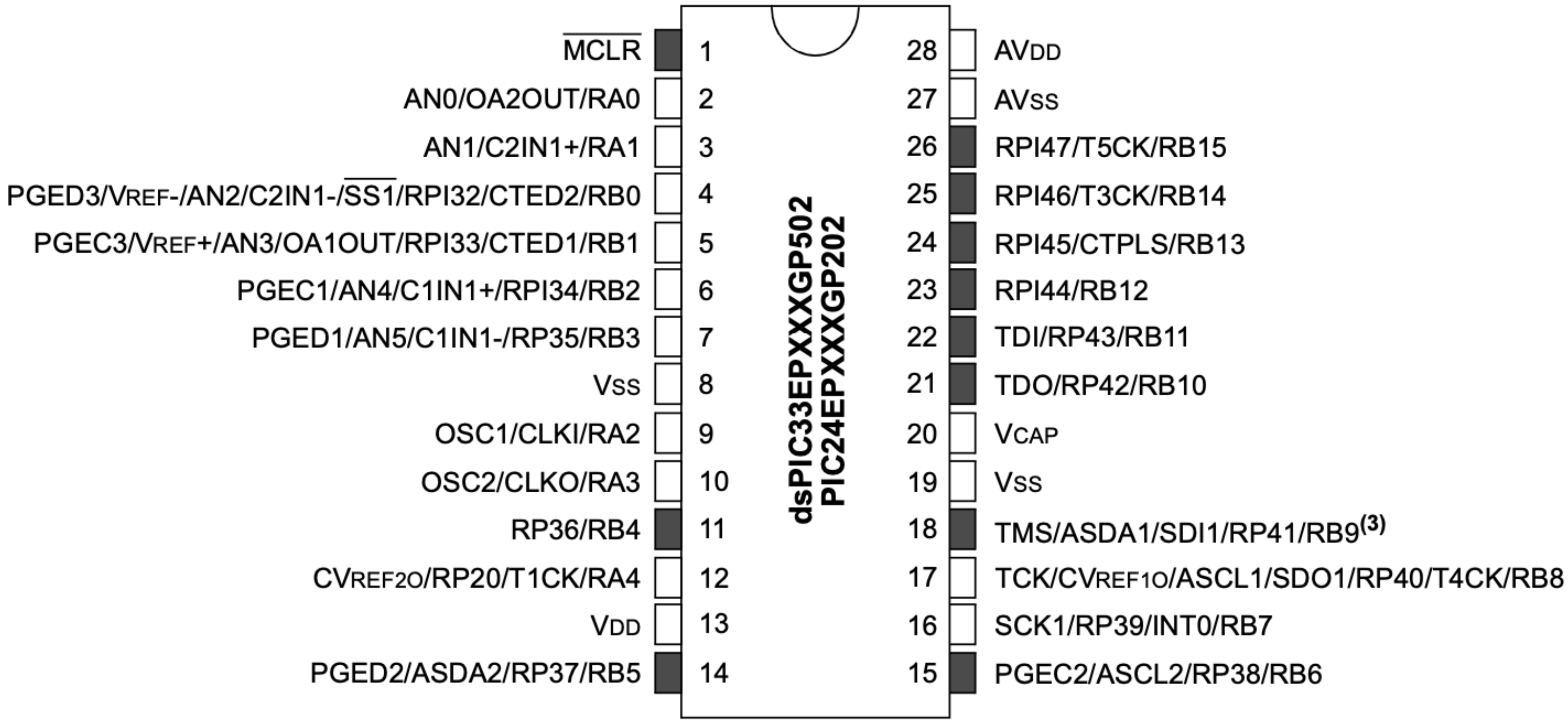


A. LATB3 = 0 —> Point A = 1 —> Transistor is open —> RB3 is tied to ground through the transistor —> Voltage at point B = 0 —> LED is turned off

B. LATB3 = 1 —> Point A = 0 —> Transistor is closed —> RB3 connects to the external Vdd through the external pull-up resistor —> Voltage at point B = Vdd (3.3 V) —> LED is turned on

C. The pull-up resistor R2 is applied to limit the current went through the LED and RB3; otherwise PIC and LED may be damaged.

Analog/Digital pin vs Digital pin



A. Most pins in PIC have multiple functions

B. Pins (e.g. AN0 and AN1) with shared analog/digital functions have a maximum input voltage of Vdd + 0.3 V = 3.3 V + 0.3 V = 3.6 V

C. “Digital-only” pins have a maximum input voltage of 5.6 V

D. Most GPIO pins can only source (output) or sink (input) a maximum 4 mA.
—> Adding a resistor to restrict the current can protect I/O port.

