SYSC 3310: Introduction to Real Time Systems

Lecture 3: Basic I/O

Dr. Paulo Garcia
Assistant Professor
Department of Systems and Computer Engineering

Revision:

- We learned about microcontroller architecture
  - How peripheral devices are memory mapped
- We learned about C programming
  - Pitfalls, details and intricacies, good practices
- We learned about using pointers, macros and structs/unions to implement device access

Today:

- Basic I/O: applying what we learned
  - Interacting with peripheral devices
  - Good coding practices
  - Important details

Revisiting "memory mapped"

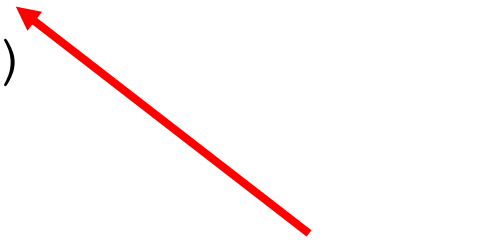
```
i = i+1;
```

(C)

Revisiting "memory mapped"

```
i = i+1;
```

(C)



"i" is at address 27

Revisiting "memory mapped"

```
i = i+1;
```

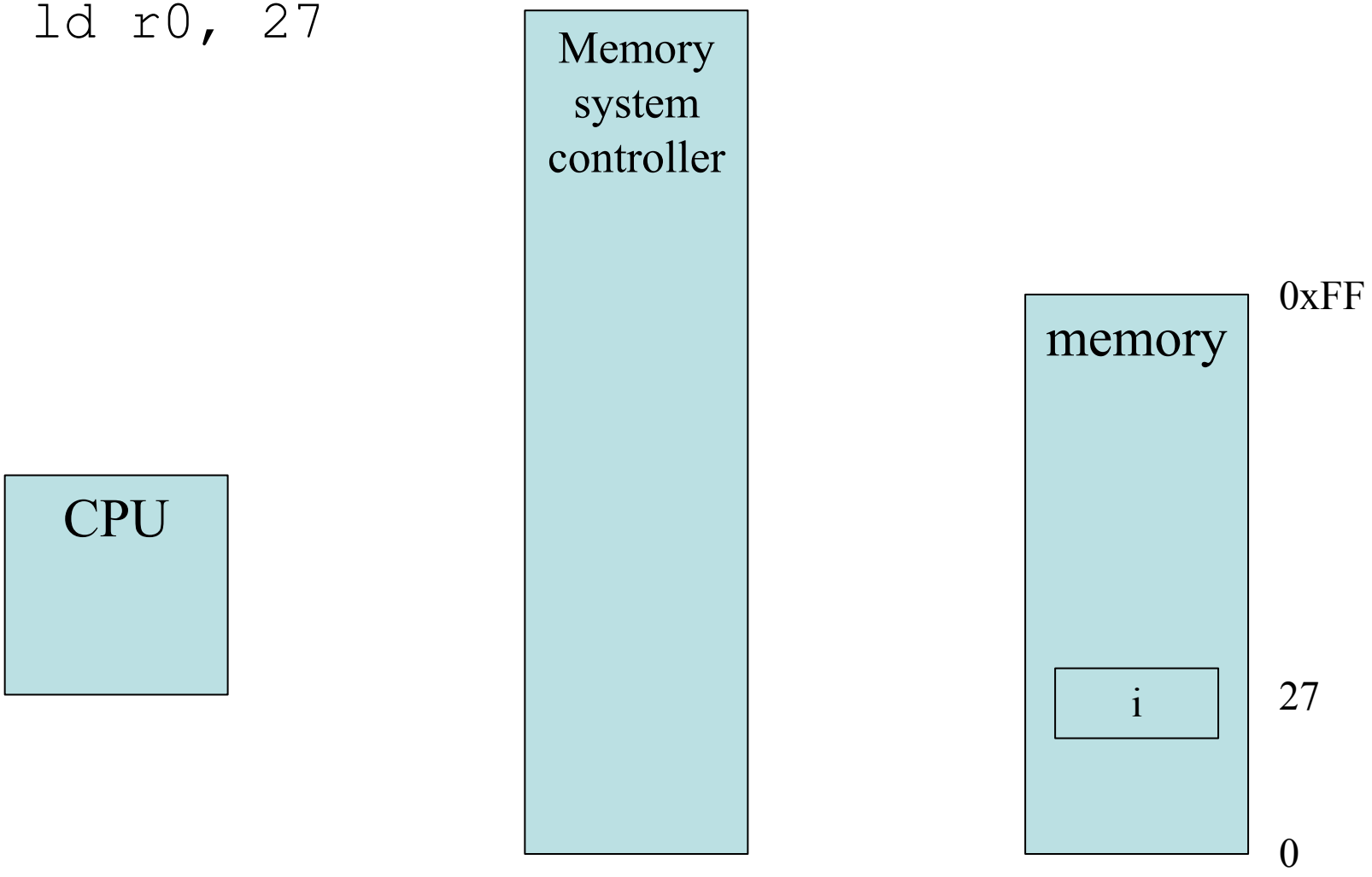
(C)

```
ld r0, 27
add r0, r0, 1
st r0, 27
```

(assembly)

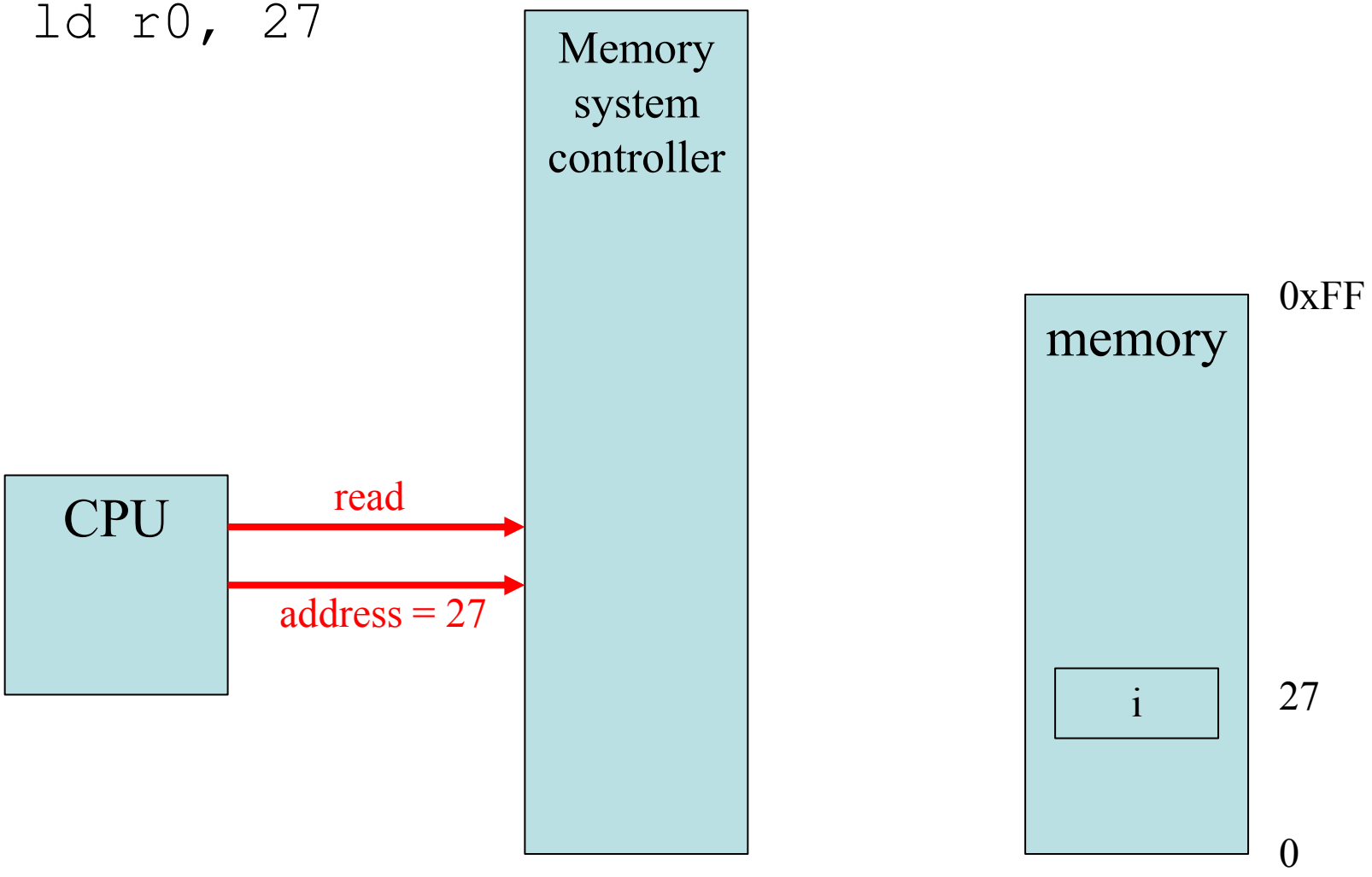
Revisiting "memory mapped"

```
ld r0, 27
```



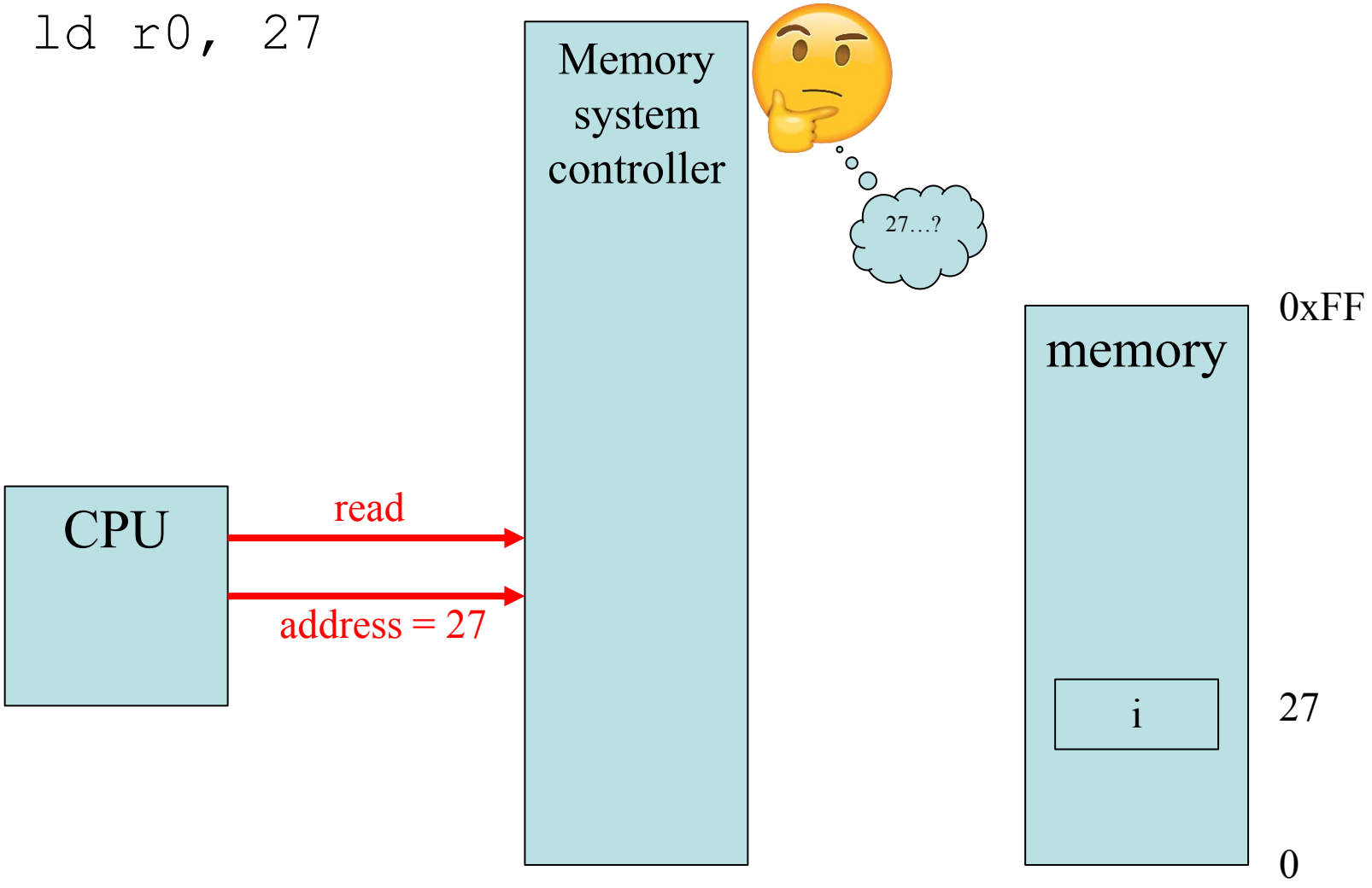
Revisiting "memory mapped"

```
ld r0, 27
```



