Embedded Systems (6)

- Will start at 15:10
- PDF of this slide is available via ScombZ

Hiroki Sato <i048219@shibaura-it.ac.jp>

15:10-16:50 on Wednesday

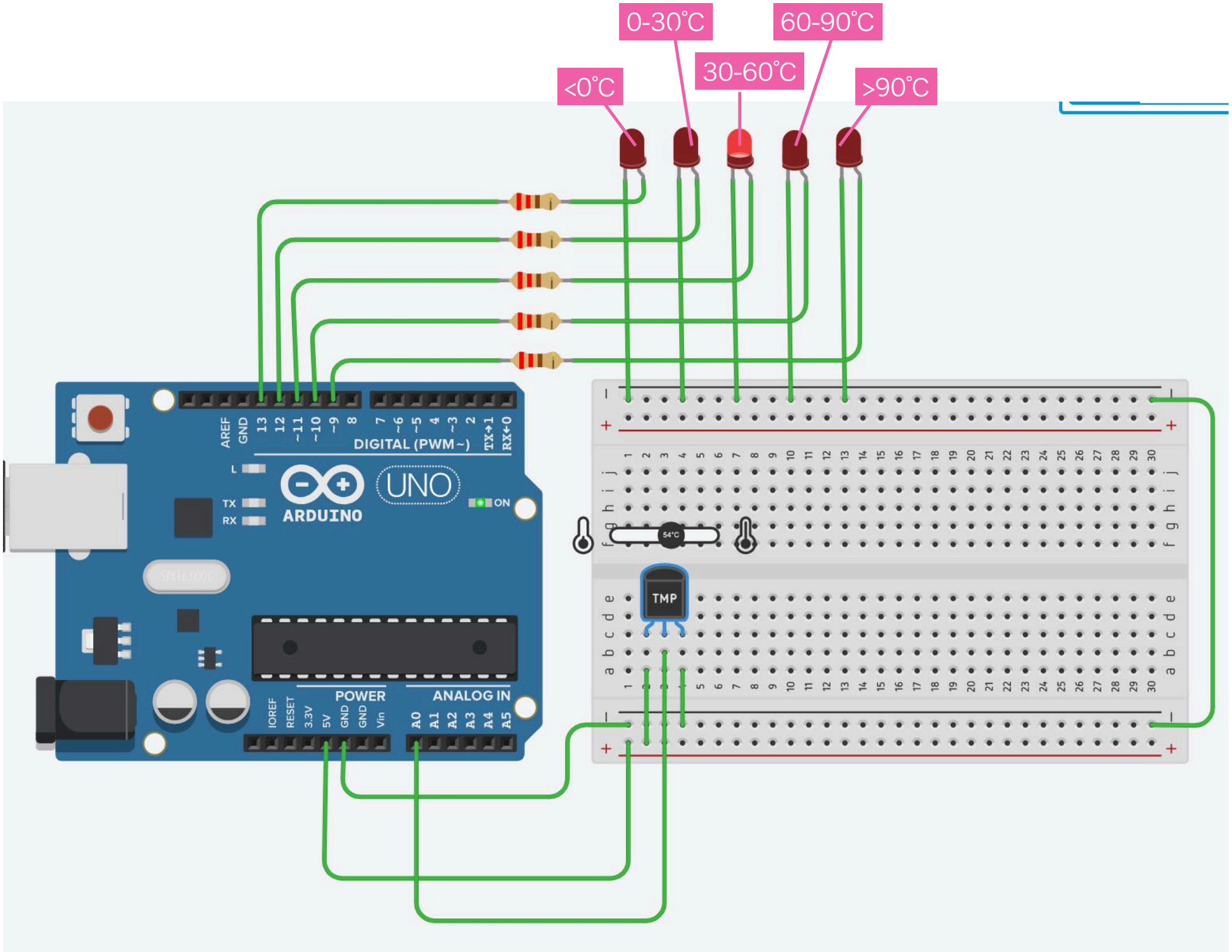
Targets At a Glance

- **What you will learn today**
  - Recap: temperature sensing system
  - Execution context management
    - Timer
    - Interrupt

  - Projects
    - a) Handling a push button as an interrupt source
    - b) Blinker by a timer
    - c) Temperature sensor and blinker by a timer

