05: Interrupts

Analog to Digital Converter

Analog signal gets discretized to some number

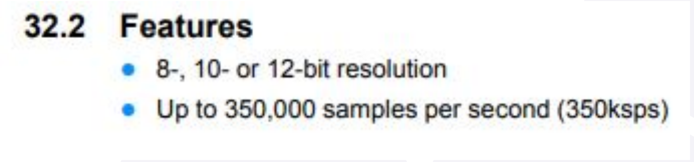
Done via an *Analog to Digital Converter* (ADC)

MCUs have ADC built in

- Different implementations - idea is to compare to reference voltages

Why it matters to embedded software developers: cost, timing, precision

- Find this in the data sheet!



32.2 Features

- 8-, 10- or 12-bit resolution
- Up to 350,000 samples per second (350ksps)

From the SAMD 21 family datasheet

What about outputs?

Digital to analog converter - pin produces voltage from 0 to VCC

Pulse Width Modulation - simulates analog output

Digital to Analog Converter (DAC)

Divide voltage based on digital number

Actuator driven by DAC has similar quantization error to ADC

Some MCUs don't have DACs

- Expensive
- Fewer applications
- Unlike ADC, cannot share

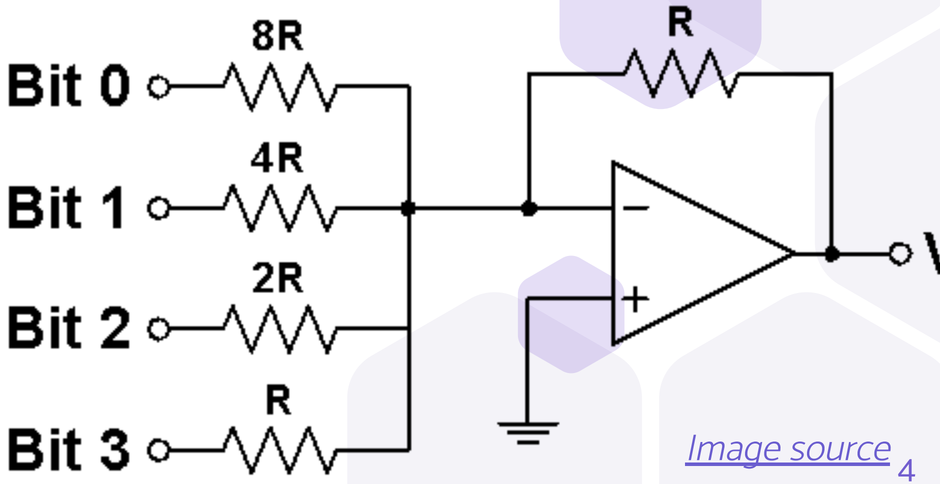


Image source

Pulse Width Modulation (PWM)

Rapidly switch digital pin on and off

Creates perception of analog output

Increasing/decreasing duty cycle increases/decreases perception of power output level

Many microcontrollers provide PWM peripherals

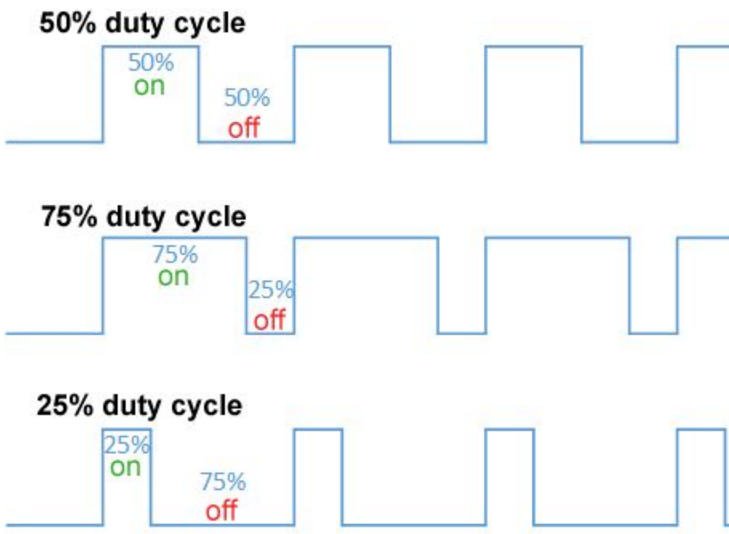


Image source

Digital Signal Processing (DSP)