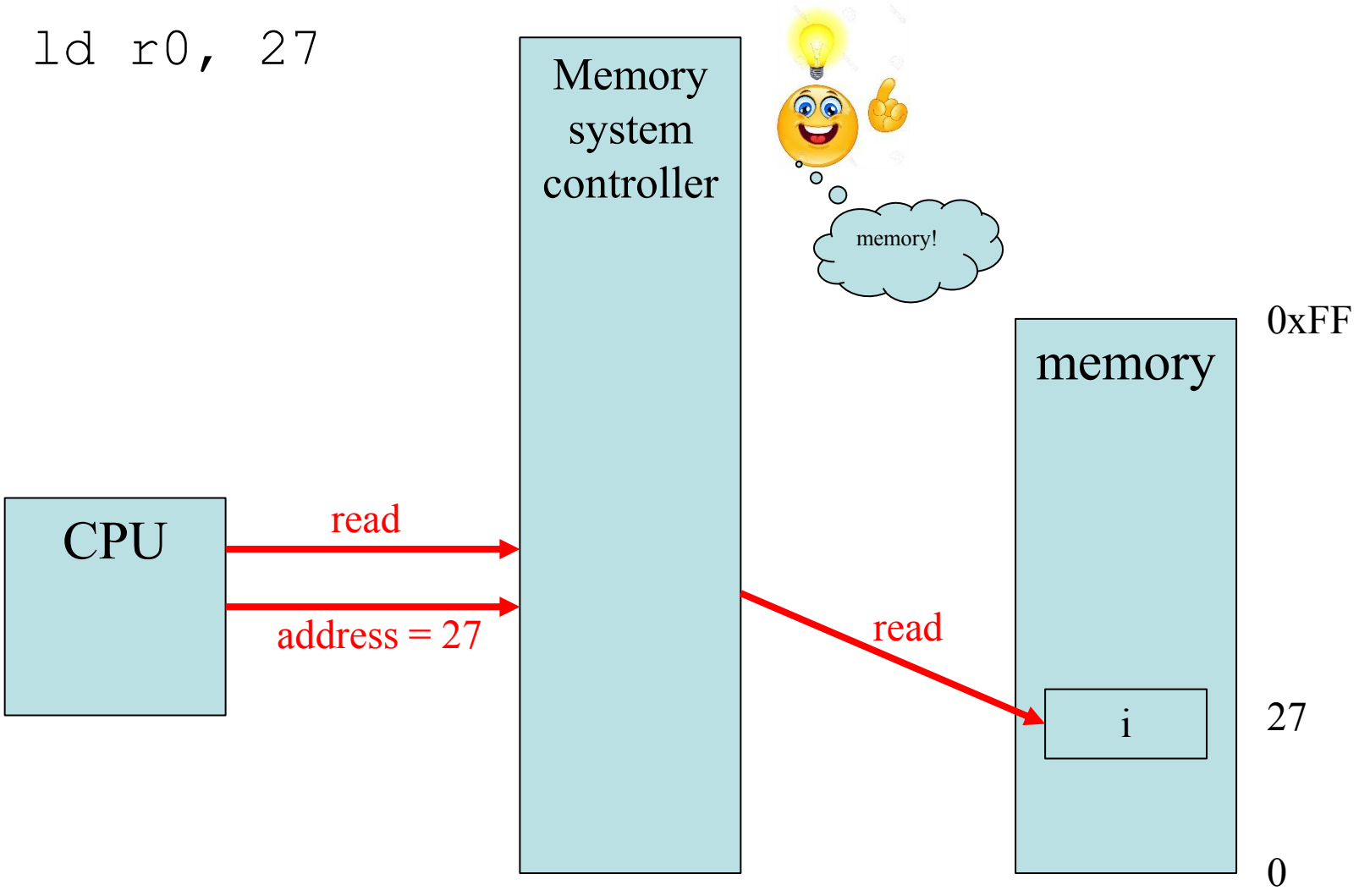
Revisiting "memory mapped"

```
ld r0, 27
```



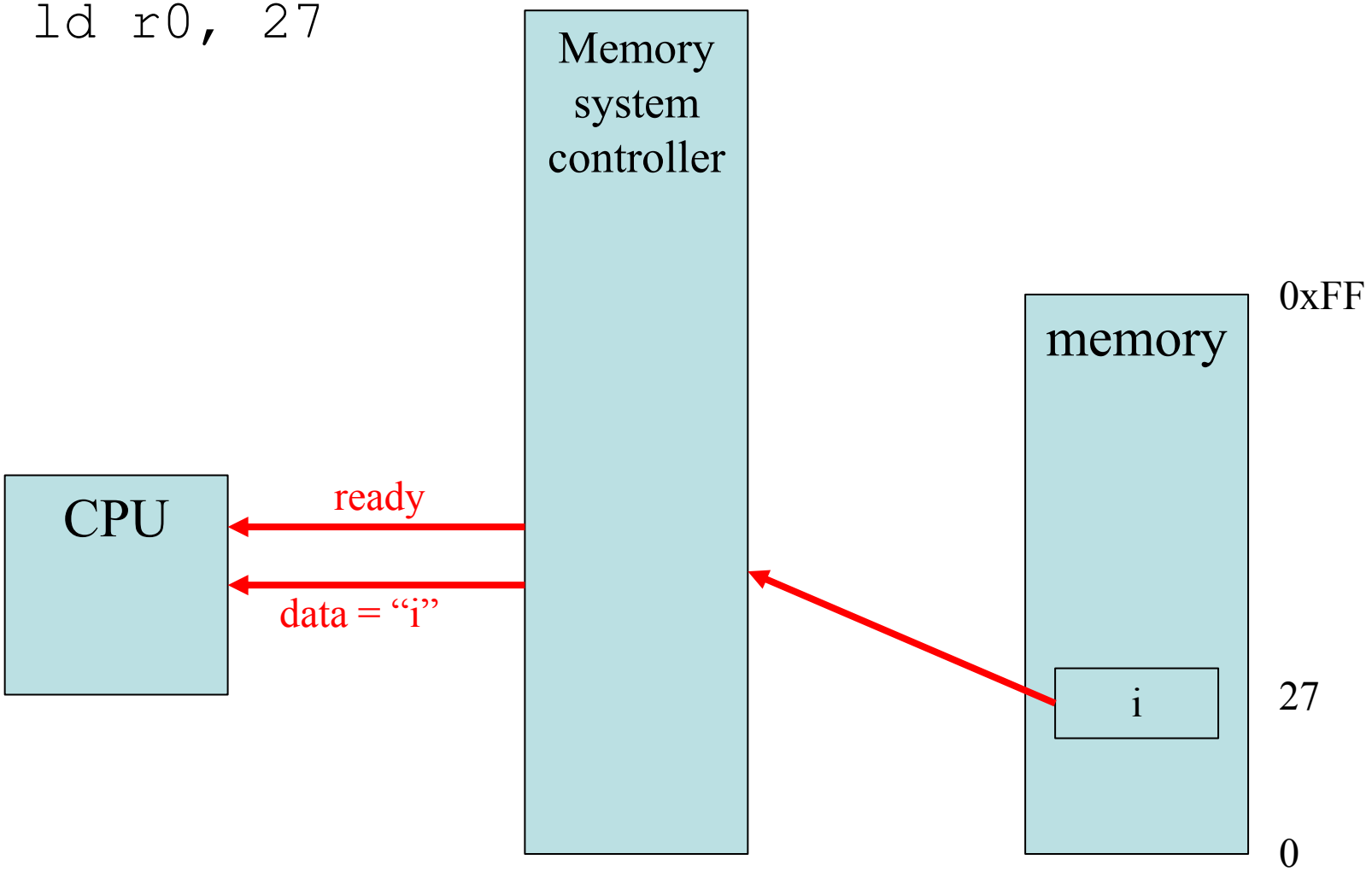
Revisiting "memory mapped"

```
ld r0, 27
```



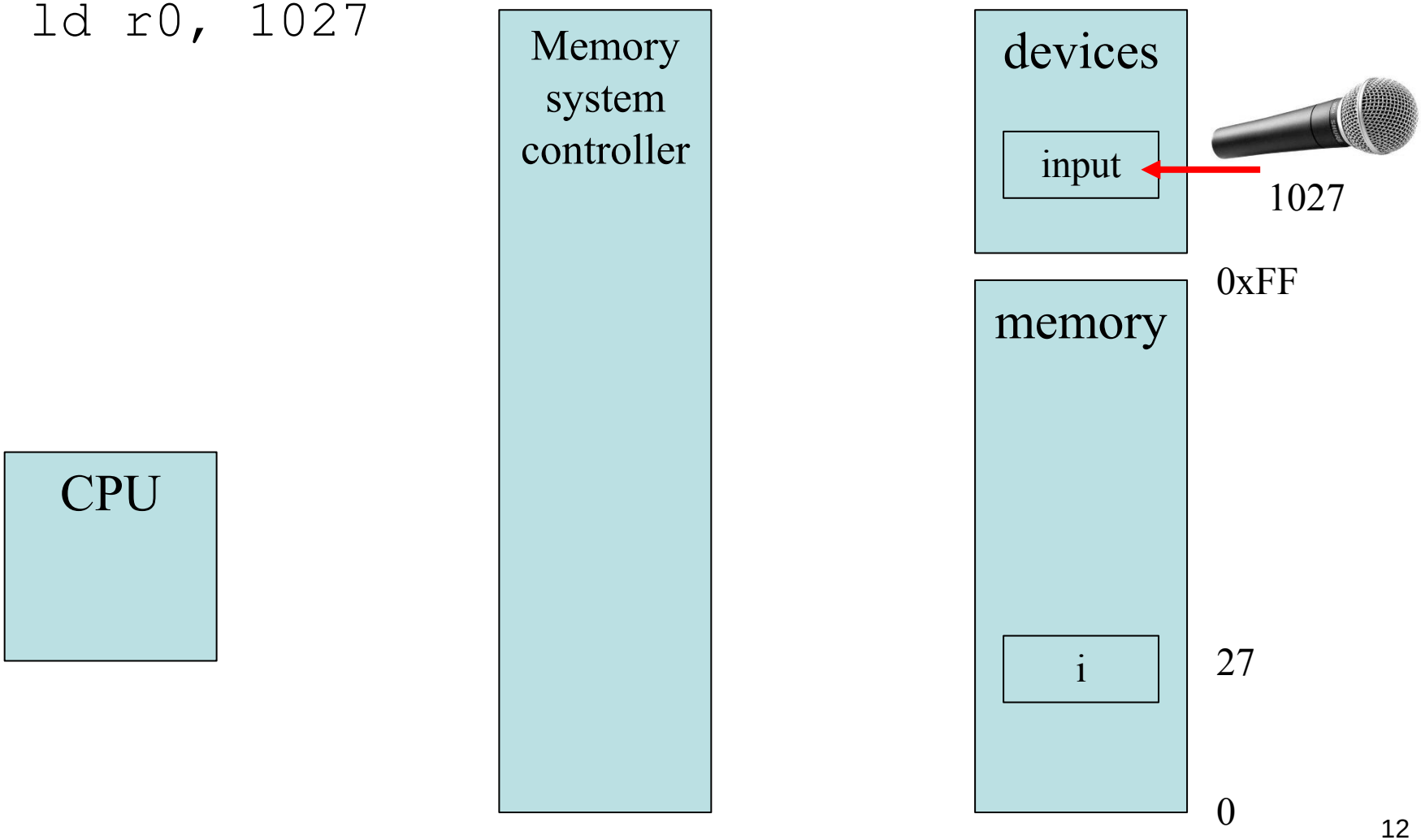
Revisiting "memory mapped"

```
ld r0, 27
```