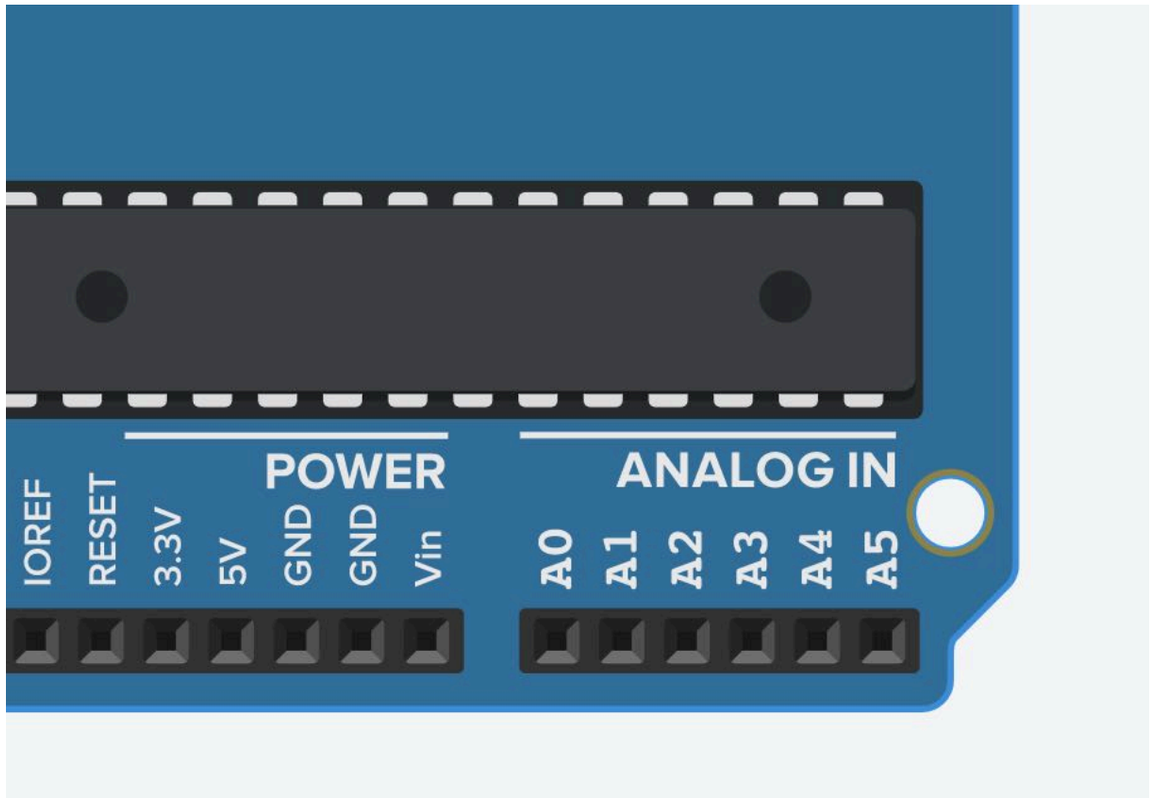
Recap the homework

Temperature Sensing System



Analog Voltage I/O (ADC)

```
void loop()
{
    int temp;

    temp = analogRead(A0);
    Serial.println(temp)
    delay(1000);
}
```



- Analog-to-Digital convertor (not GPIO)
  - 10-bit resolution: analogRead() returns 0 to 1023.
- TMP36 has 5-V full-scale; 0 means 0 V, 1 means (5.0/1024.0) V, and...
  - When 25°C, Vout of TMP36 will be 750mV, 10mV/°C

Temperature Sensing System

```
/* Temperature sensor with 5 LEDs */

int pos = 9;
float temp;

void setup()
{
  int i;

  for (i = 9; i <= 13; i++)
    pinMode(i, OUTPUT);
  Serial.begin(9600);
}
```

TMP36 returns a voltage of
temperature * 10 mV + 500 mV

```
void loop()
{
  int i;

  temp = (analogRead(A0) * 5.0 / 1024.0 - 0.5)
           * 100.0;
  Serial.println((int)temp);  /* for debug */
  if ((int)temp < 0) {
    pos = 13;
  } else if ((int)temp < 30) {
    pos = 12;
  } else if ((int)temp < 60) {
    pos = 11;
  } else if ((int)temp < 90) {
    pos = 10;
  } else {
    pos = 9;
  }
  for (i = 9; i<= 13; i++) {
    digitalWrite(i, LOW);
  }
  Serial.println(pos); /* for debug */
  digitalWrite(pos, HIGH);
  delay(1000);
}
```

analogRead() returns
5/1024 volts as 1.

Convert the temperature
to a pin number.

Why 1024, instead of 1023?

- **Common mistake is "analogRead() / 1023.0"**
  - The result is in a range from 0 to 1023
  - The full scale is 5V, so
    - 0 means (5/1024)x0 to (5/1024)x1
    - 1 means (5/1024)x1 to (5/1024)x2
    - …
    - 1023 means (5/1024)x1023 to (5)
  - (5/1024) V is "1 LSB"

analogRead() returns 5/1024 volts as 1.

```
void loop()
{
  int i;

  temp = (analogRead(A0) * 5.0 / 1024.0 - 0.5)
           * 100.0;
  Serial.println((int)temp);  /* for debug */
  if ((int)temp < 0) {
    pos = 13;
  } else if ((int)temp < 30) {
    pos = 12;
  } else if ((int)temp < 60) {
    pos = 11;
  } else if ((int)temp < 90) {
    pos = 10;
  } else {
    pos = 9;
  }
  for (i = 9; i<= 13; i++) {
    digitalWrite(i, LOW);
  }
  Serial.println(pos); /* for debug */
  digitalWrite(pos, HIGH);
  delay(1000);
}
```

Execution Context Management

Execution Context

```
/* Blinker with 5 LEDs */

void loop()
{
  digitalWrite(pos, HIGH);
  delay(1000);
  digitalWrite(pos, LOW);
  pos++;
  pos = pos % 14;
  if (pos < 9)
    pos = 9;

  Serial.println(pos);
}
```