ADC/DAC/PWM combined with computational power of an MCU has enabled the explosion of digital applications

- Audio, video, robotics, medical…

MCU lets you take in an analog signal, do computations on it, and produce a new analog signal

DSP is a cool area but (mostly) beyond the scope of this course

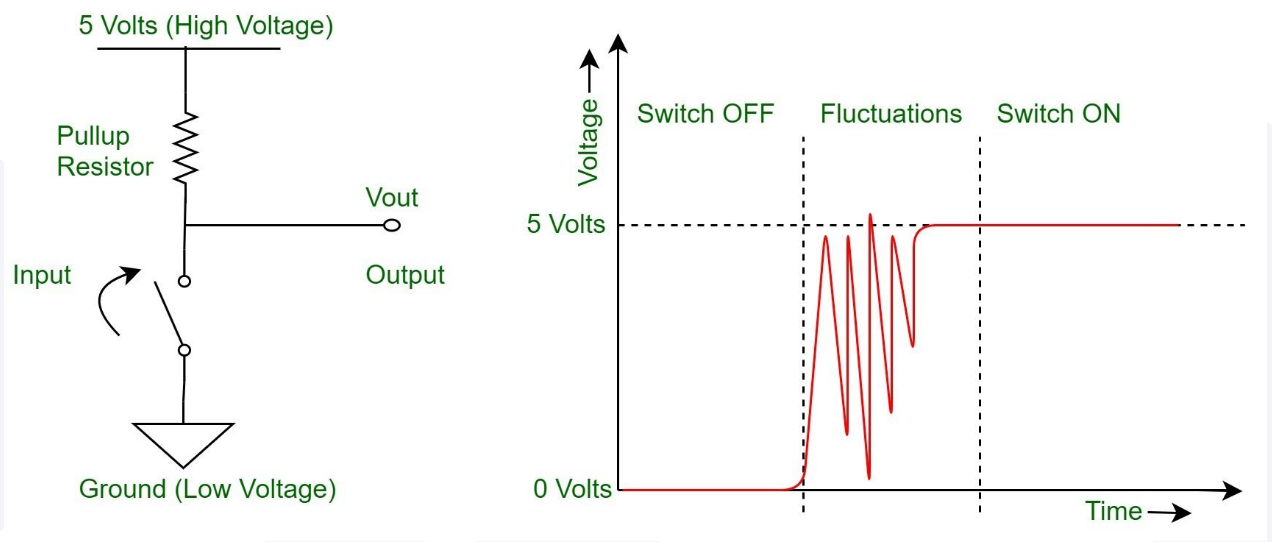
These things are not perfect

Quantization, non-linearity, error in components all contribute to imprecision

DSP can help alleviate some sources of error

Design and models that take sources of error into account are vital for some applications

Example of very simple DSP: debouncing



What do we do here?

Image source

How did we read inputs from sensors in the first lab? What are the upsides and downsides to this method?

Sensors

Polling vs interrupts

Polling: reading input periodically, keep track of changes

Interrupt: be alerted when input changes/does something we are watching for

Real life examples: push notification vs checking texting app, rice cooker alerting you vs manually checking doneness, pet begging for food instead of you checking their bowl...