Port Configuration Macros (1)

- Most pins in PIC have multiple functions —> Enable target function while disable all other functions

Example: Configure RB15 as digital output

```
; Configure PORTB as output
mov #0, W0
mov W0, _TRISB15
; Initialize PORTB15 to output zero
mov W0, _LATB15
; Disable PORTB analog
mov W0, _ANSB15
; Disable PORTB  open drain
mov W0, _ODCB15
```

Macros

CONFIG_RB15_AS_DIG_OUTPUT( );

Port Configuration Macros (2)

- Macros make pin configuration easy
- Macros for pin configurations are included in *pic24_ports.h* file.
- More examples about Macros for pin configurations:

A. ENABLE_RB15_PULLUP ( );

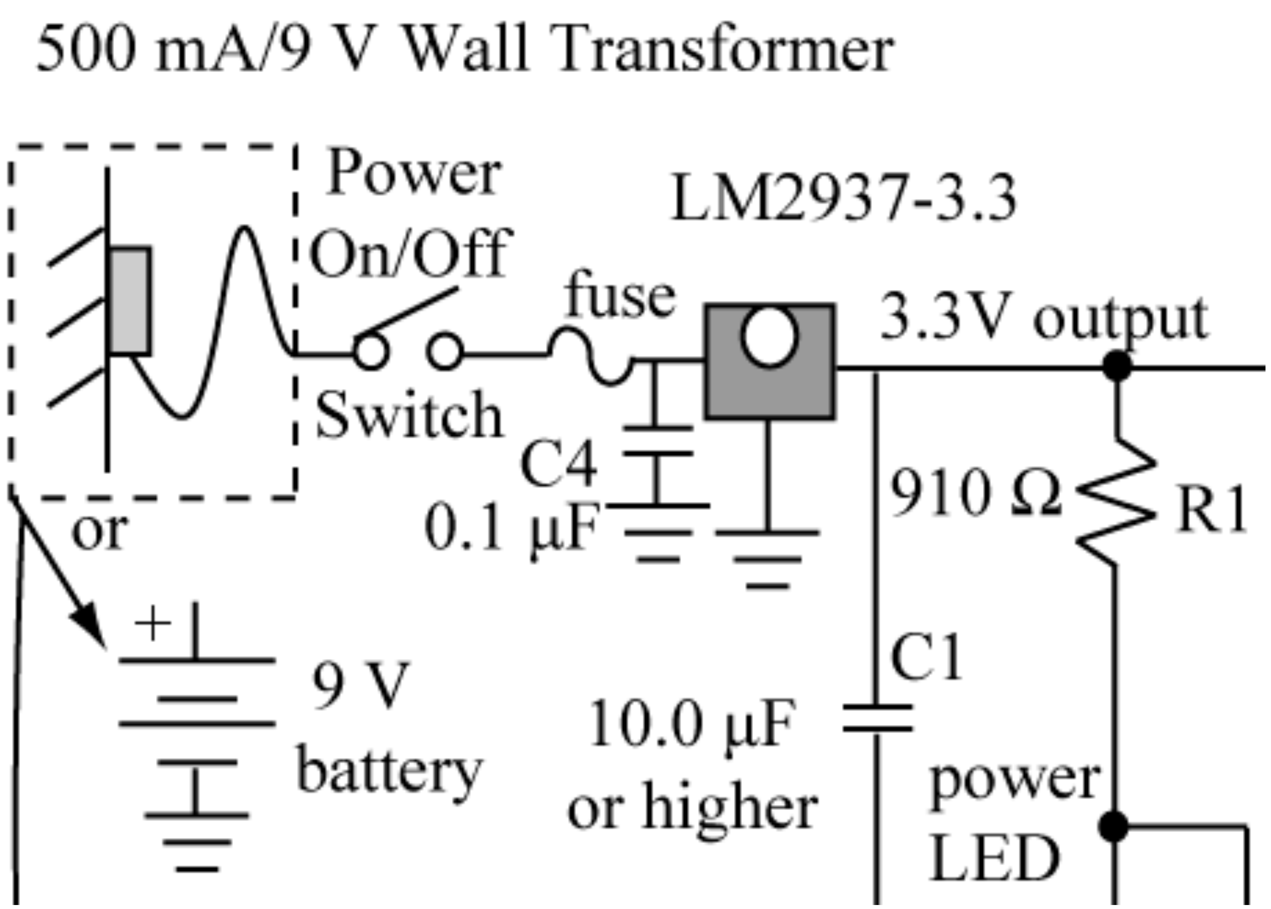
B. DISABLE_RB15_PULLUP ( );