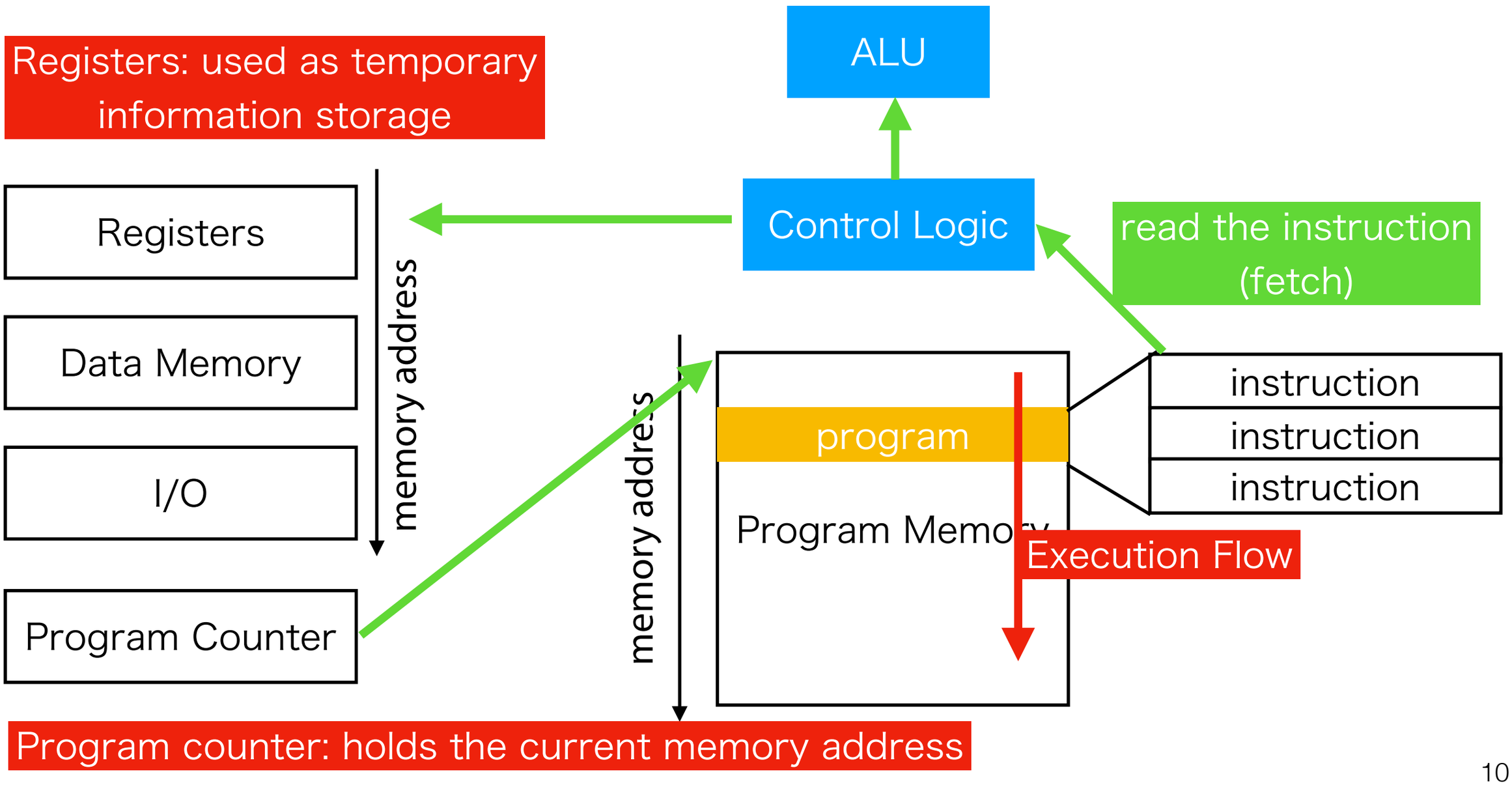
- **A single "program counter" means a single execution flow of the program.**

- Processor's basic behavior: *fetch an instruction, run it, and go the next.*

- An Execution context: *a set of the "processor status" including registers, stacks, and etc.*

- It is often called as "thread"; the loop() function runs as a thread.