Types of interrupts

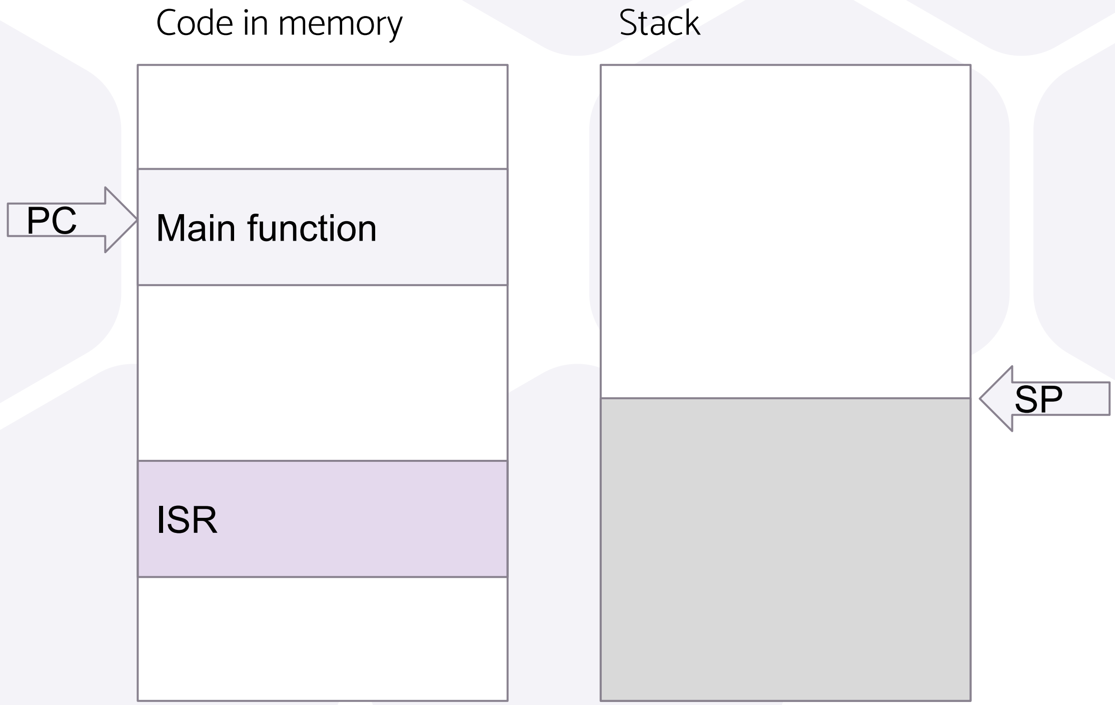
Software interrupts - function is called or some bits are set in a specific memory location to tell the software to go to an *interrupt service routine (ISR)*

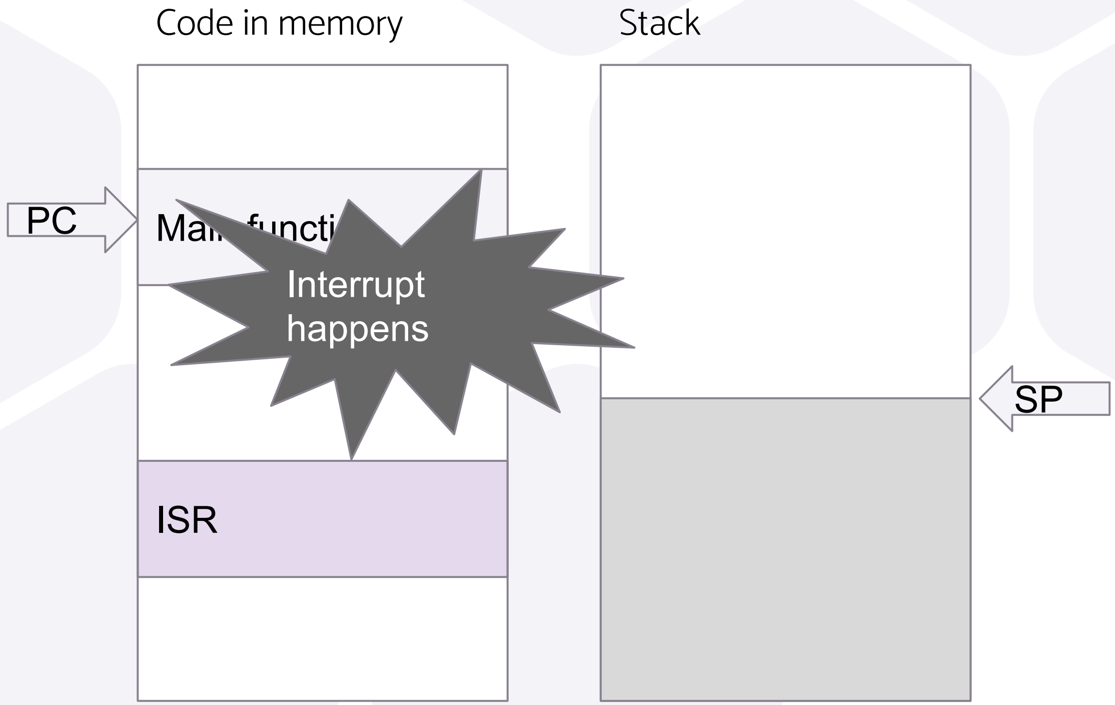
Hardware interrupts - external trigger (voltage change on pin) tells software to go to an ISR