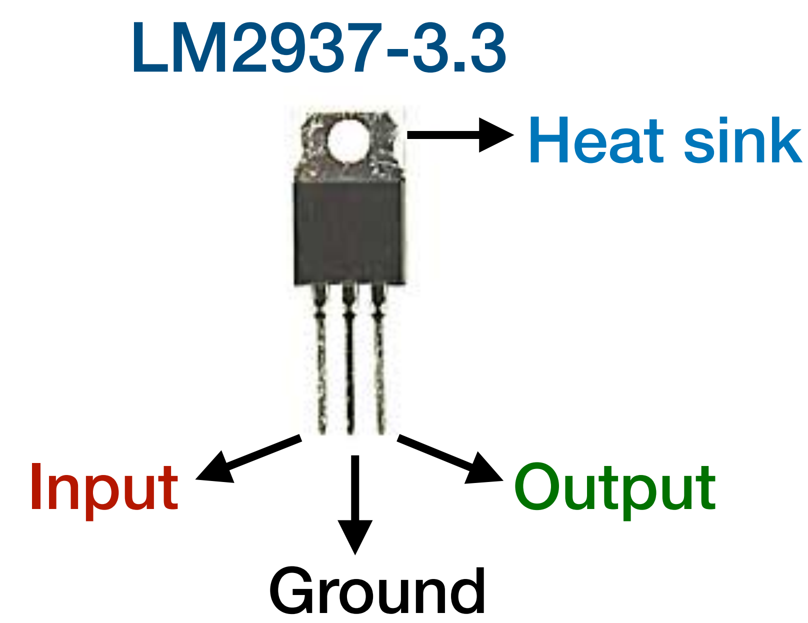
C. ENABLE_RB13_OPENDRAIN ( );

D. DISABLE_RB13_OPENDRAIN ( );

E. CONFIG_RB8_AS_DIG_OD_OUTPUT ( );

Powering PIC μC





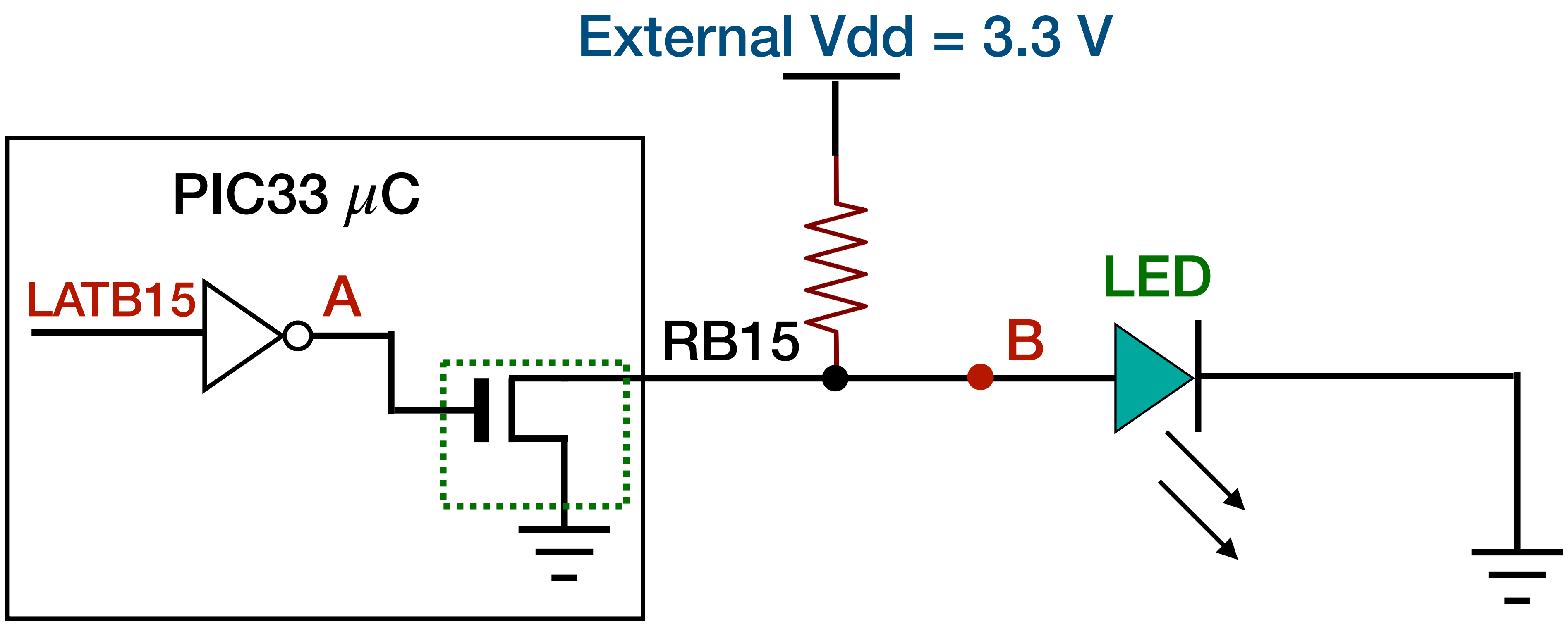
- A Wall transformer provides 6V unregulated DC voltage