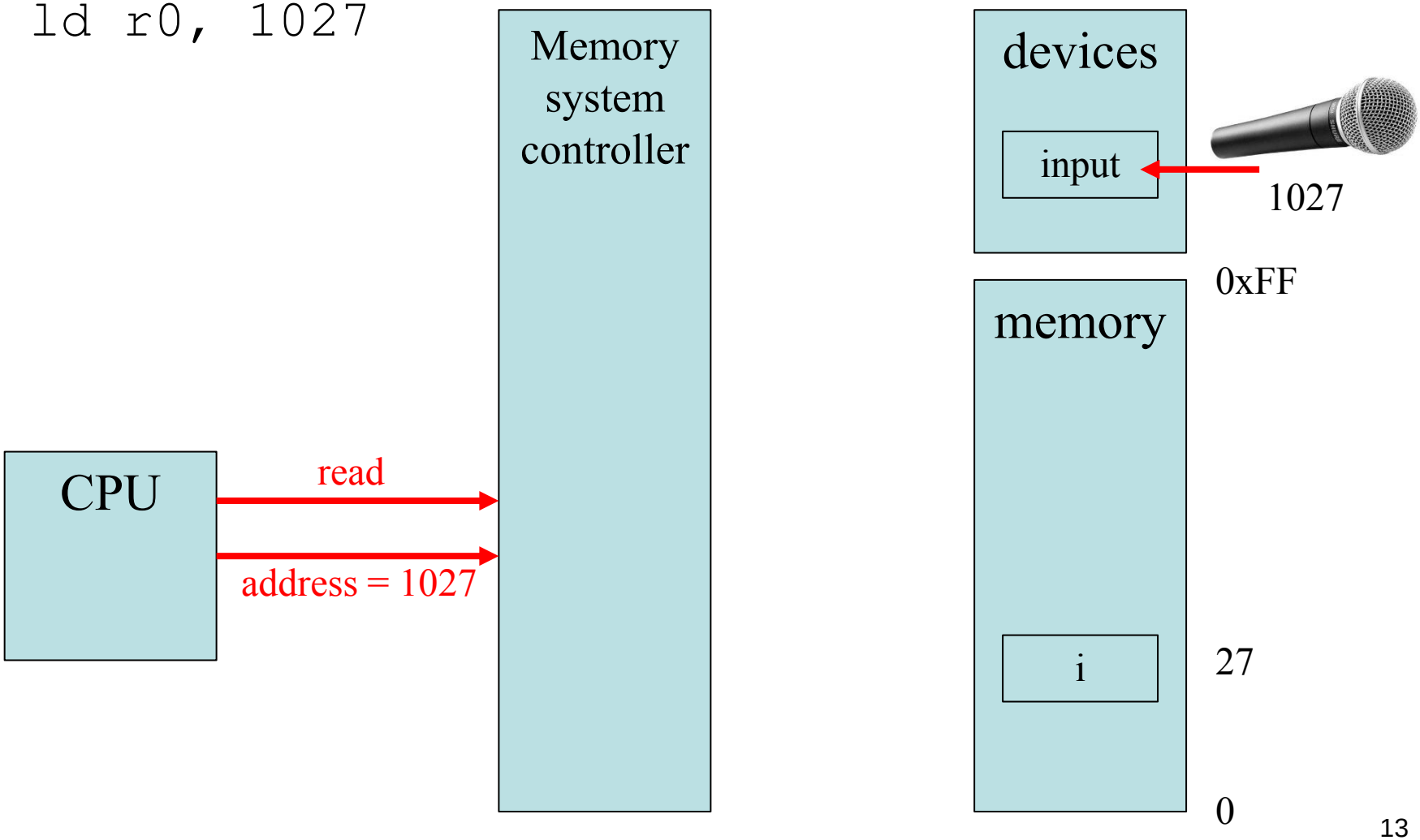
Revisiting "memory mapped"

```
ld r0, 1027
```



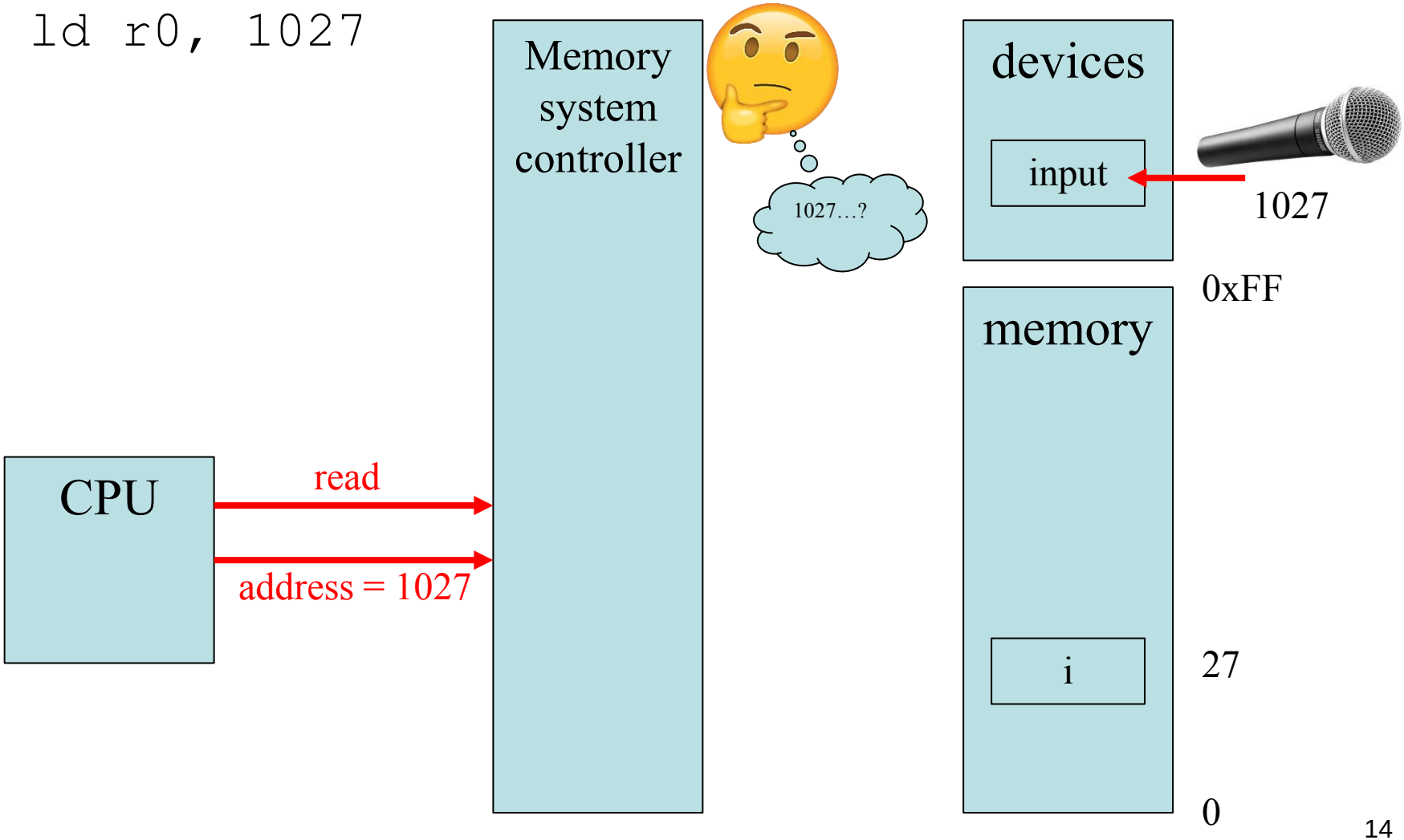
Revisiting "memory mapped"

```
ld r0, 1027
```



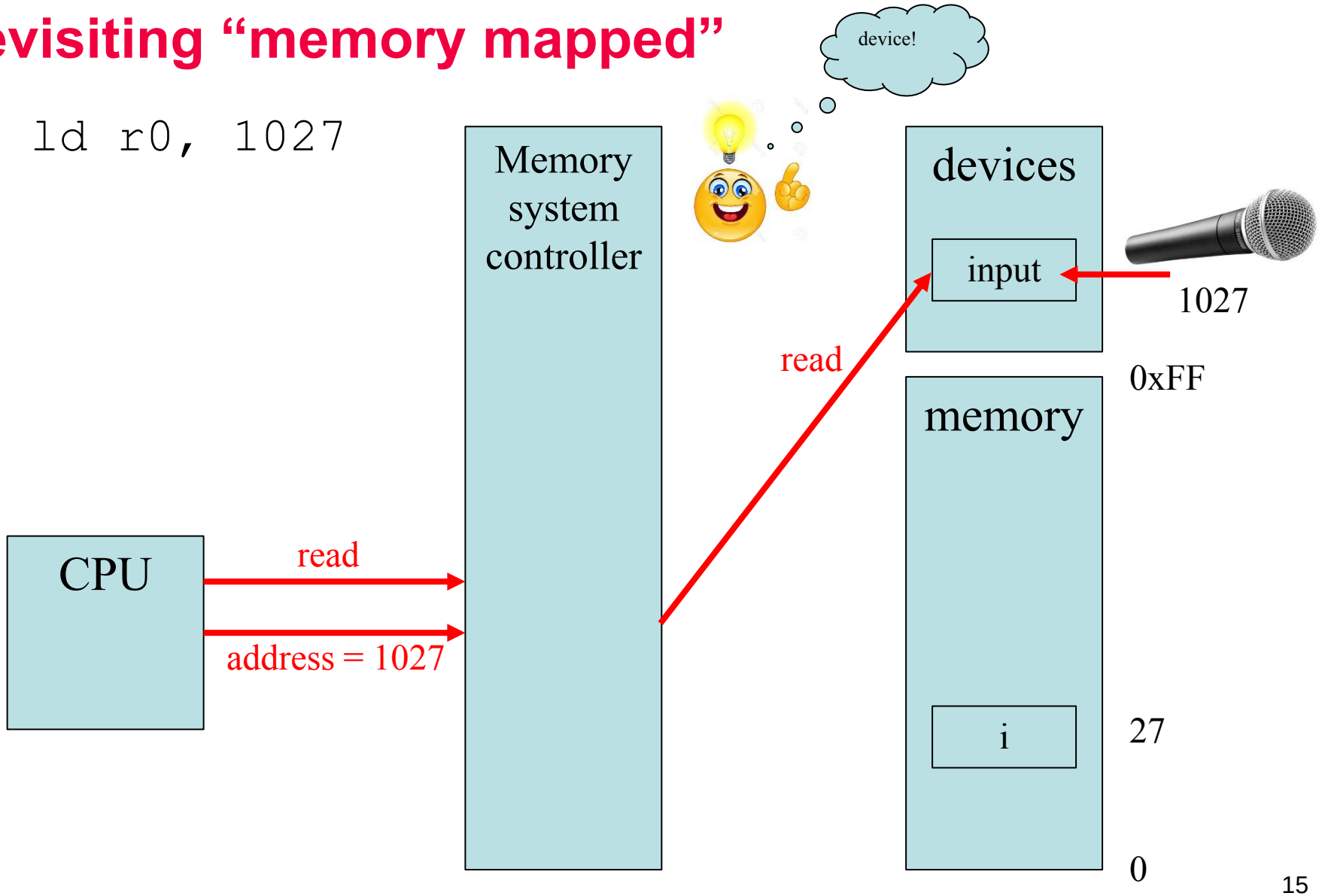
Revisiting "memory mapped"

```
ld r0, 1027
```



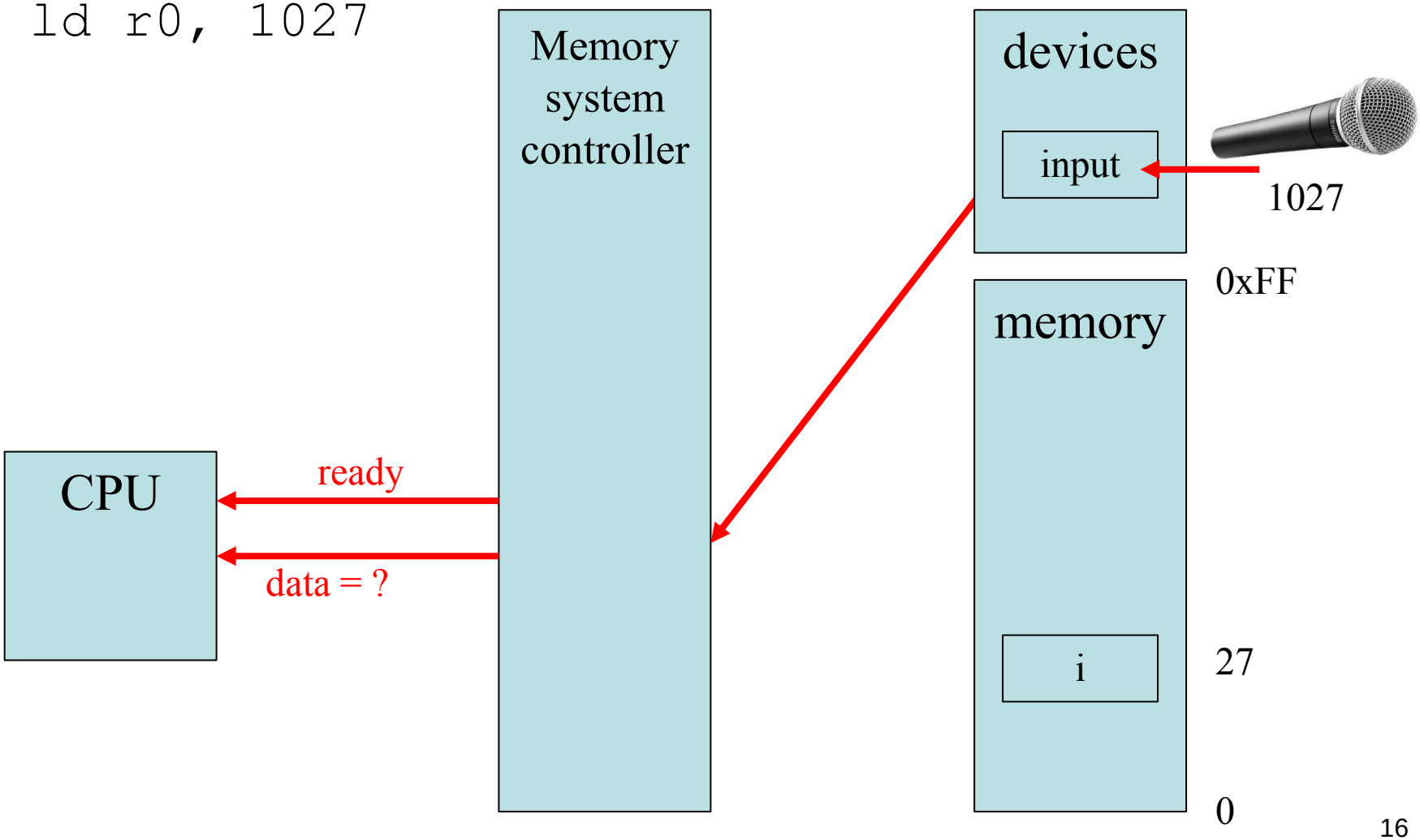
Revisiting "memory mapped"

```
ld r0, 1027
```



Revisiting "memory mapped"

```
ld r0, 1027
```



Your main function

- How to actually write code for embedded systems

Your main function

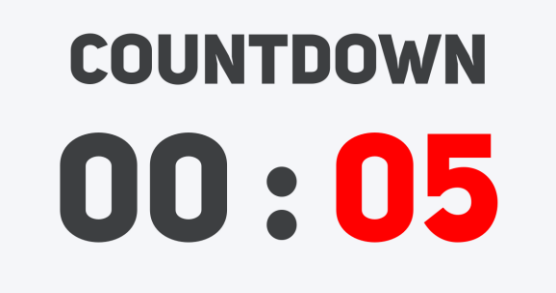
- Should always start with this line

```
// Stop watchdog timer
WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
```

  - This stops the watchdog timer so your program can run properly

What's a watchdog timer?