Execution Context

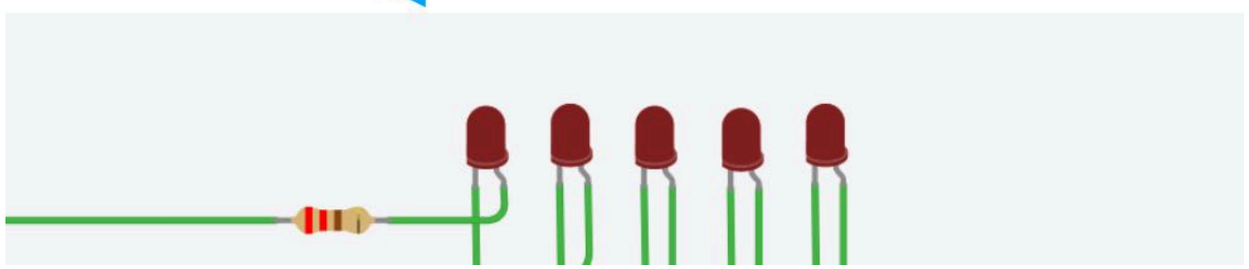


Single Thread Model

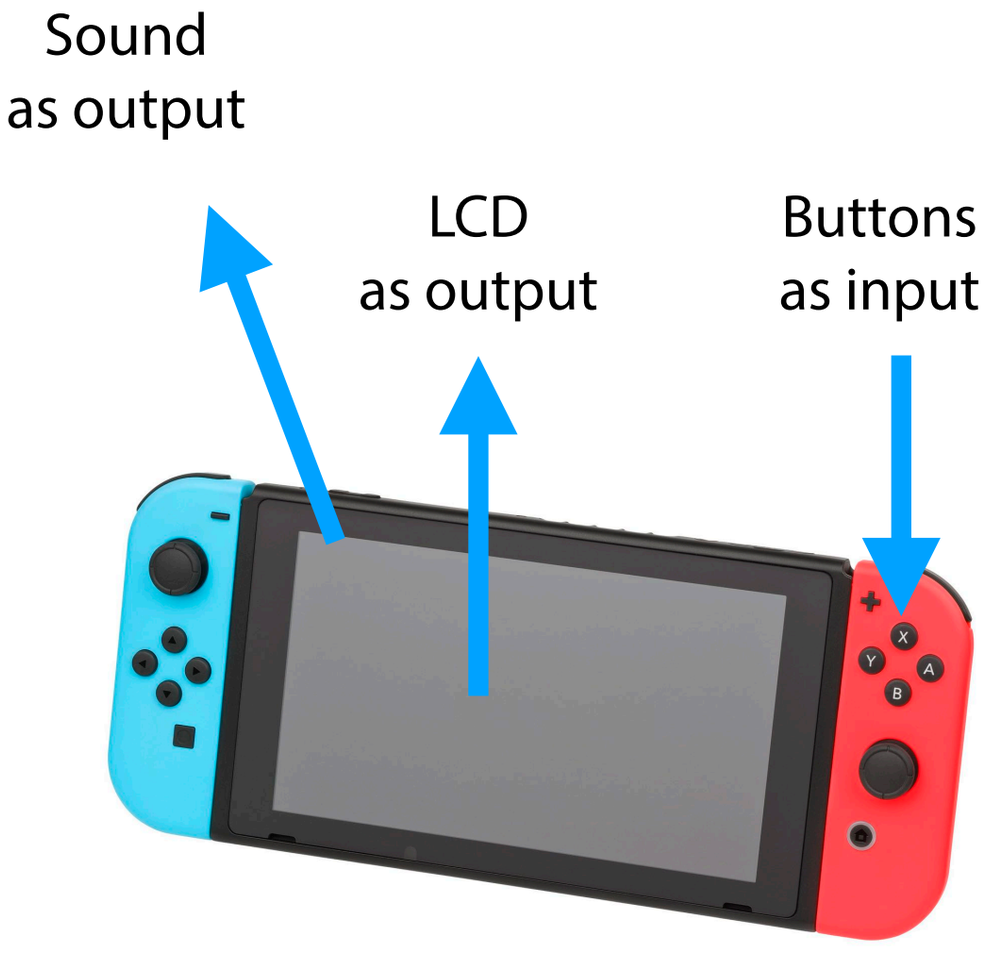
```
/* Blinker with 5 LEDs */

void loop()
{
  digitalWrite(pos, HIGH);
  delay(1000);
  digitalWrite(pos, LOW);
  pos++;
  pos = pos % 14;
  if (pos < 9)
    pos = 9;

  Serial.println(pos);
}
```

- Everything you want to do is handled in a single thread.
- A thread runs *sequentially.*