  Voltage can vary significantly depending on current being drawn

- The LM2937-3.3 voltage regulator provides a regulated 3.3 V

  Voltage will stay stable up to maximum current rating of device

Flash LED — Hardware



A. LATB15 = 0 —> Point A = 1 —> Transistor is open —> RB15 is tied to ground through the transistor —> Voltage at point B = 0 —> LED is turned off

B. LATB15 = 1 —> Point A = 0 —> Transistor is closed —> RB15 connects to the external Vdd through the external pull-up resistor —> Voltage at point B = Vdd (3.3 V) —> LED is turned on

Flash LED — Method 1

```
# include "pic24_all.h"

void a_delay (void)
{
    uint16 u16_i, u16_k;
    for (u16_k = 1800;  - -u16_k; )
    {
        for (u16_i = 1200;  - -u16_i; )
    }
}
```

A simple delay function

```
void main (void)
{
    configClock;
    // Enable open drain
    _ODCB15 = 1;
    // Configure RB15 as output
    _TRISB15 = 0;
    // RB15 initially low (LED off)
    _LATB15 = 0;
    while (1)
    {
        // Let RB15 maintains current status a
           a certain time
        a_delay{};
        // Toggle RB15 output (Toggle LED)
        _LATB15 = !_LATB15;
    }
}
```

Flash LED — Method 2

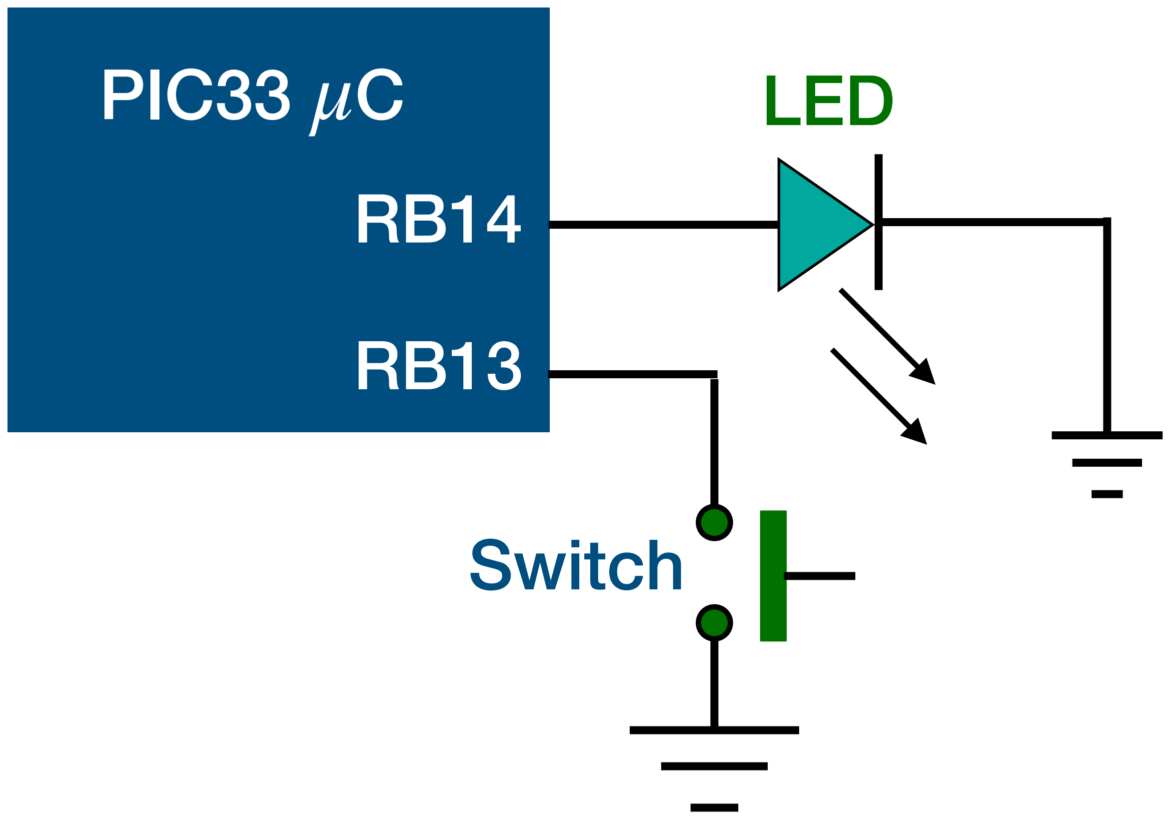
```
# include "pic24_all.h"

#define CONFIG_LED1 ()      CONFIG_RB15_AS_DIG_OD_OUTPUT
#define LED1      _LATB15
```