- Simply a timer that starts operating when the device starts
- Operation: counting down



  - When it reaches 0, the device is reset

What's a watchdog timer?

- Why?
  - To prevent bugs (e.g., software infinite loops)
  - Software must periodically update the timer
    - I.e., load its initial value again
    - Basically saying "I'm moving along, don't reset me"

What's a watchdog timer?

- Why?
  - To prevent bugs (e.g., software infinite loops)
  - Software must periodically update the timer
    - I.e., load its initial value again
    - Basically saying "I'm moving along, don't reset me"
  - We'll just stop it
    - And assume your code doesn't have bugs

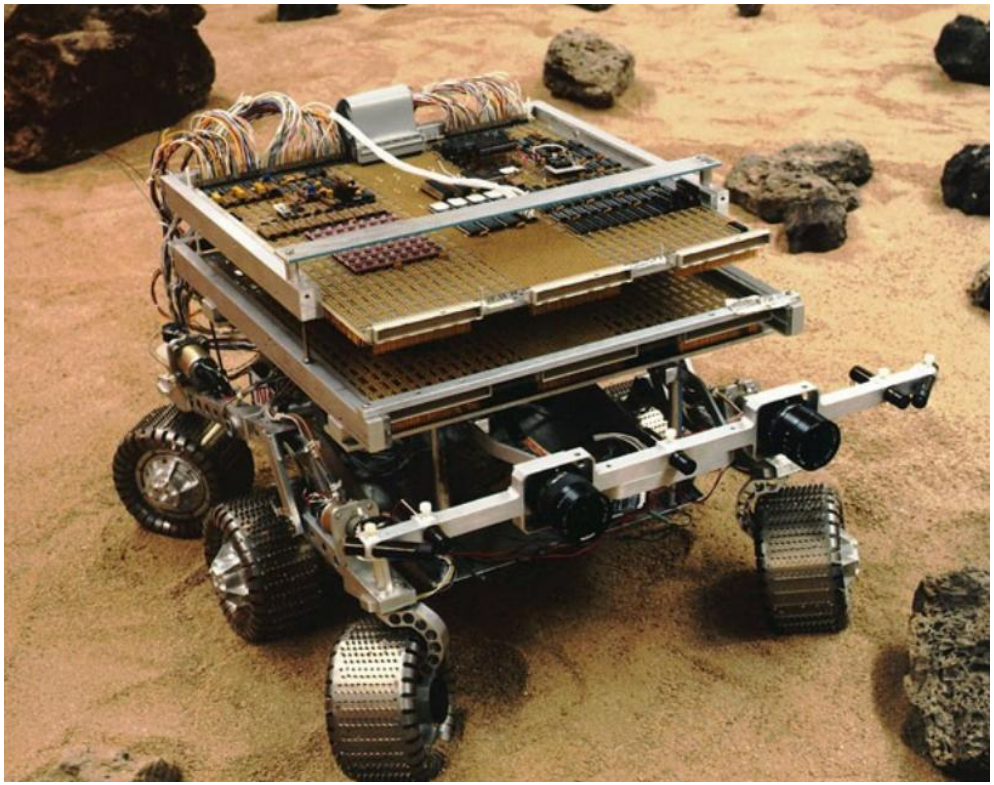
Case Study

Case study: Mars Pathfinder

- First mission to land a rover on Mars
  - Dec. 4, 1996
    - Pathfinder launched.
  - July 4, 1997
    - Pathfinder lands on Mars
  - Sept. 27, 1997
    - last successful data transmission from Mars Pathfinder

Case study: Mars Pathfinder

- Within a few days of landing, when Pathfinder started gathering meteorological data, spacecraft began experiencing total system resets



Case study: Mars Pathfinder

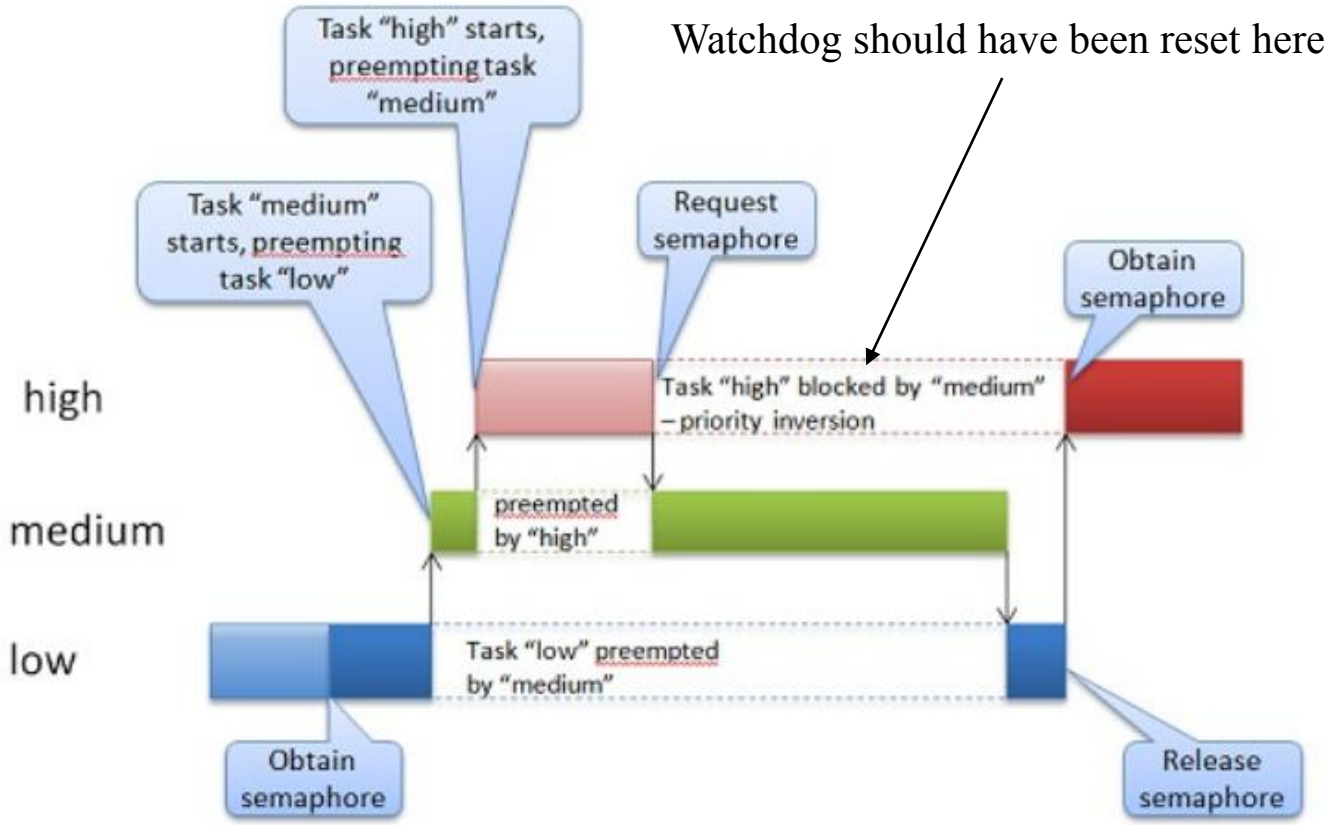
- NASA and JPL engineers had exact replica of the spacecraft in their lab
- After 18 hours of execution, early next morning when all but one engineer had gone home, the symptom was reproduced.
  - Watchdog timer wasn't being updated properly

Case study: Mars Pathfinder

- Spacecraft ran IBM RS6000 processor and WindRiver's VxWorks RTOS
  - 20 MIPS performance
  - 128 MB of DRAM for storage of flight software and engineering and science data, including images and rover information.
  - 6 MB ROM stored flight software and time-critical data.
- Hard real-time OS with concurrent threads execution

Case study: Mars Pathfinder

- Priority inversion bug prevented Watchdog update



Case study: Mars Pathfinder

- In the book "The Martian", the title character uses the computer from the Pathfinder
- Has to patch it (modify binary code) to perform communication
  - (You will learn how to do this)
  - The science and engineering are actually really accurate: well worth a read

Case study: Mars Pathfinder

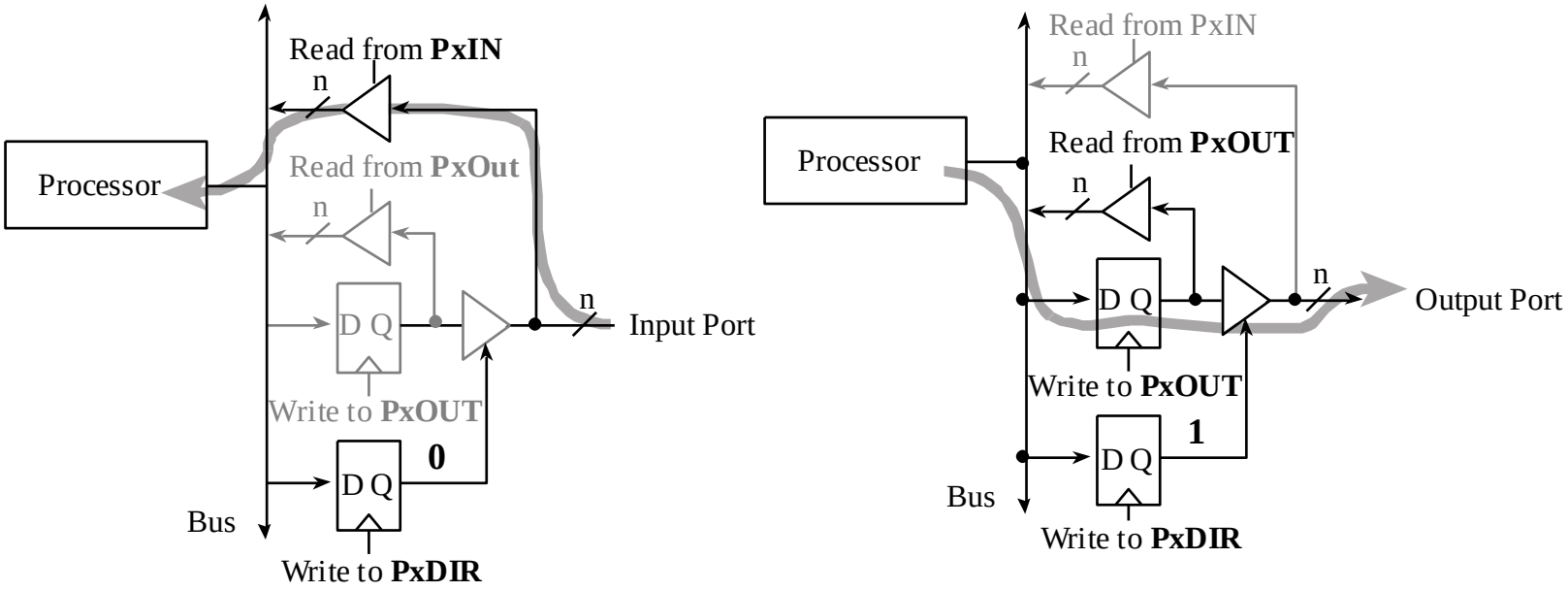
- To remember:
  - Software must pay close attention to the hardware
  - In real time systems, *when* something happens matters!
    - Making sure everything happens on time is not easy.
    - To be safe: stop your watchdog timer!