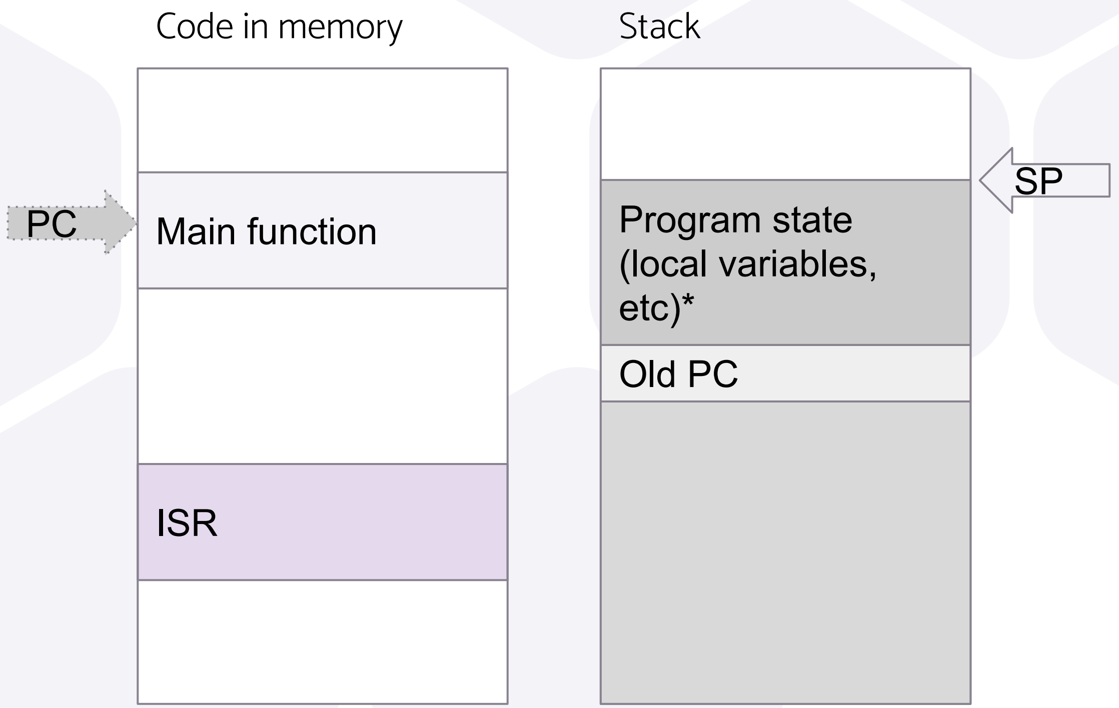
Exceptions - internal trigger (like writing to a protected memory location) triggers a fault