Use macros to improve code clarity

```
void main (void)
{
   configClock ();
   CONFIG_LED1 ();  // Marcos: configure RB15 as open drain output
   LED = 0;             // Initially turn off LED
   while (1)
   {
      DELAY_MS (250)  // LED maintains current status 250 ms
      LED1 = !LED1;   // Toggle LED
   }
}
```

This is a delay function defined in header file. We can call it directly.

Interact with Push Button (Hardware)



Implement following functions:

A. Press and release button: Turn on LED

B. Press and release button again: Turn off LED

C. Repeat the above two steps

Analysis:

- No external pull-up resistor and Vdd connect to RB13

↓

RB13 should be configured as weak pull-up input to detect status of switch

- No external pull-up resistor and Vdd connect to RB14

↓

RB14 should be configured as normal output to control LED

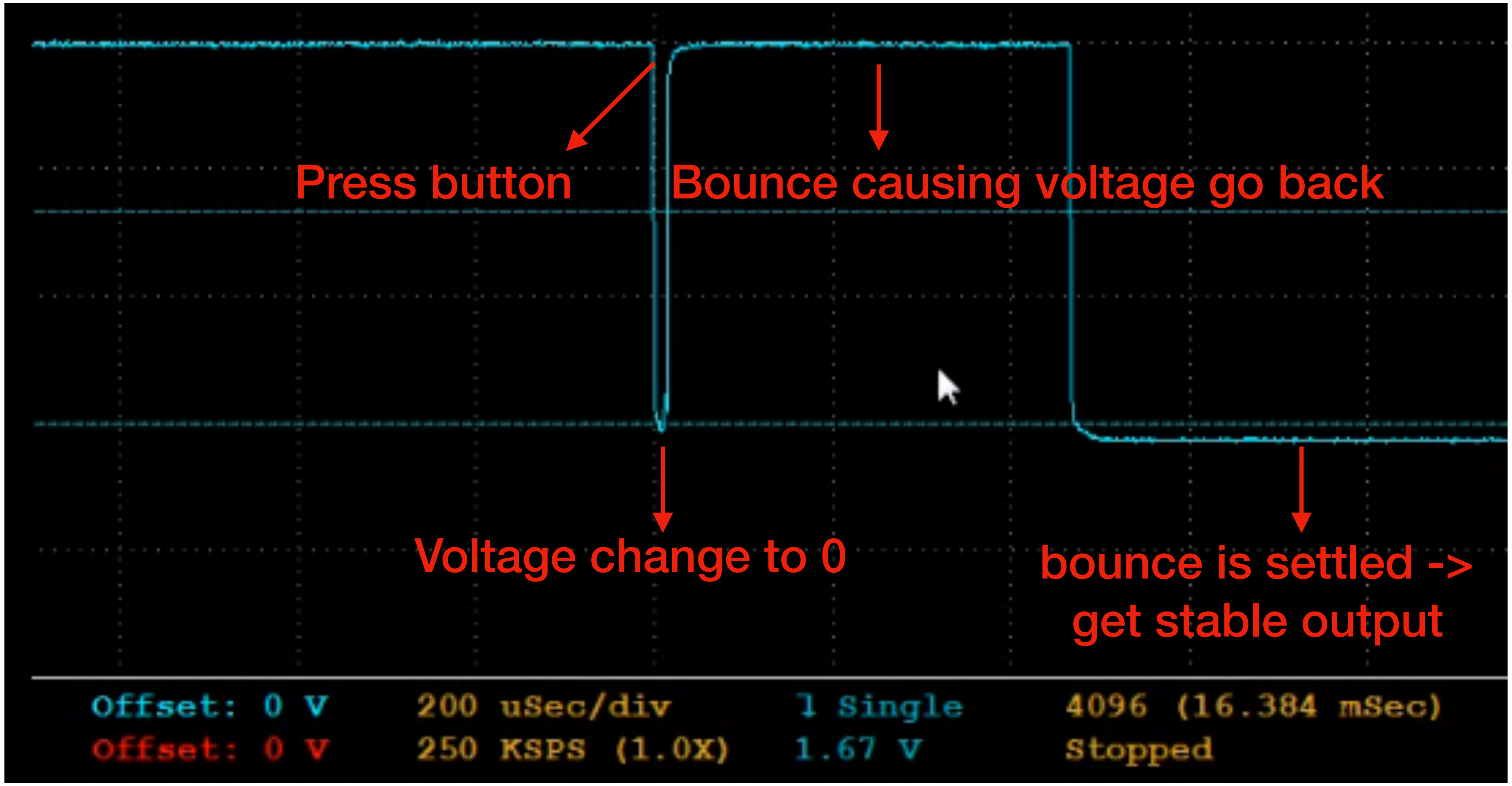
Interact with Push Button (Code)

```
# include "pic24_all.h"

// Configure RB14 as normal digital output
#define CONFIG_LED1 ()      CONFIG_RB14_AS_DIG_OUTPUT
#define SW1       _RB13
#define SW1_PRESSED ()      SW1 == 0    // If SW1 = _RB13 = 0 —> Switch is pressed
#define SW1_RELEASED ()     SW1 == 1    // If SW1 = _RB13 = 1 —> Switch is released

inline void CONFIG_SW1 ()
{
    CONFIG_RB13_AS_DIG_INPUT ();
    ENABLE_RB13_PULLUP;
}
```

```
void main (void)
{
    CONFIG_SW1 (); // Configure _RB13
    // Delay a short time to enable weak pull-up
    DELAY_US (1);
    CONFIG_LED1 ();  // Configure _RB14
    while (1){
        while (SW1_RELEASED ()); // Wait for pressing SW
        DELAY_MS (15);
        while (SW1_PRESSED ());  // Wait for releasing SW
        DELAY_MS (15);
        LED1 = !LED1;
    }
}
```