I/O

- What is GPIO?
  - General Purpose Input and Output
  - Digital pins
- Why general purpose?
  - There is application specific I/O
    - E.g., SPI, I2C, USART…..
  - Where hardware implements specific protocols
- In GPIO, there is no protocol
  - Software can do anything (bit-banging)

I/O

- Understand the hardware (for each pin)



- Flip-flops for holding output data and direction
- Tri-state buffers to control input or output (direction)

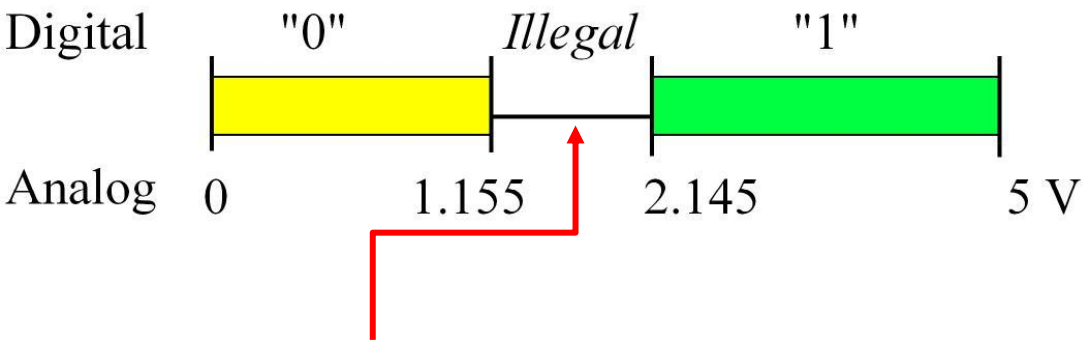
I/O

- Reminder:

There's no such thing as "digital"!

I/O

- Everything is analog
  - "Digital" is an *abstraction*
  - A convenient lie we tell ourselves and everyone else….
    - (to make life easier)

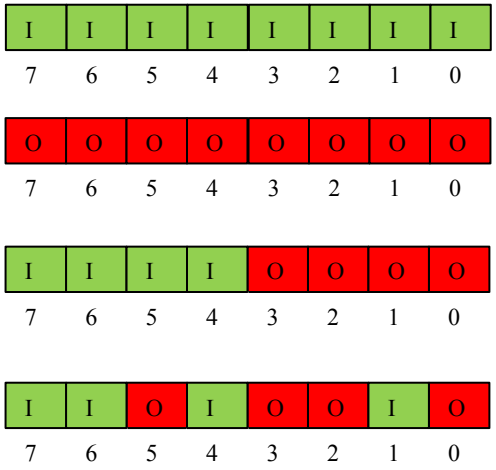


I/O

- Circuits *behave* as digital
  - As long as certain rules are obeyed
    - Frequency
    - Voltage
    - Current
    - …

I/O

- Each I/O *pin* is 1 bit "digital"
- Pins are grouped into *ports* (1 Port = 8 bits)
  - For every port, each pin can be configured individually:

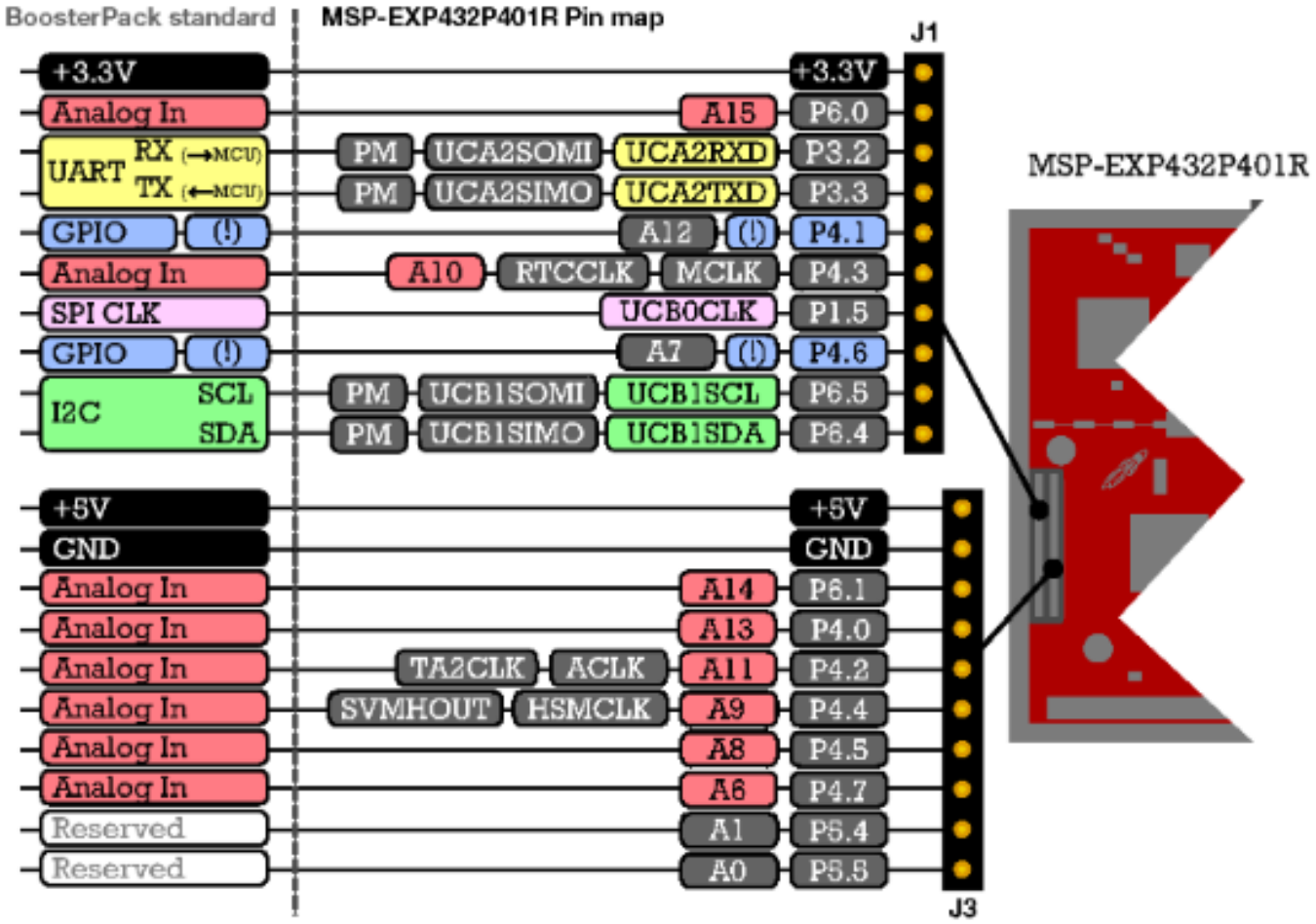


I/O

- Names and naming conventions:
  - P1 – Port 1 (entire byte)
  - P2 – Port 2 (entire byte)
  - P1.0 – Port 1, pin 0 (single bit)
  - P1.2 – Port 1, pin 1 (single bit)
    - Remember those nice bit-mapped structs for peripheral device access?

I/O

- Careful: some pins are multi-purpose



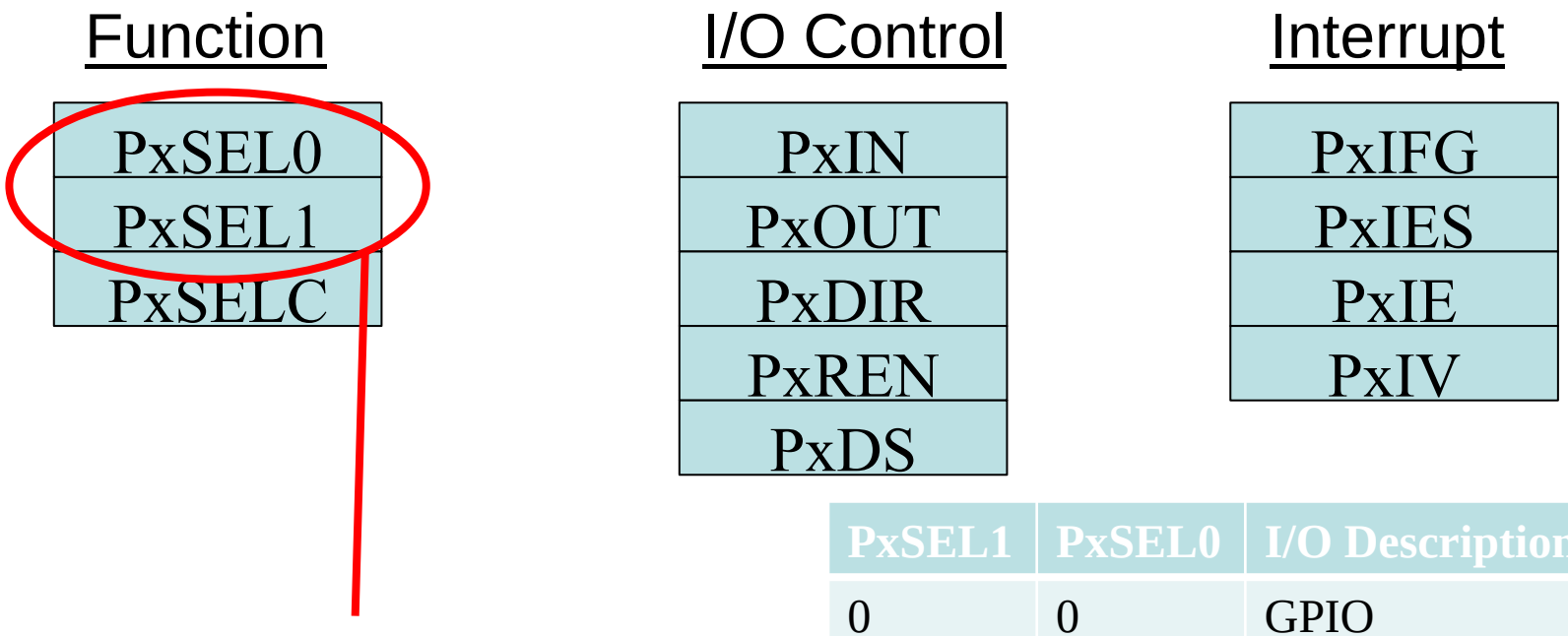
I/O

- Port control Registers: Each port has 12

Function	I/O Control	Interrupt
PxSEL0	PxIN	PxIFG
PxSEL1	PxOUT	PxIES
PxSELC	PxDIR	PxIE
	PxREN	PxIV
	PxDS	

I/O

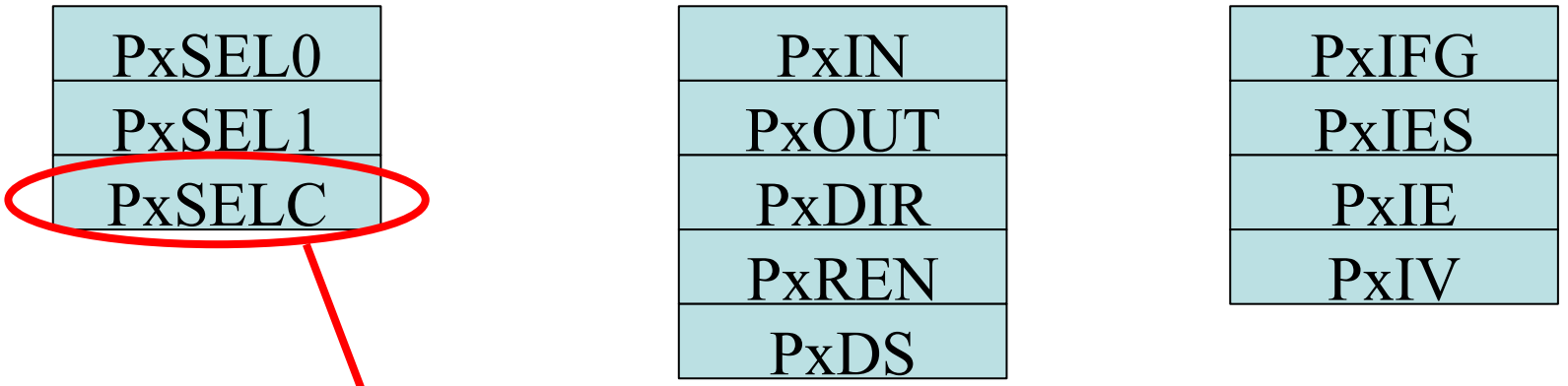
- Port control Registers: Each port has 12



PxSEL1	PxSEL0	I/O Description
0	0	GPIO
0	1	Alternate function 1
1	0	Alternate function 2
1	1	Alternate function 3

I/O

- Port control Registers: Each port has 12



Writing to SEC toggles (complements) both SEL0 and SEL1 at same time
WHY?