Interrupt process

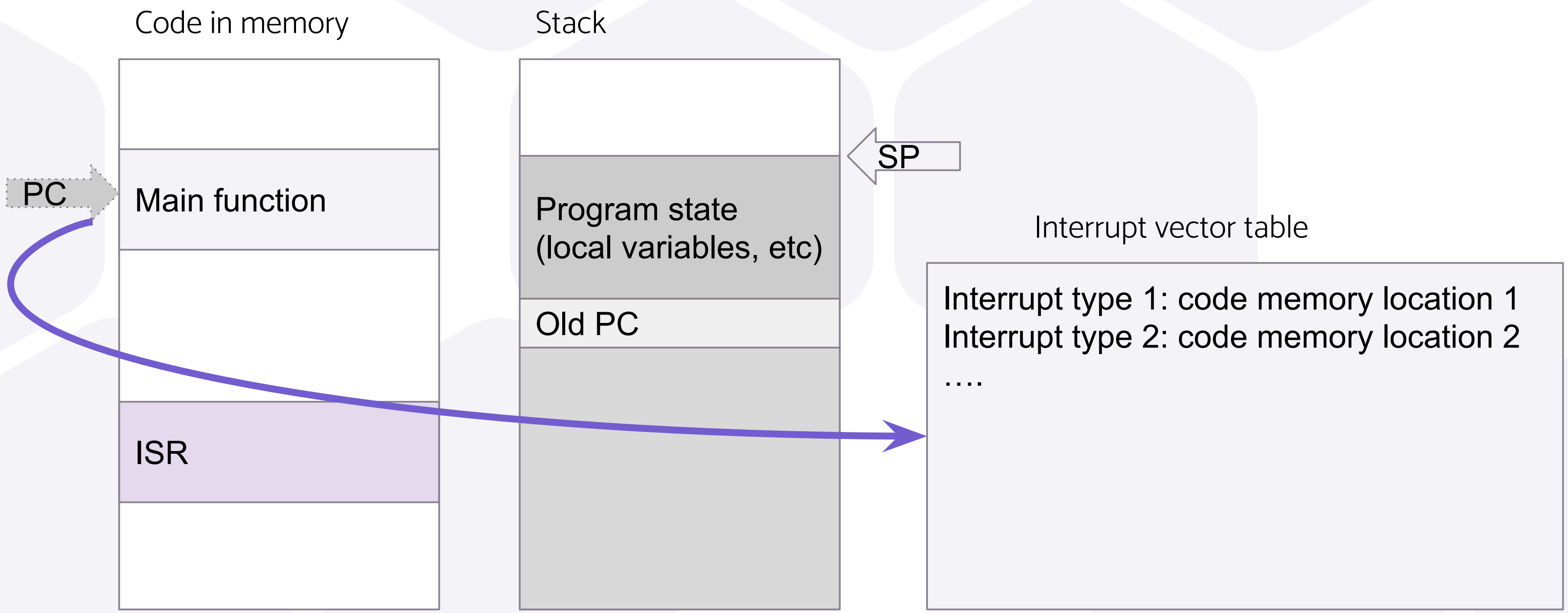
1. Program executing normally
2. Interrupt triggered
3. Processor saves program state
4. Processor enters ISR
5. Processor restores program state
6. Program resumes executing