Delay a short time to avoid switch bounce

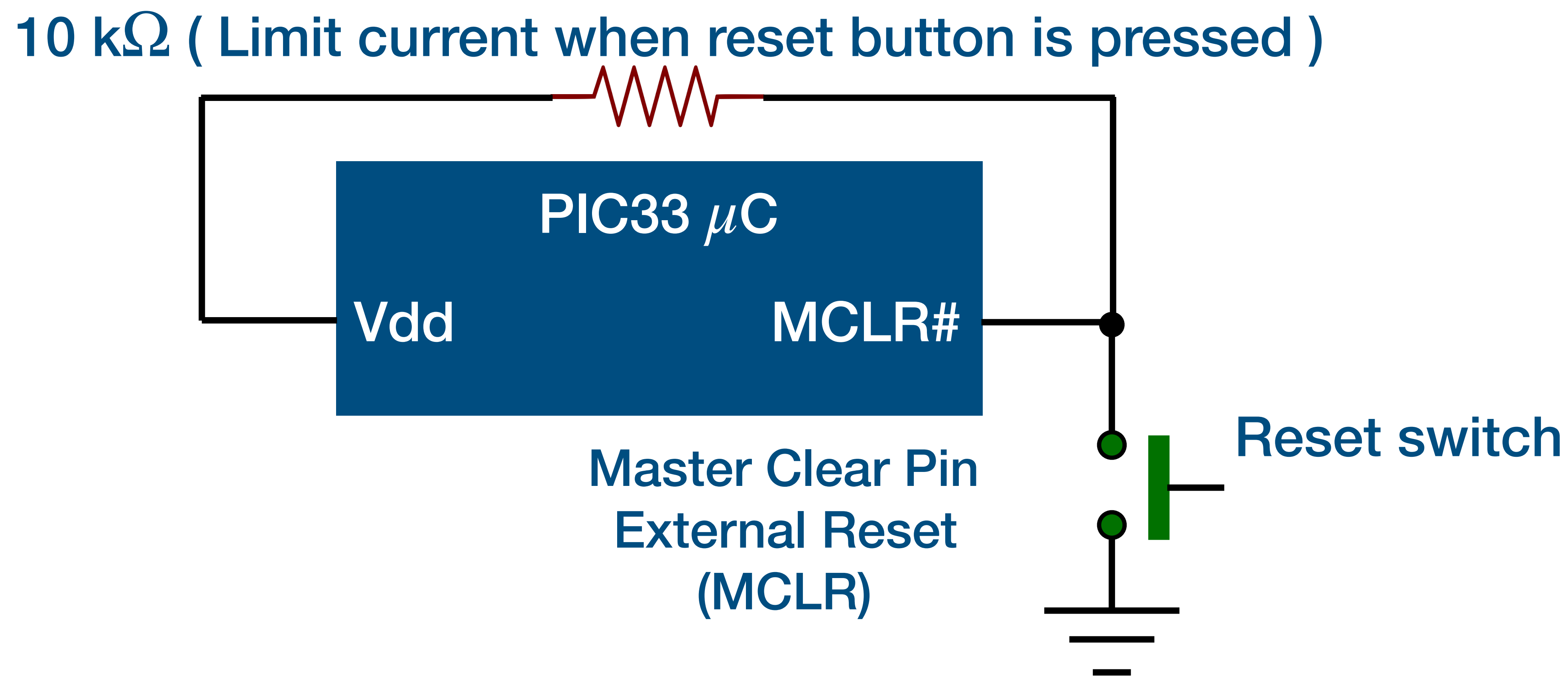
Mechanical Switch Bounce



- Mechanical switches can 'bounce' multiple times when pressed
- Don't sample again until switch bounce has settled

```
while (SW1_RELEASED ()); // Wait for pressing SW
DELAY_MS (15);     <— wait until switch bounce has settled
 while (SW1_PRESSED ());  // Wait for releasing SW
DELAY_MS (15);     <— wait until switch bounce has settled
```

Reset



- PIC detects voltage on MCLR# pin. When voltage becomes low —> PIC is reset

- Press reset switch —> MCLR# is tied to ground —> Voltage on MCLR# pin becomes zero —> PIC is reset —> program counter is reset to 0 —> next instruction fetched will be from location 0.

- All μCs have a reset pin in order to force the μC to a known state.

Reset Types

- There are many reasons can cause a reset. We can check each bit in RCON register to know what type reset occurred
- All Reset flag bit in RCON may be set or cleared by the user software

Flag Bit	Set by:	Cleared by:
TRAPR (RCON<15>)	Trap conflict event	POR, BOR
IOPUWR (RCON<14>)	Illegal opcode or initialized W register access	POR, BOR
CM (RCON<9>)	Configuration Mismatch	POR,BOR
EXTR (RCON<7>)	MCLR# Reset	POR
SWR (RCON<6>)	`reset` instruction	POR, BOR