blinking in this direction with 1s interval

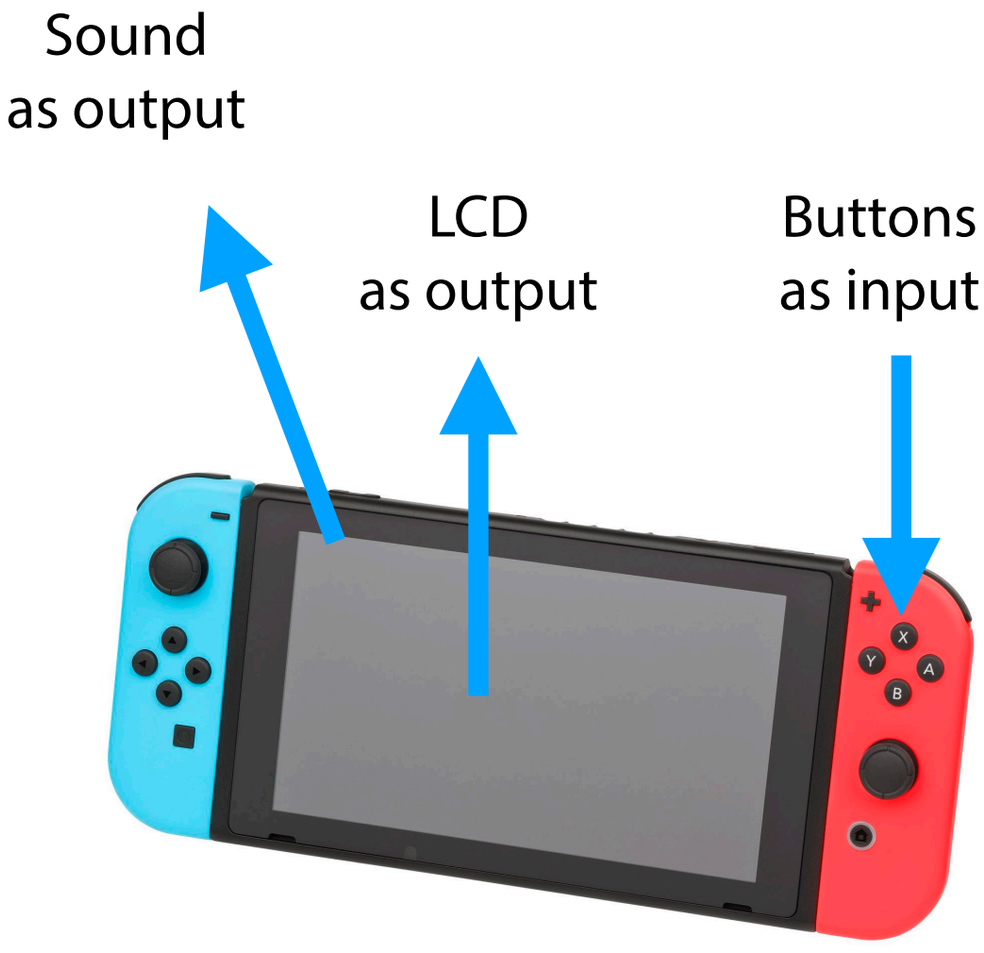


Single Thread Model



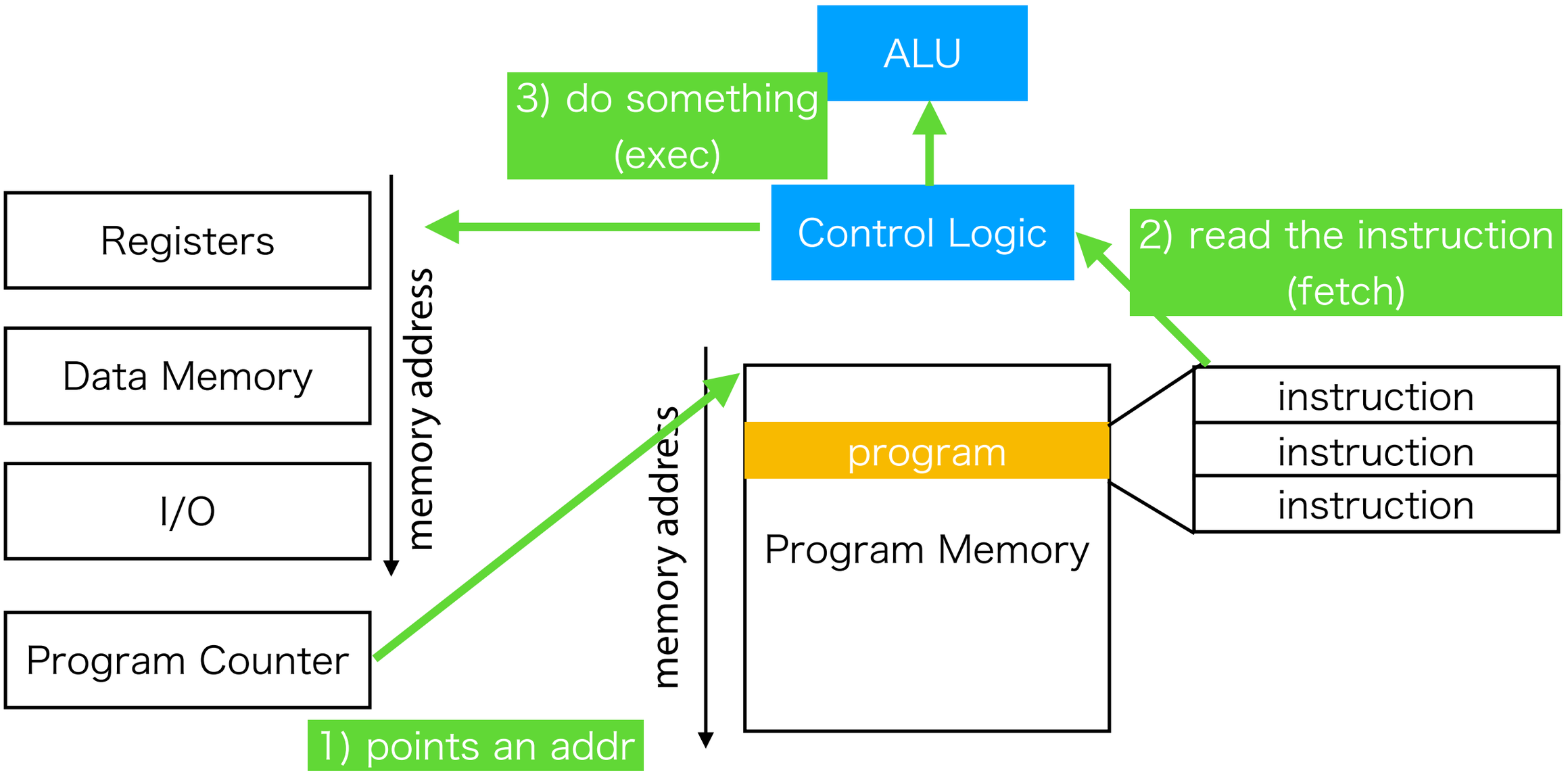
- Everything you want to do is handled in a single thread.
- A thread runs *sequentially*.
- What if your goal is a complex behavior involving multiple input/output devices?