Code in memory
Stack
PC
Main function
ISR
SP

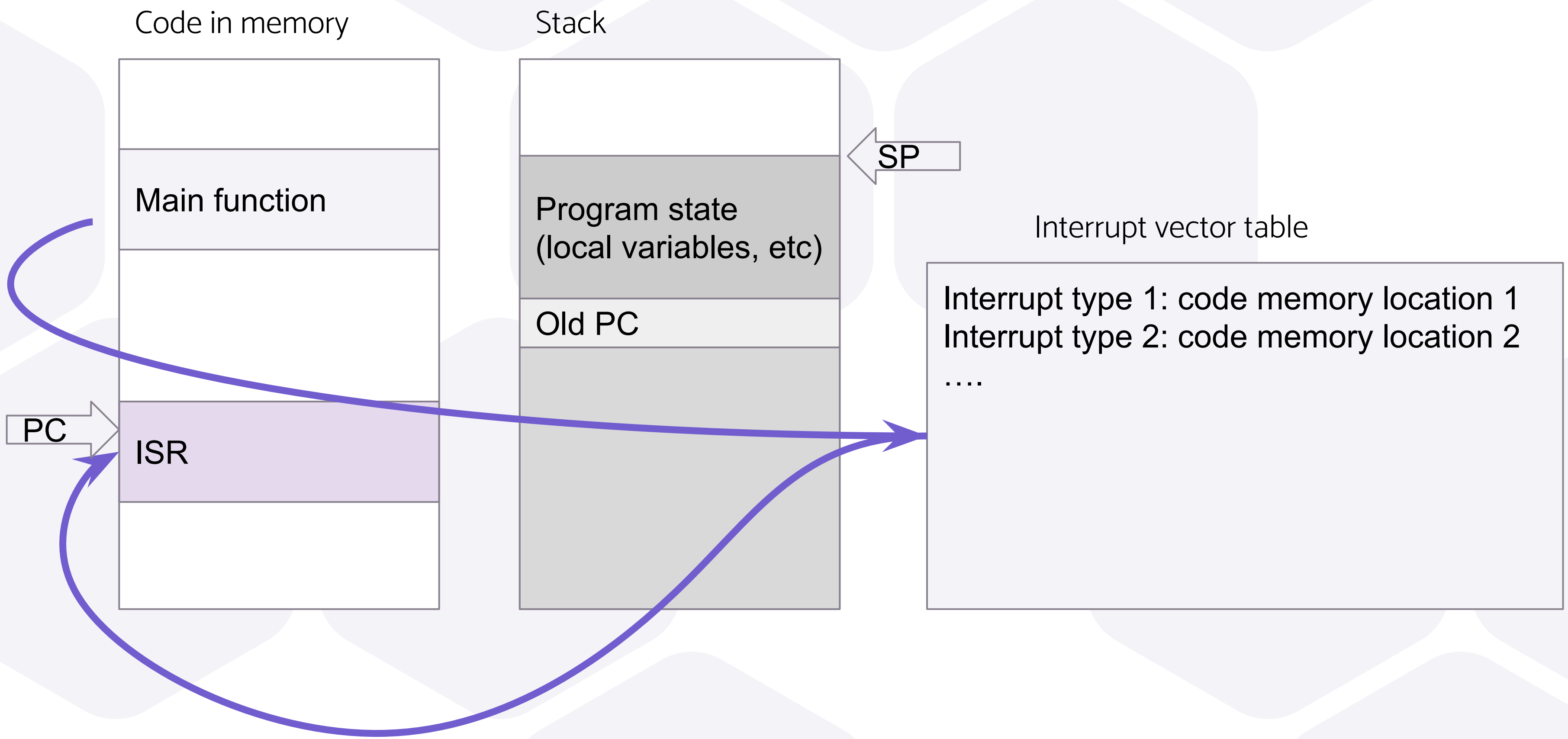

Code in memory
Stack
PC
Main functi
Interrupt happens
ISR
SP



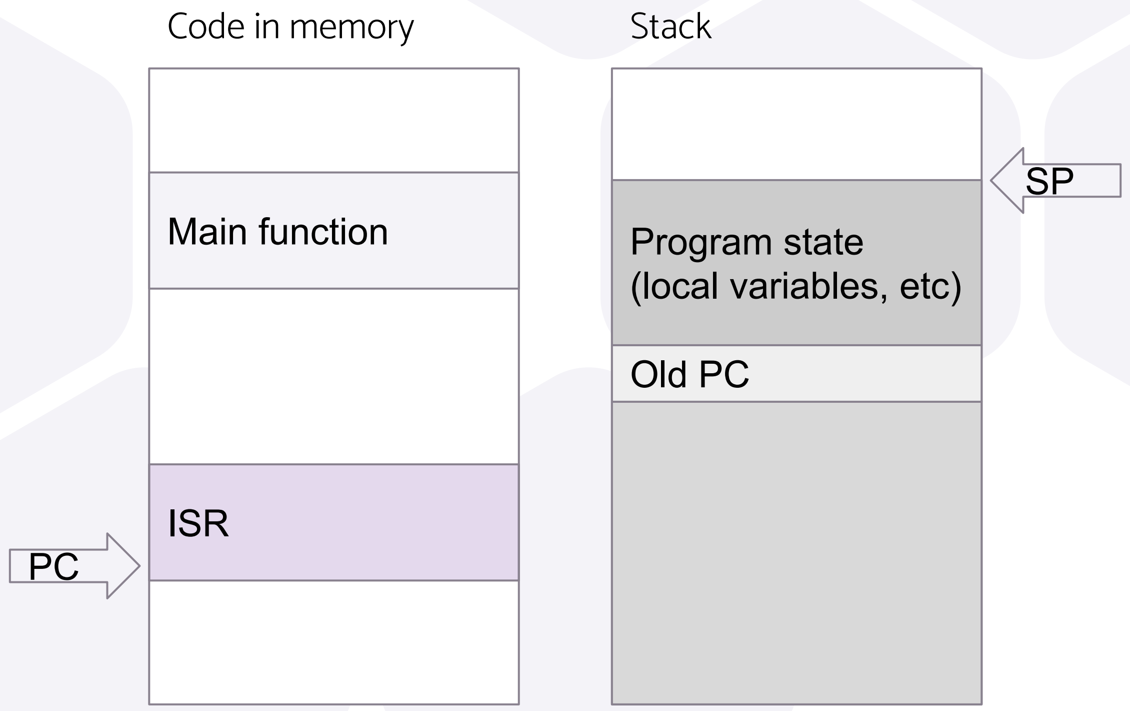
*architecture/MCU-dependent. Sometimes it is up to the programmer to save specific state such as registers