WDTO (RCON<4>)	WDT time-out	`pwrsav` instruction, `clrwdt` instruction, POR,BOR
SLEEP (RCON<3>)	`pwrsav #0` instruction	POR,BOR
IDLE (RCON<2>)	`pwrsav #1` instruction	POR,BOR
BOR (RCON<1>)	BOR	n/a
POR (RCON<0>)	POR	n/a

Power on reset

Brown-out reset (When Vdd drops below a threshold)

Subroutine to Check Reset Reason

```
void printResetCause(void) {
  if (_SLEEP) {
    outString("\nDevice has been in sleep mode\n"); _SLEEP = 0;
  }
  if (_IDLE) {
    outString("\nDevice has been in idle mode\n");  _IDLE = 0;
  }
  outString("\nReset cause: ");
  if (_POR) {
    outString("Power-on.\n");  _POR = 0; _BOR = 0; //clear both
  } else { //non-POR causes
   if (_SWR) {
    outString("Software Reset.\n");              _SWR = 0;     }
   if (_WDTO) {
    outString("Watchdog Timeout. \n");           _WDTO = 0;    }
   if (_EXTR) {
    outString("MCLR assertion.\n");              _EXTR = 0;    }
   if (_BOR) {
    outString("Brown-out.\n");                   _BOR = 0;     }
   if (_TRAPR) {
    outString("Trap Conflict.\n");               _TRAPR = 0;   }
   if (_IOPUWR) {
    outString("Illegal Condition.\n");           _IOPUWR = 0; }
   if (_CM) {
    outString("Configuration Mismatch.\n");      _CM = 0;      }
  }//end non-POR causes
  checkDeviceAndRevision();
  checkOscOption();
}
```

A status bit is cleared if it has been set.