Event-Driven Context Switching



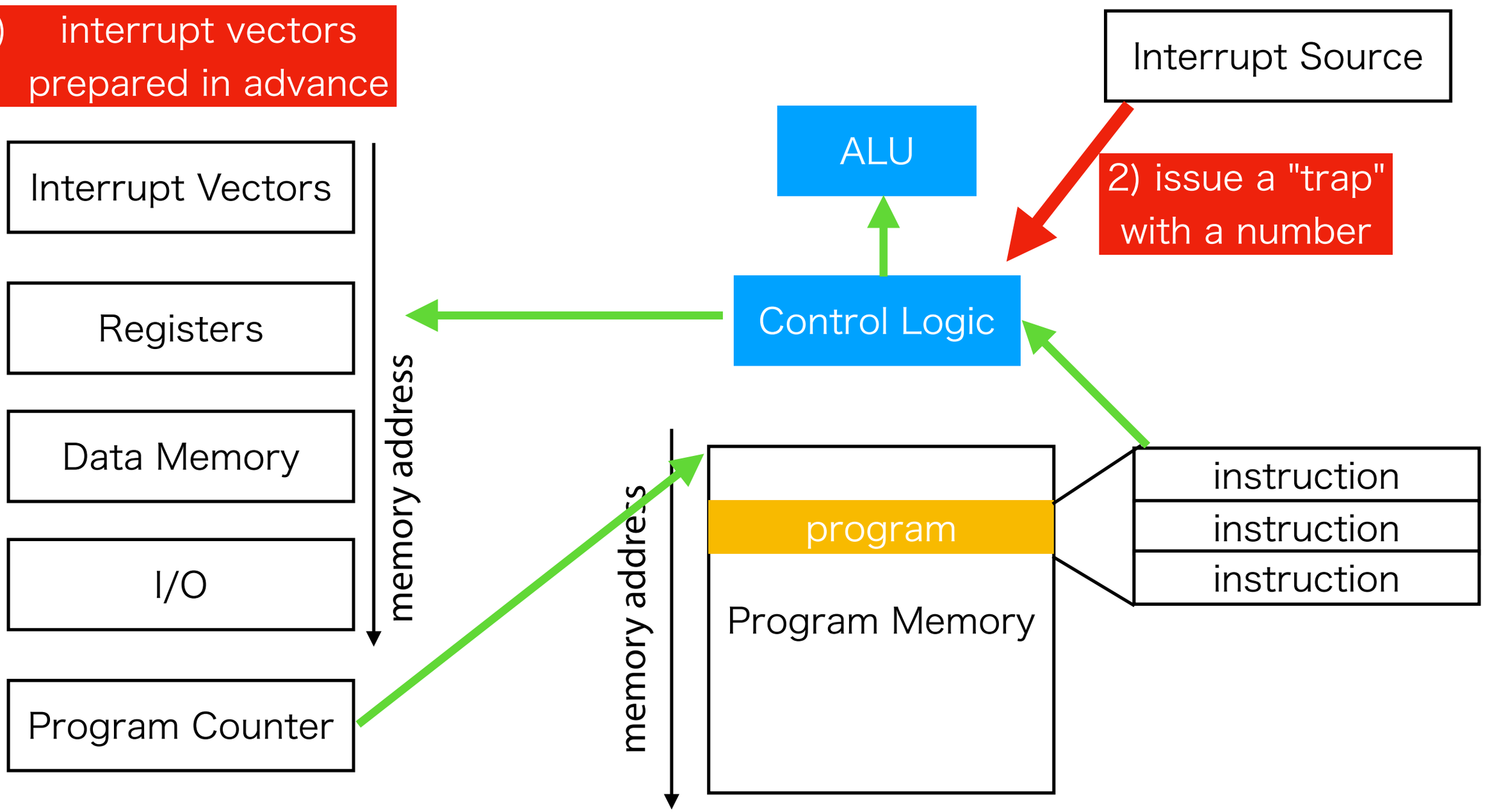
- Execution happens by an event.
- An event means arrival of a specific information such as state of the button.
- "**Interrupt:"** processor's capability to realize event-driven execution context switching.