PxSEL1	PxSEL0	I/O Description
0	0	GPIO
0	1	Alternate function 1
1	0	Alternate function 2
1	1	Alternate function 3

I/O

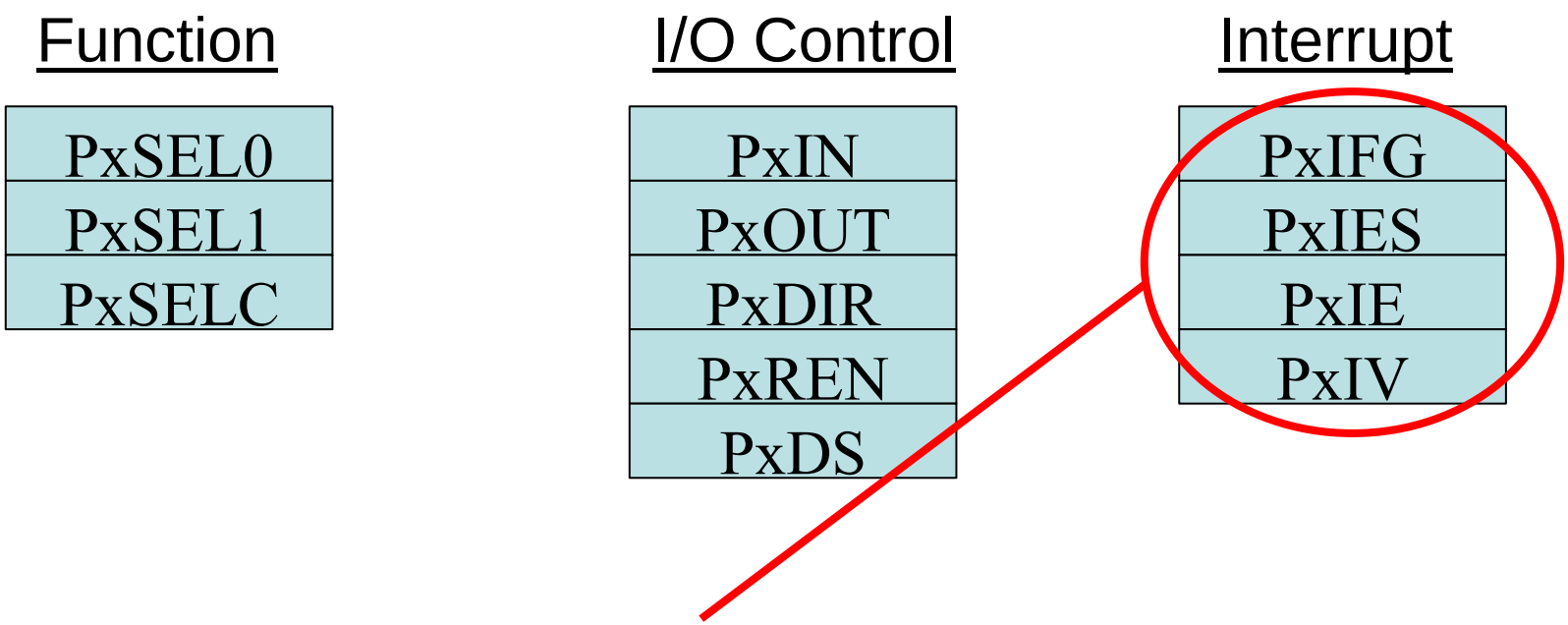
- For now, we just care about GPIO, so both SEL0 and SEL1 should be 0
  - By default (reset) they are set to 0
  - BUT……
    - Your code must guarantee this!
    - At the beginning of "main", you should do all initializations
    - Set them to 0

DO NOT TRUST THE HARDWARE

https://theconversation.com/dont-trust-your-hardware-why-security-vulnerabilities-affect-us-all-105773

I/O

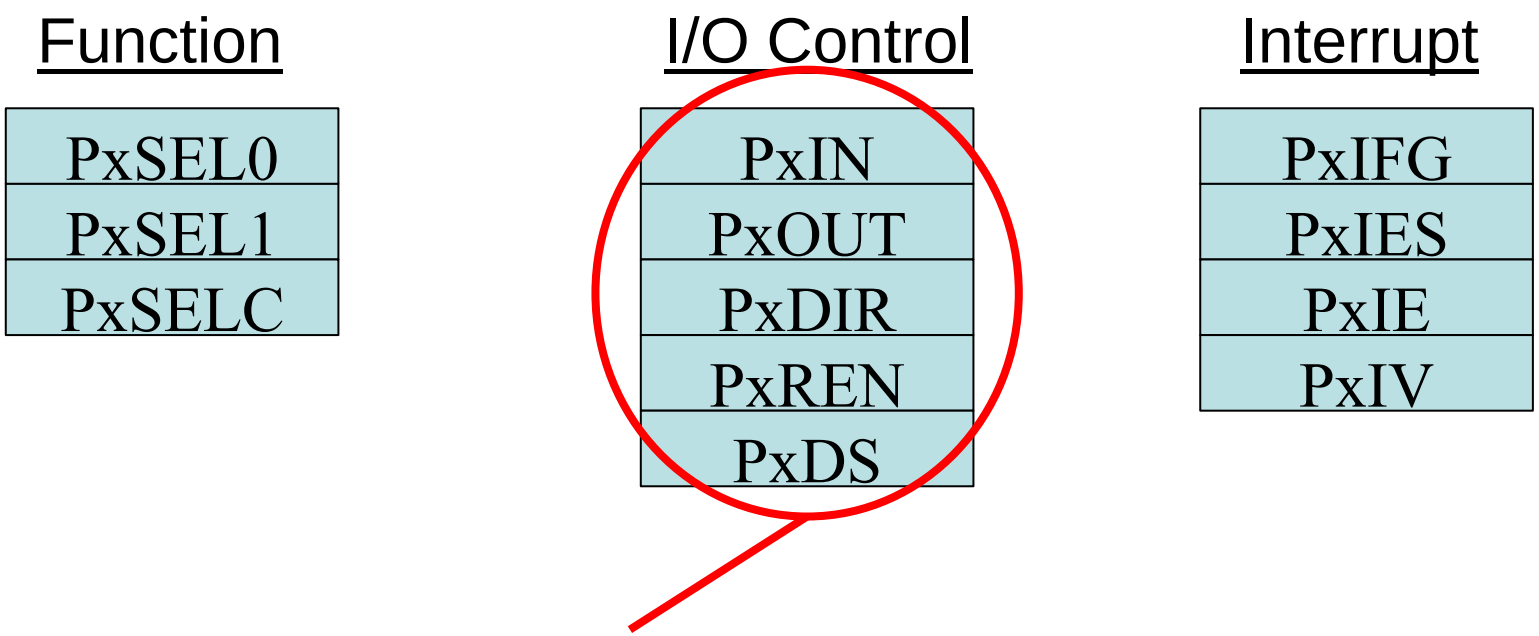
- Port control Registers: Each port has 12



We won't worry about these for now (we'll talk about interrupts next week)

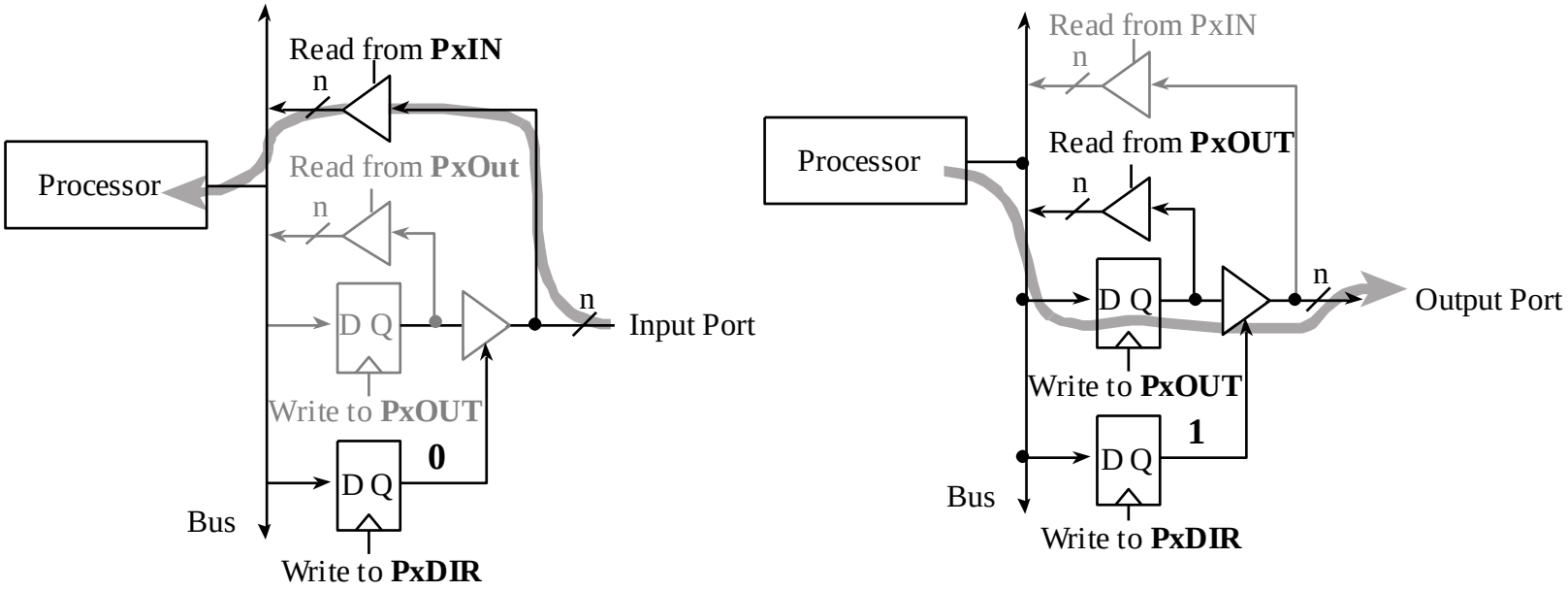
I/O

- Port control Registers: Each port has 12



I/O

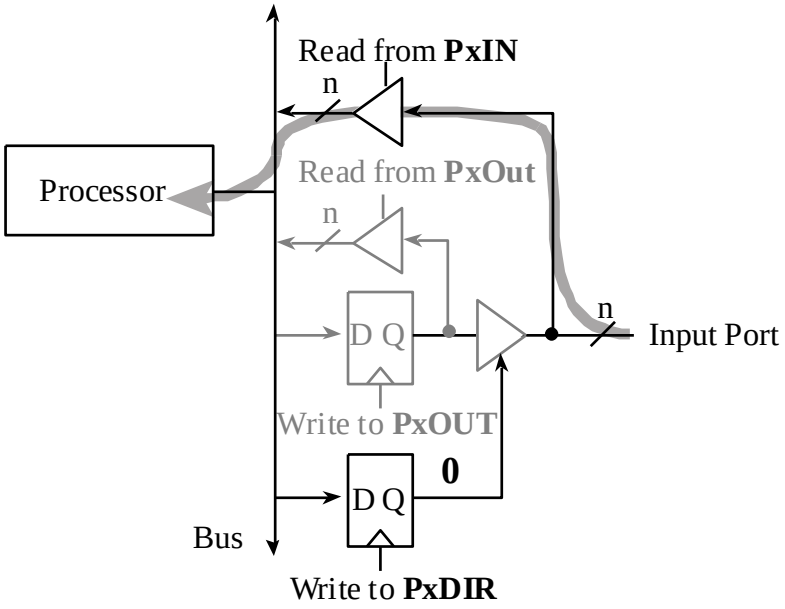
- Understand the hardware (for each pin)

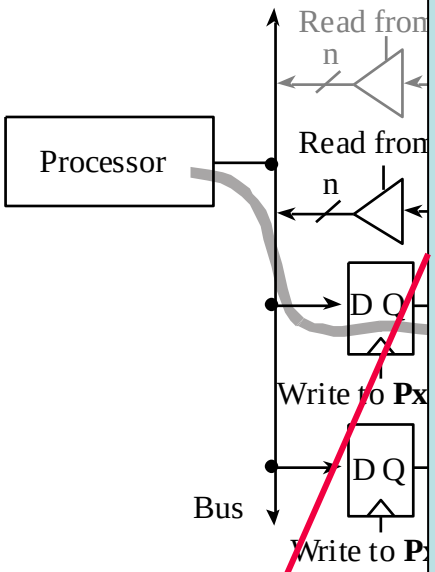


- PxIN: Read (if configured as input)
- PxOUT: Write (if configured as output)
- PxDIR: configure direction (Input 0, Output 1)

I/O

- Understand the hardware (for each pin)





Initialization

Configure pins accordingly.

Why is input the default one?