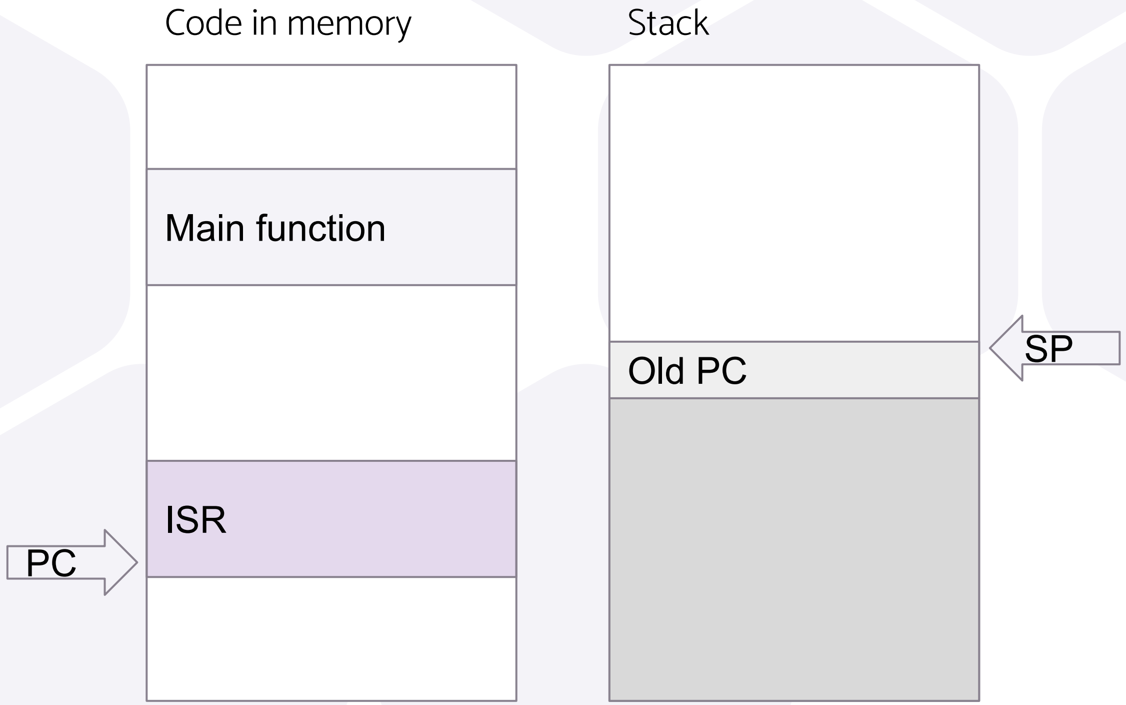


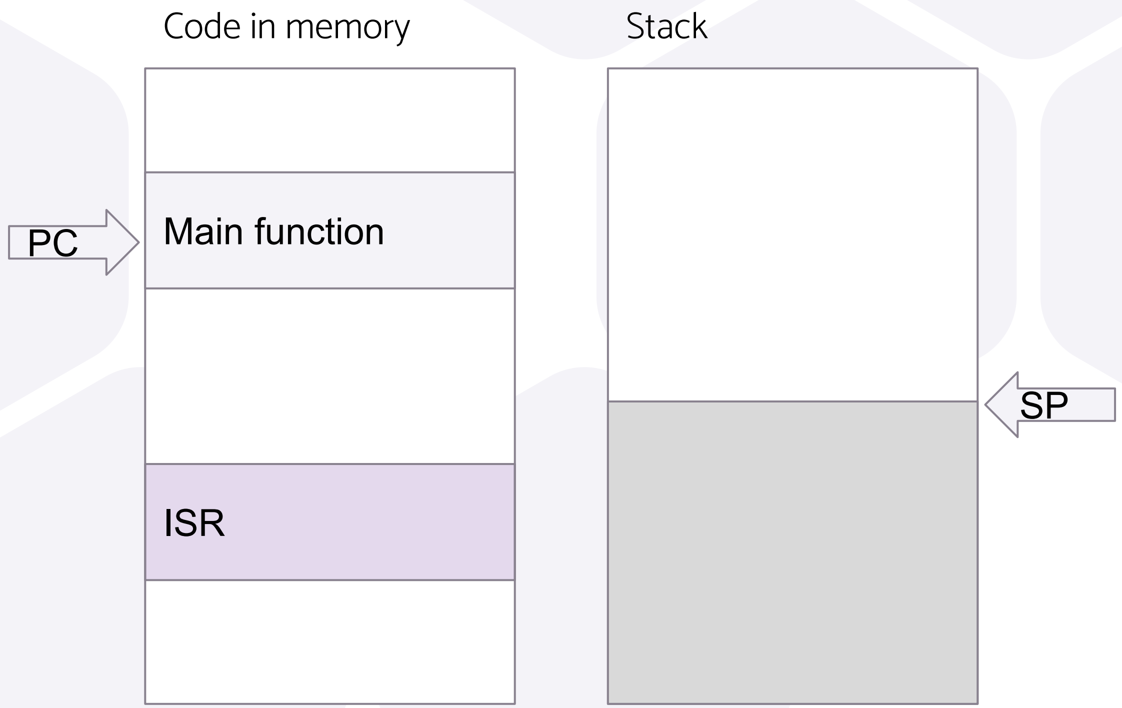
Interrupt type 1: code memory location 1
Interrupt type 2: code memory location 2
….



Interrupt type 1: code memory location 1
Interrupt type 2: code memory location 2
….


Code in memory
Stack
Main function
ISR
PC
SP
Program state
(local variables, etc)
Old PC


Code in memory
Stack
Main function
ISR
PC
Old PC
SP


Code in memory
Stack
PC
Main function
ISR
SP

What if multiple interrupts happen?

Often interrupts are **prioritized**

Higher priority interrupt is allowed to interrupt lower priority interrupt

- Ties broken by position in vector interrupt table

Programmer configures this when setting up code

Implementing ISRs

Often only one handler for some type of interrupt

- Example on SAMD21 (your Arduino MCU): external interrupts share one ISR
- **Why?**

Check flags that are set by MCU to see which pin/peripheral triggered the interrupt -- you will see this in lab 3

What is the difference between an interrupt and a subroutine call?

Interrupts can happen at any time

Subroutine call: you know exactly when in the code you call it

Interrupt: can happen at any point, even “inside” of a command

- Even x = x + 1 is made up of multiple machine instructions (load x from memory, increment x, write value back to memory)

Atomic instructions: values being read/changed in atomic instruction cannot be read/changed by anyone else

Homework problem 10.7