Interrupt

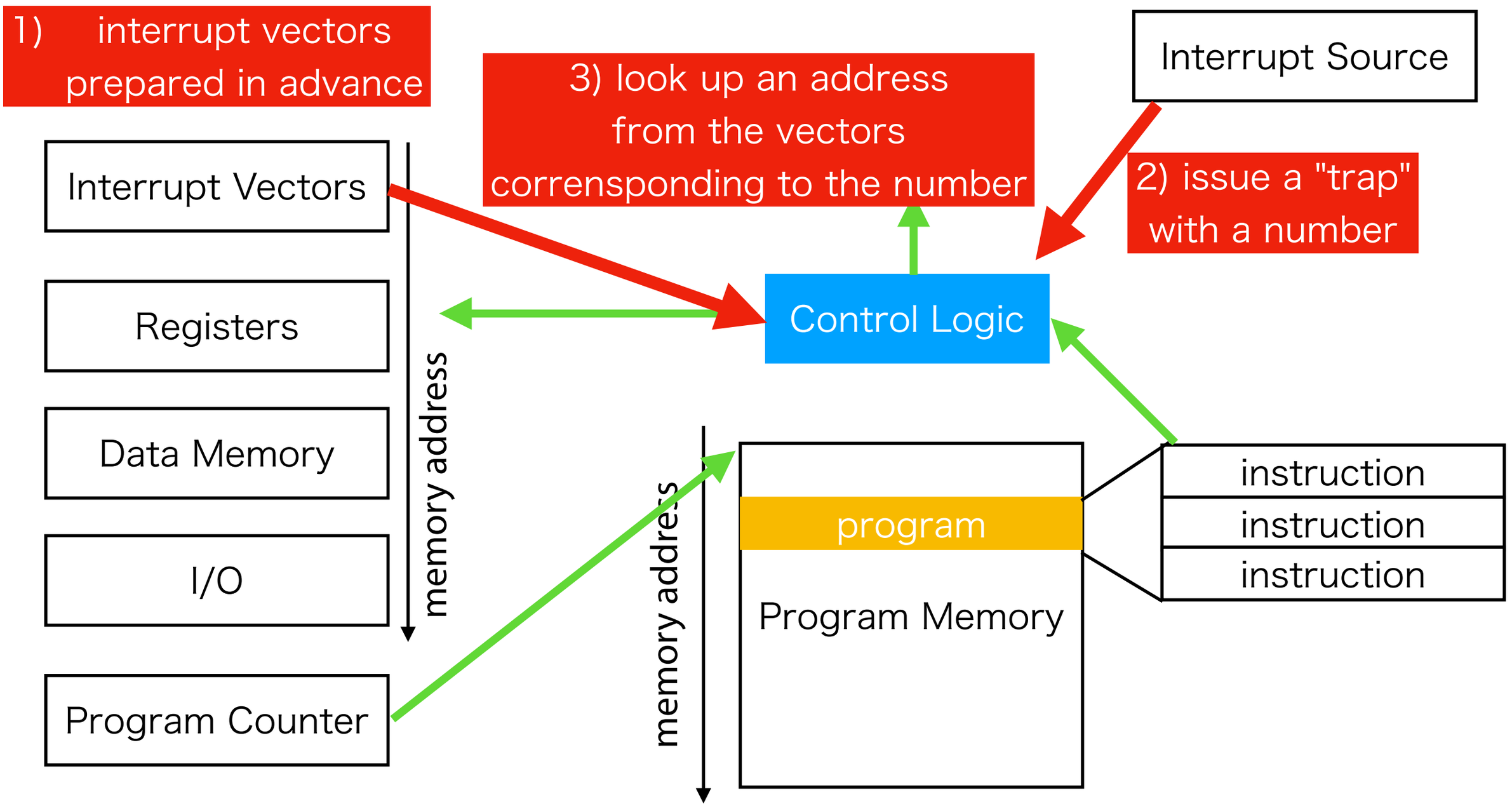

ALU
3) do something
(exec)
Control Logic
2) read the instruction
(fetch)
Registers
Data Memory
I/O
Program Counter
memory address
memory address
program
Program Memory
instruction
instruction
instruction
1) points an addr