- PxIN: Read (if configured as input)
- PxOUT: Write (if configured as output)
- PxDIR: configure direction (Input 0, Output 1)

I/O

- PxREN and PxDS
  - PxREN is Resistor Enable (for inputs)
  - PxDS is Drive Strength (for outputs)

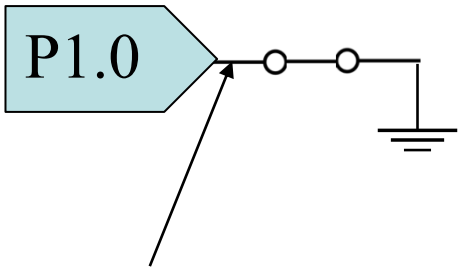
I/O

- PxDS: Drive Strength
  - For (some) ports, selects whether to use regular strength or high strength
  - Basically, sets the upper limit on the current the port can supply
  - We'll just use regular strength
    - Good design: Pins never supply current, only voltage
    - I.e., outside circuitry is controlled by a MOSFET, pin drives MOSFET gate

I/O

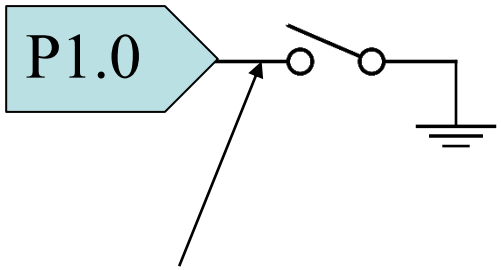
- PxREN: Resistor Enable
  - Example: Bad design (remember, no current flows into the pin)

(configured as input)



voltage: 0V (logical 0)

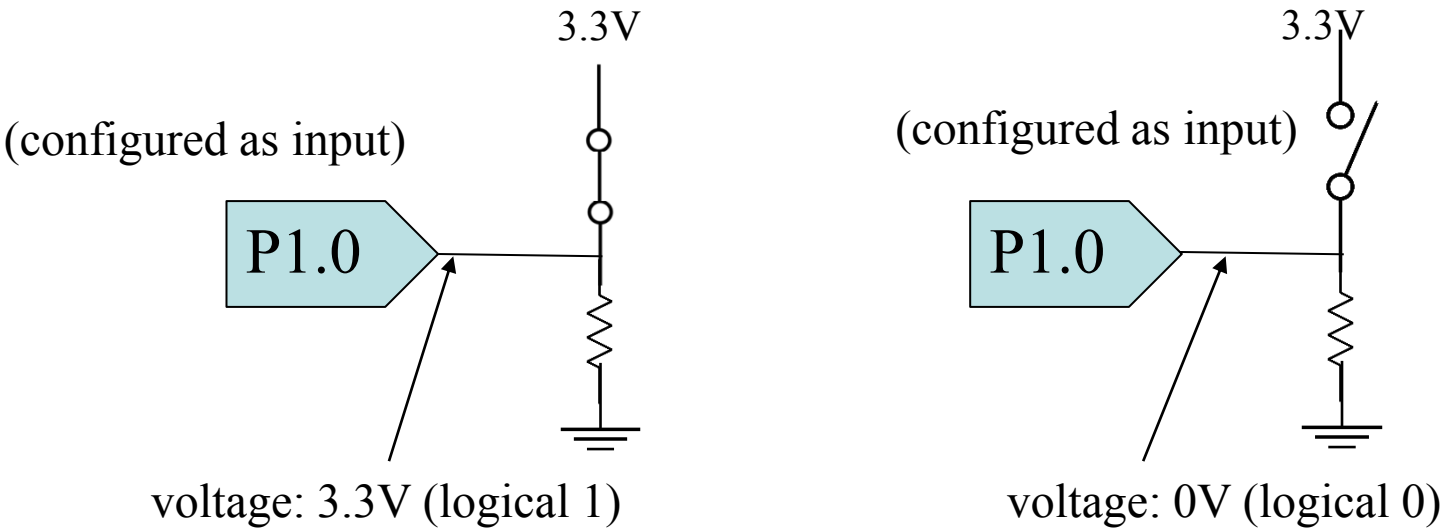
(configured as input)



voltage: ???????

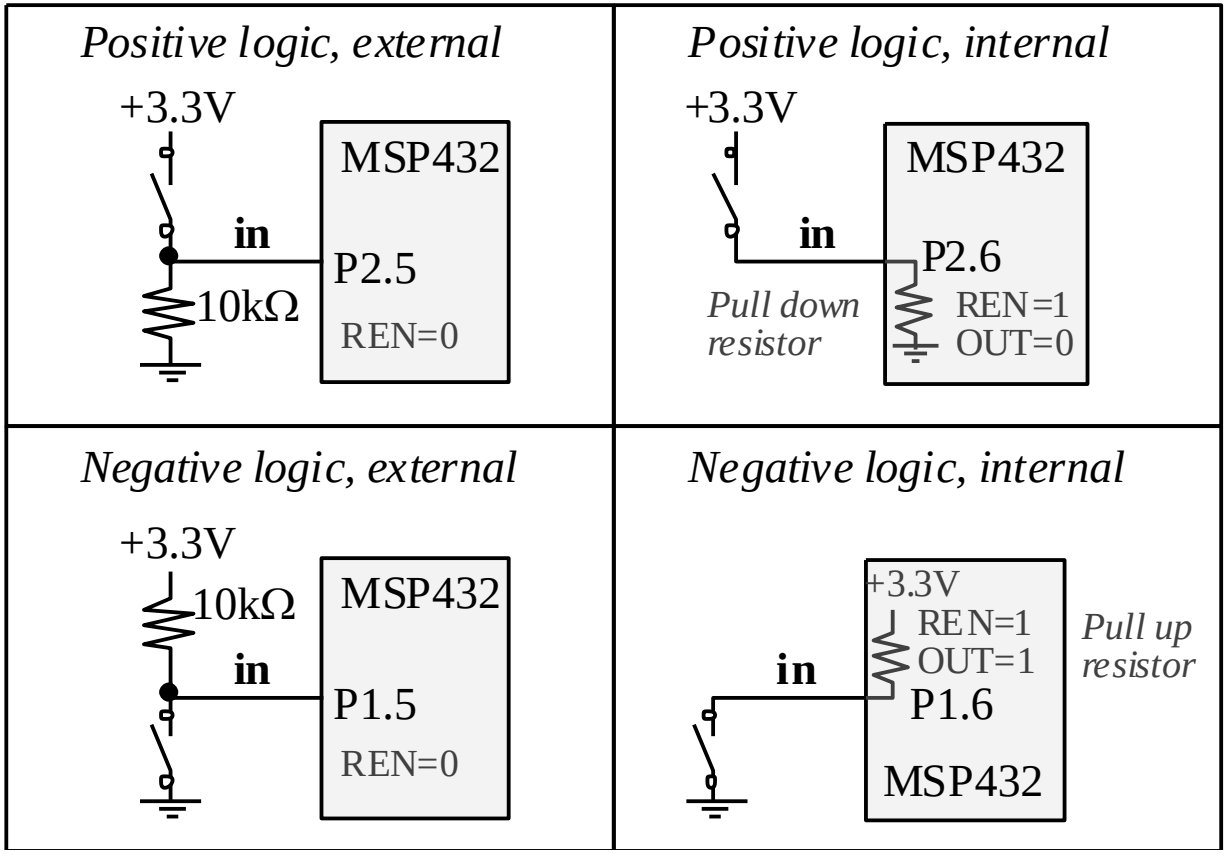
I/O

- PxREN: Resistor Enable
  - Example: Good design (remember, no current flows into the pin)



I/O

- Either you use an **external** pull-up/pull-down resistor…
  - Or you use an **internal** one (configured through PxREN)



I/O

- PxREN
  - If pin is configured as INPUT, PxREN activated internal resistor
    - PxOUT controls which one (PxOUT is dual use)

PxDIR	PxREN	PxOUT	I/O description
0	0	x	Input
0	1	0	Input with pull-down resistor
0	1	1	Input with pull-up resistor
1	x	x	Output

I/O

- Tying it all together…..
  - At the beginning of "main", the first thing that should happen is a call to a function to INITIALIZE GPIO
    - (later, also to functions to initialize other devices)
  - Input/Output, Drive Strength, Resistor, Pull-up/down….

I/O

- Initialization:
  - Configure function in PxSEL0 and PxSEL1 (GPIO)
  - Configure direction in PxDIR
  - If output:
    - Configure drive strength in PxDS
    - Configure default value in PxOUT
  - If input:
    - Configure resistor enable or not in PxREN
    - If enabled, configure up or down in PxOUT
  - For now: disable all interrupts (we'll come back top this)

I/O

- C code that initializes pins 3 and 4 of Port 2 as outputs. Pins are connected to active low circuitry and do not require internal resistors

```
//GPIO function
P2->SEL0 &= (uint8_t)(~((1 << 4) | (1 << 3)));
P2->SEL1 &= (uint8_t)(~((1 << 4) | (1 << 3)));
//Set Output direction
P2->DIR |= (uint8_t)((1 << 4) | (1 << 3));
//Drive pins high (active low circuit: we want it off)
P2->OUT |= (uint8_t)((1 << 4) | (1 << 3));
```

LaunchPad with MSP432

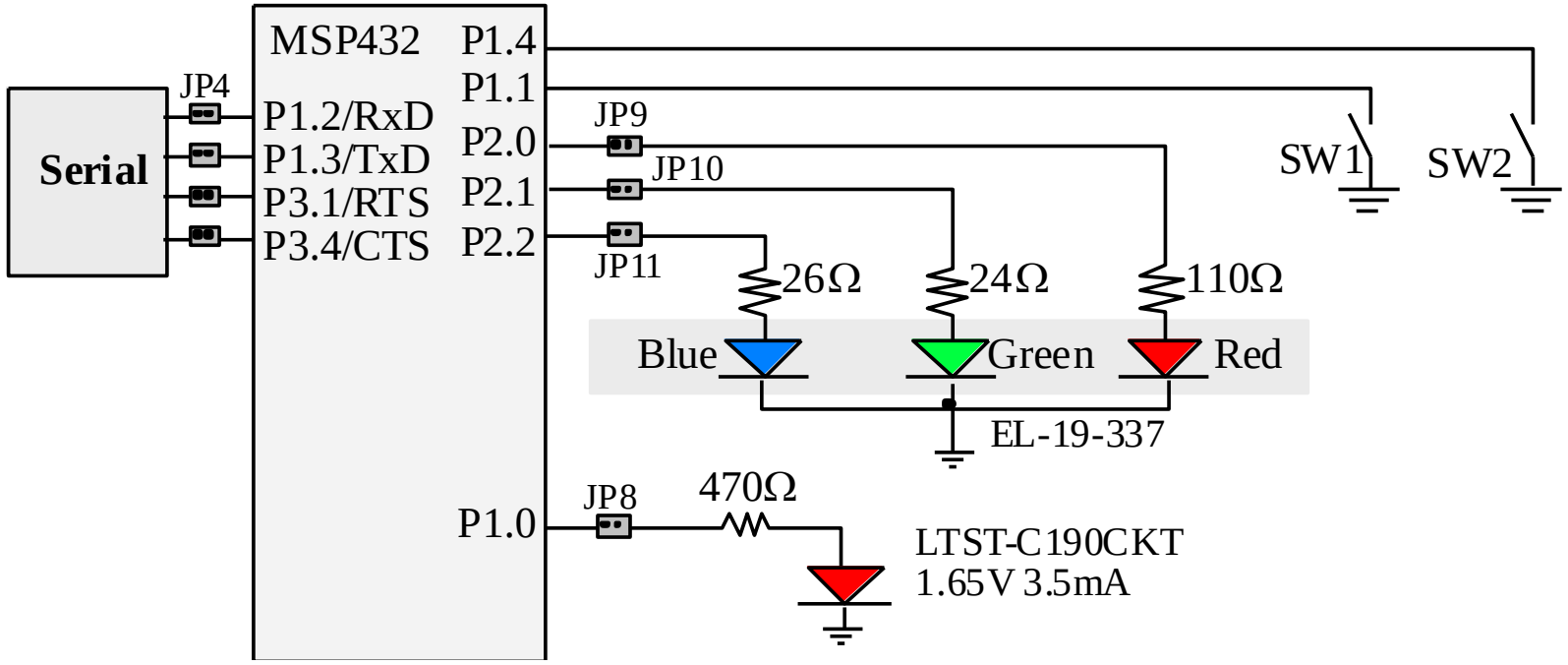
Revision 2.0

Button 1
SW1 (P1.1)
User Programmed

Button 2
SW2 (P1.4)
User Programmed

Red LED
User Programmed
P1.0

RGB LED
User programmed
P2.0..2



- Pay attention to schematic!
  - Switches are ACTIVE LOW and NEED INTERNAL RESISTOR (pull-up)
  - LEDs are ACTIVE HIGH

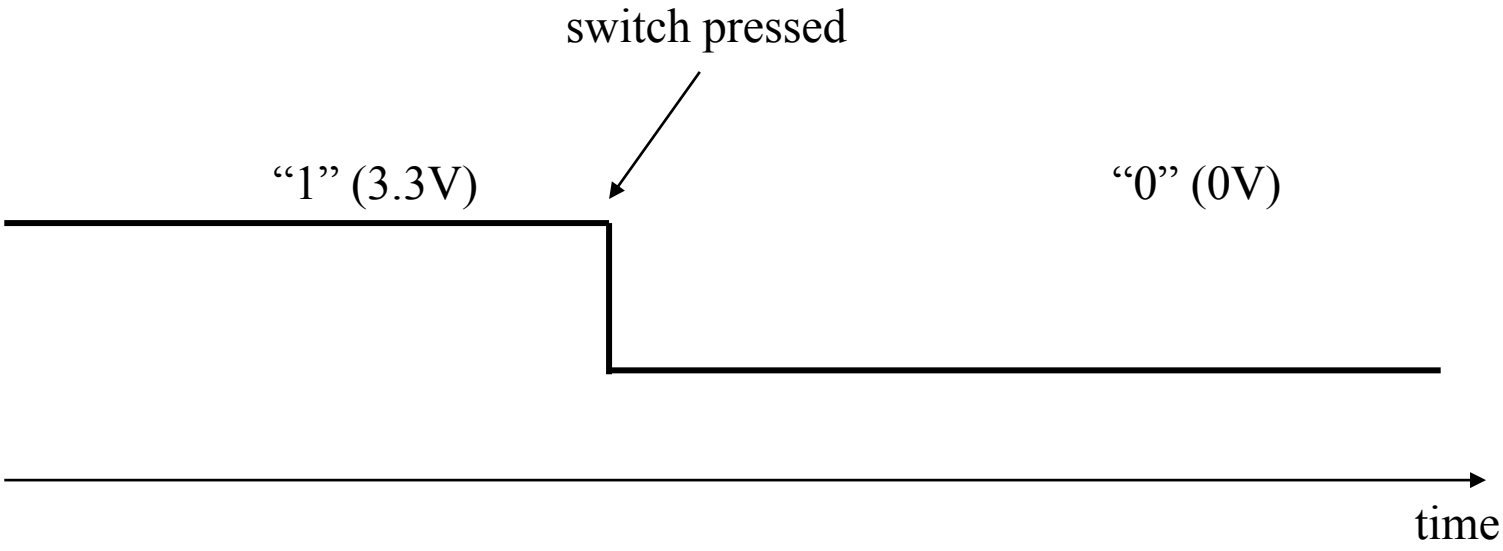
Lab assignment

- Interact with switches and LEDs
  - Switches control LED lighting and patterns
  - Use bit setting, clearing, toggling operators
  - Tying in everything we learned so far