(a) Is it possible for the ISR to update the value of sensor1 while the main function is checking whether sensor1 is faulty? Why or why not?

(b) Suppose a spurious error occurs that causes sensor1 or sensor2 to be a faulty value for one measurement. Is it possible for that this code would not report "Sensor1 faulty" or "Sensor2 faulty"?

(d) Suppose that instead being interrupt driven, ISR and main are executed concurrently, each in its own thread. Assume a microkernel that can interrupt any thread at any time and switch contexts to execute another thread. In this scenario, is it possible for the ISR to update the value of sensor1 while the main function is checking whether sensor1 is faulty? Why or why not?

```
char flag = 0;
volatile char* display;
volatile short sensor1, sensor2;

void ISR() {
  if (flag) {
    sensor1 = readSensor1();
  } else {
    sensor2 = readSensor2();
  }
}

int main() {
  // ... set up interrupts ...
  // ... enable interrupts ...
  while(1) {
    if (flag) {
      if isFaulty2(sensor2) {
        display = "Sensor2 Faulty";
      }
    } else {
      if isFaulty1(sensor1) {
        display = "Sensor1 Faulty";
      }
    }
    flag = !flag;
  }
}
```

Why do we care?

Interrupts are powerful and used widely in embedded programming

Understanding how interrupts affect program state/variables aids in design and debugging

Interrupts complicate modeling and timing analysis of a system (more on this later)