Print status on processor ID and revision, and clock source.

Watchdog Timer (WDT)

- A watchdog timer is a timer that is used to detect and recover from μC malfunctions.

- During normal operation, the controller regularly resets the watchdog timer to prevent it from timing out.

- If there is a hardware fault or program error, the μC fails to reset the watchdog, the timer will timeout and then generate a timeout interrupt to reset the μC.

WDT Specifics (1)

- WDT uses independent free-running RC oscillator as a clock. The frequency is 32.768 kHz, runs even when normal clock is stopped.
- WDT timeout occurs when counter overflows from max value back to 0. The timeout period can be calculated as follows:

$$\text{WDT timeout} = \text{clock period} \times \text{WDT prescaler} \times \text{WDT postscaler}$$

$$= \frac{1}{32.768\ \text{kHz}} \times \text{WDT prescaler} \times \text{WDT postscaler}$$

Prescaler is a frequency divider that make additional division of the clock source frequency before it gets into the WDT timer

Postscaler is a counter. It determines how frequent that WDT generates the timeout interrupt

WDT Specifics (2)

Clock source frequency = f

↓

Prescaler = 2

↓

frequency1 = $\frac{f}{2}$

↓

Postscaler = 6

↓

interrupt frequency = $\frac{\text{frequency1}}{6}$

$= \frac{f}{2 \times 6} = \frac{f}{12}$

- In PIC, WDT prescaler can be 32 or 128, only two options
  A. If bit WDTPRE = 0 —> WDT prescaler = 32
  B. If bit WDTPRE = 1 —> WDT prescaler = 128

- WDT postscaler can be 1, 2, 4, 8, …, 2^{15}, by setting WDTPOST, which has 15 bits

- When WDTPRE = 0 and WDTPOST = 1, WDT get the minimum timeout time:

$$\text{WDT timeout} = \frac{1}{32.768\ \text{kHz}} \times 32 \times 1 \approx 1\ \text{ms}$$

- When WDTPRE = 1 and WDTPOST = 2^{15}, WDT get the maximum timeout time:

$$\text{WDT timeout} = \frac{1}{32.768\ \text{kHz}} \times 128 \times 2^{15} \approx 131\ \text{s}$$

WDT Uses (1)

Important !

- A WDT timeout during normal operation will reset the PIC
- A WDT timeout during sleep or idle mode will wake up the PIC and resumes operations

- _SWDTEN bit can be used to enable/disable WDT
  A. If _SWDTEN = 0 —> WDT is disabled
  B. If _SWDTEN = 1 —> WDT is enabled and start to count

- The clrwdt instruction clears the WDT timer, prevents timeout

WDT Uses (2)

- Error recovery: If the μC is designed to wait for the response of a peripheral (e.g., keyboard). WDT can break the controller from an infinite wait loop by reseting the μC if a response does not come back in a particular time period.

- Wakeup a sleeping controller: If the μC has been put in a sleep mode, then WDT can wake the controller after the WDT timeout period has elapsed.

Power Saving Modes

- PIC provides several different power saving modes.

- Sleep mode:
  A. CPU and all peripherals stop working.
  B. Can be awoke by the WDT timeout and external interrupt.
  C. Use the pwrsav #0 instruction to enter the sleep mode.

- Idle mode:
  A. CPU stop working.
  B. Peripherals can still work (e.g., receive data through UART).
  C. Use the pwrsav #1 instruction to enter the idle mode.

- Doze mode:
  A. CPU and peripherals still work.
  B. Main clock to CPU is divided by doze prescaler (2, 4, ..., 128)
  C. Peripheral clocks unaffected. CPU runs slower, but peripherals run at full speed

Current Consumption

Mode	PIC24@40MHz (mA)	PIC24@16MHz (mA)
Normal	42.3	5.6
Sleep	0.03	0.004
Idle	17.6	2.0
Doze/2	32.2	4.0
Doze/128	17.9	2.0