Debouncing

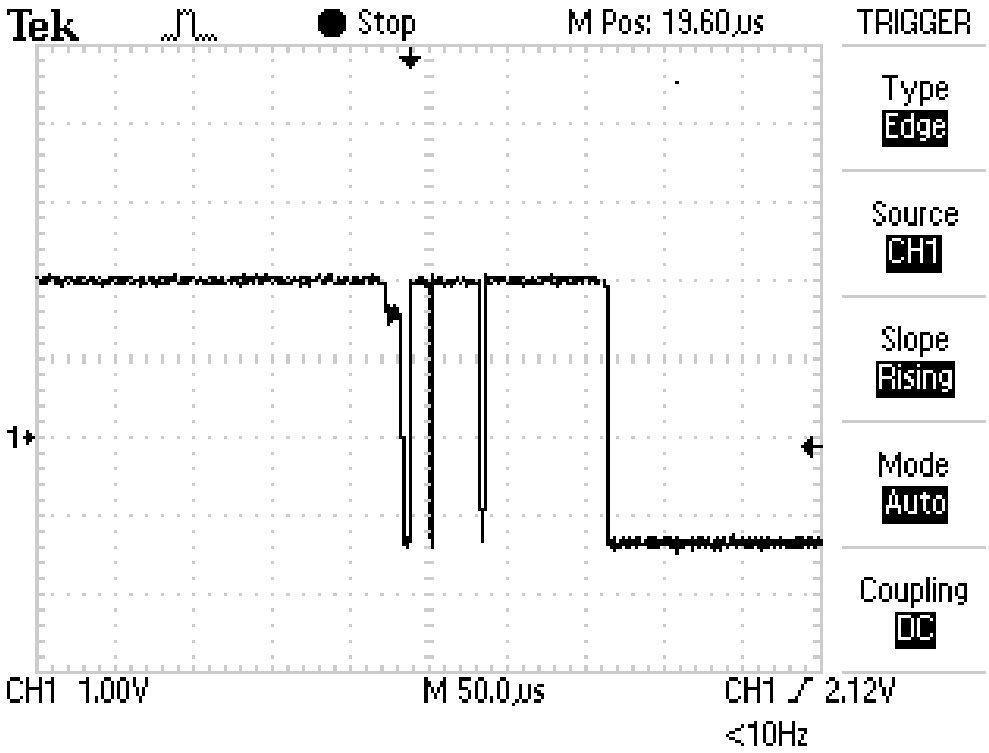
- Remember: there's no such thing as digital

- You'd expect an active low switch would behave like this:



Debouncing

- Actually, it behaves something like this:

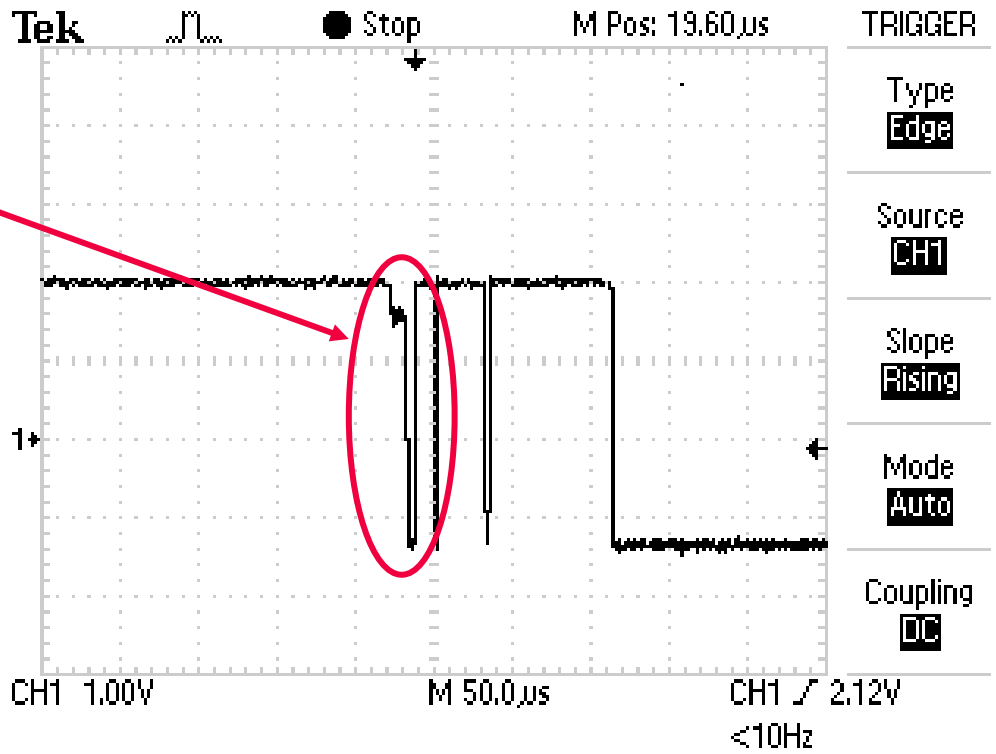


Debouncing

- Why?
  - A switch is a mechanical system
    - Subject to momentum, inertia, elasticity…..
  - It won't behave digitally
  - To treat it as digital, we need a system that "converts" it into digital
    - A way to filter out the analog ugliness

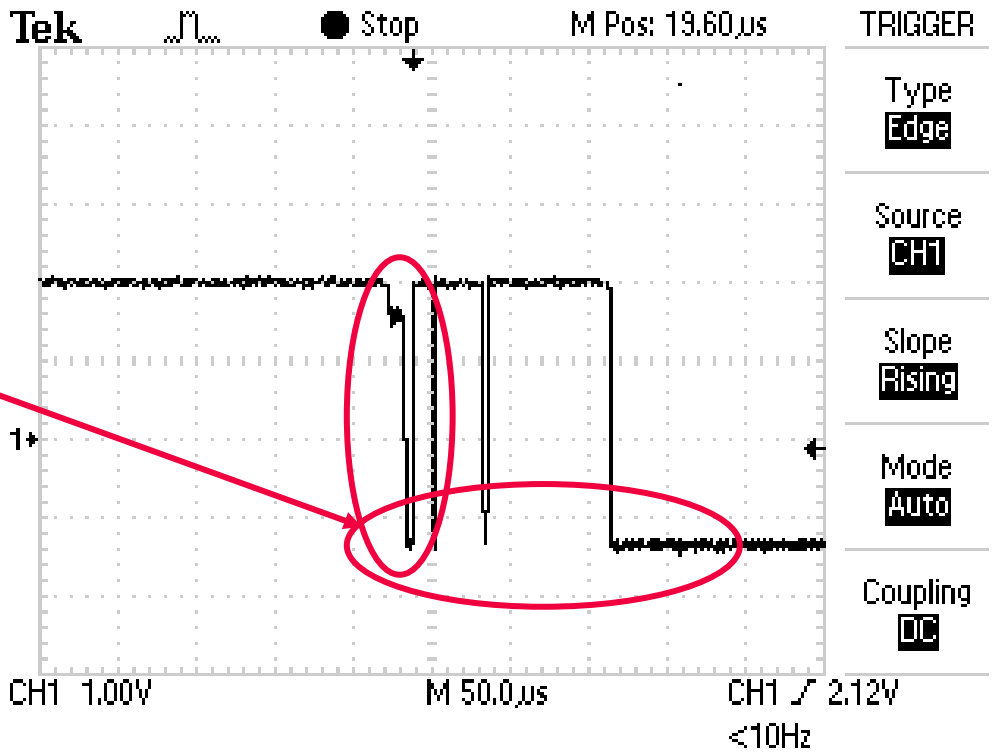
Debouncing

- This is called "switch debouncing"
  - Basic algorithm:
    - 1-Detect the first "0"



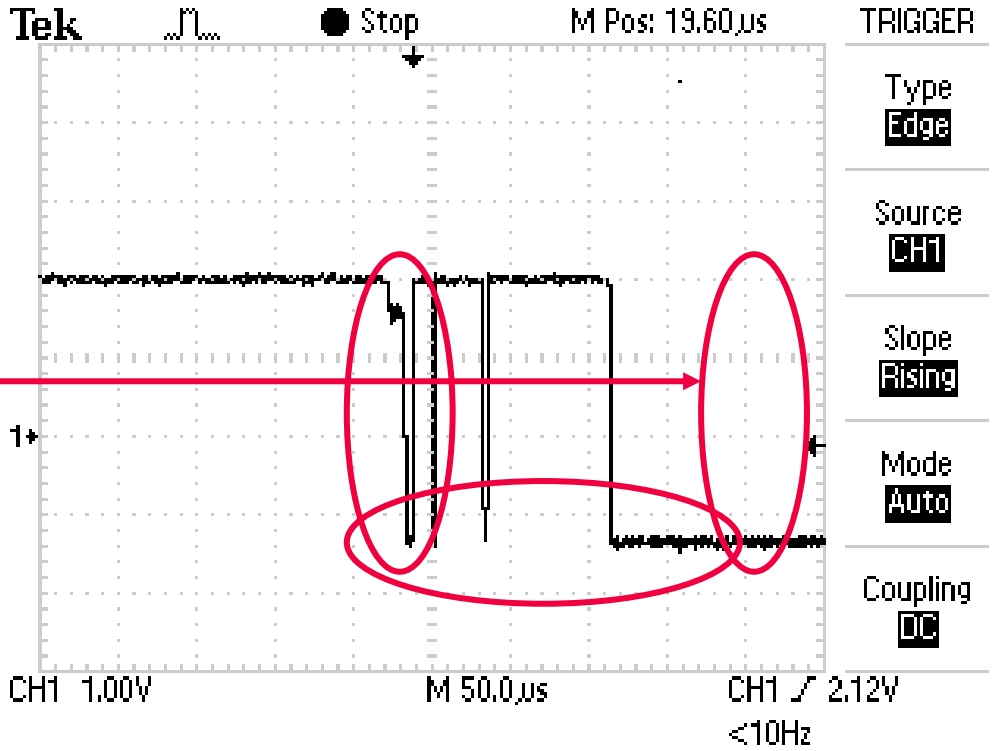
Debouncing

- This is called "switch debouncing"
  - Basic algorithm:
    - 1-Detect the first "0"
    - 2-Wait some time
      - Typically, 100ms



Debouncing

- This is called "switch debouncing"
  - Basic algorithm:
    - 1-Detect the first "0"
    - 2-Wait some time
      - Typically, 100ms
    - 3-Check if it's still "0"
      - If yes, valid
      - If not, back to step 2
        - (or 1 in some cases)



Debouncing

- How to "wait some time"?
  - Later, we'll learn how to use Timers to count the passage of time precisely
  - For now, let's use delay loops

```
int i = DELAY_VALUE;
while(i > 0){i--;}
```

Debouncing

- How to "wait some time"?
  - Later, we'll learn how to use Timers to count the passage of time precisely
  - For now, let's use delay loops

```
int i = DELAY_VALUE;
while(i > 0){i--;}
```

What's this value? 10? 1000? 1000000000000?
How do we determine this?

Debouncing

- Calculating the execution time of software is not easy
  - Cache behavior
  - Compiler optimisations
  - Pipelining
  - …
  - For now, determine it **empirically**
    - **I.e., try different values until it works**

Debouncing

- Creating polling loops
  - Poll – continuously check the value
  - In this case, negative logic (active low)

```
while(1) //infinite loop
{
      i = DELAY_VALUE;
      while(P1.1){} //do nothing while it is "1"
      while(i > 0){i--;} //delay loop
      if(P1.1) //still "1": go back to waiting
            continue;
      else
            //real press: do stuff
}
```

Review

- Watchdog timer
  - Remember to disable it
- How I/O pins work
  - There's no such thing as "digital"
- I/O control registers
  - Pay attentions to active high/low logic, internal resistors….
  - Don't trust the hardware! Initialize registers and default pin values
- Input is tricky
  - Switch debouncing

Next

- Interrupts