Interrupt

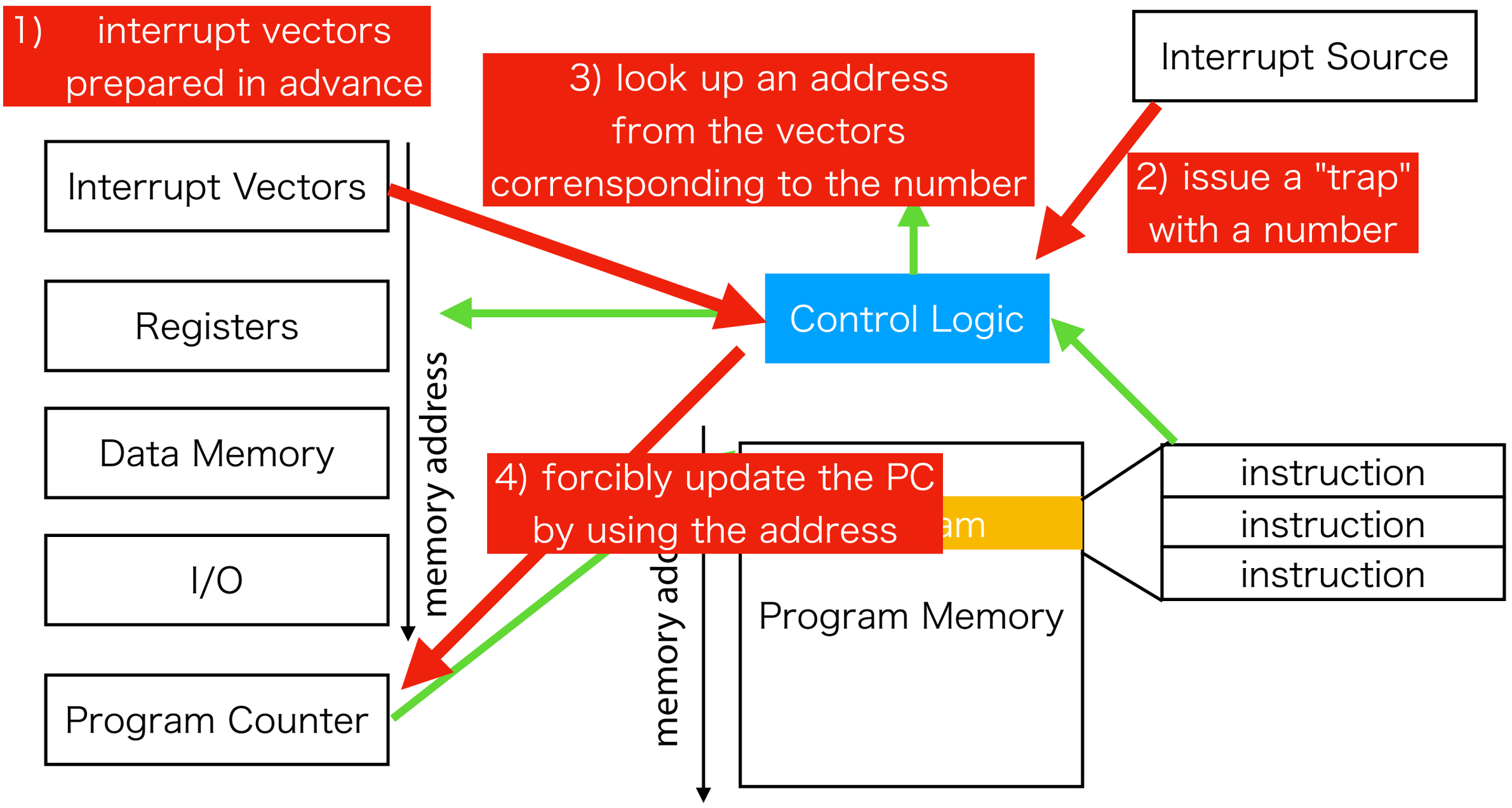
1) interrupt vectors prepared in advance



Interrupt


1) interrupt vectors prepared in advance
Interrupt Vectors
Registers
Data Memory
I/O
Program Counter
memory address
3) look up an address from the vectors correnspondingto the number
Interrupt Source
2) issue a "trap" with a number
Control Logic
memory address
program
Program Memory
instruction
instruction
instruction

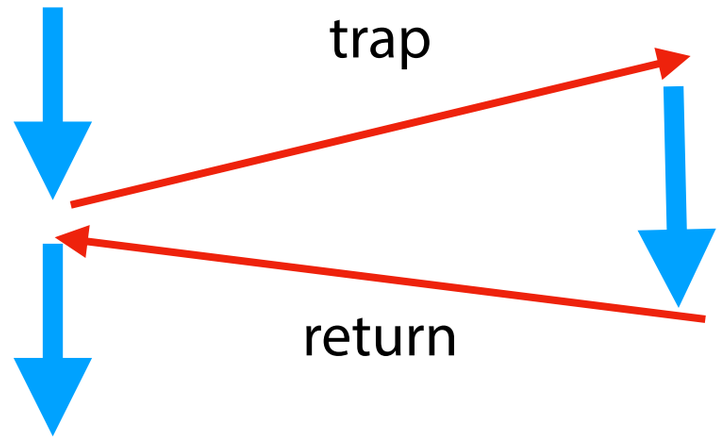
Interrupt


1) interrupt vectors prepared in advance
Interrupt Vectors
Registers
Data Memory
I/O
Program Counter
memory address
3) look up an address from the vectors correnspondingto the number
Interrupt Source
2) issue a "trap" with a number
Control Logic
4) forcibly update the PC by using the address
memory add
Program Memory
instruction
instruction
instruction

Interrupt

```
/* Blinker with 5 LEDs */

void loop()
{
  digitalWrite(pos, HIGH);
  delay(1000);
  digitalWrite(pos, LOW);
  pos++;
  pos = pos % 14;
  if (pos < 9)
    pos = 9;

  Serial.println(pos);
}
```

The main thread




```
void intsr()
{
  // do something
}
```

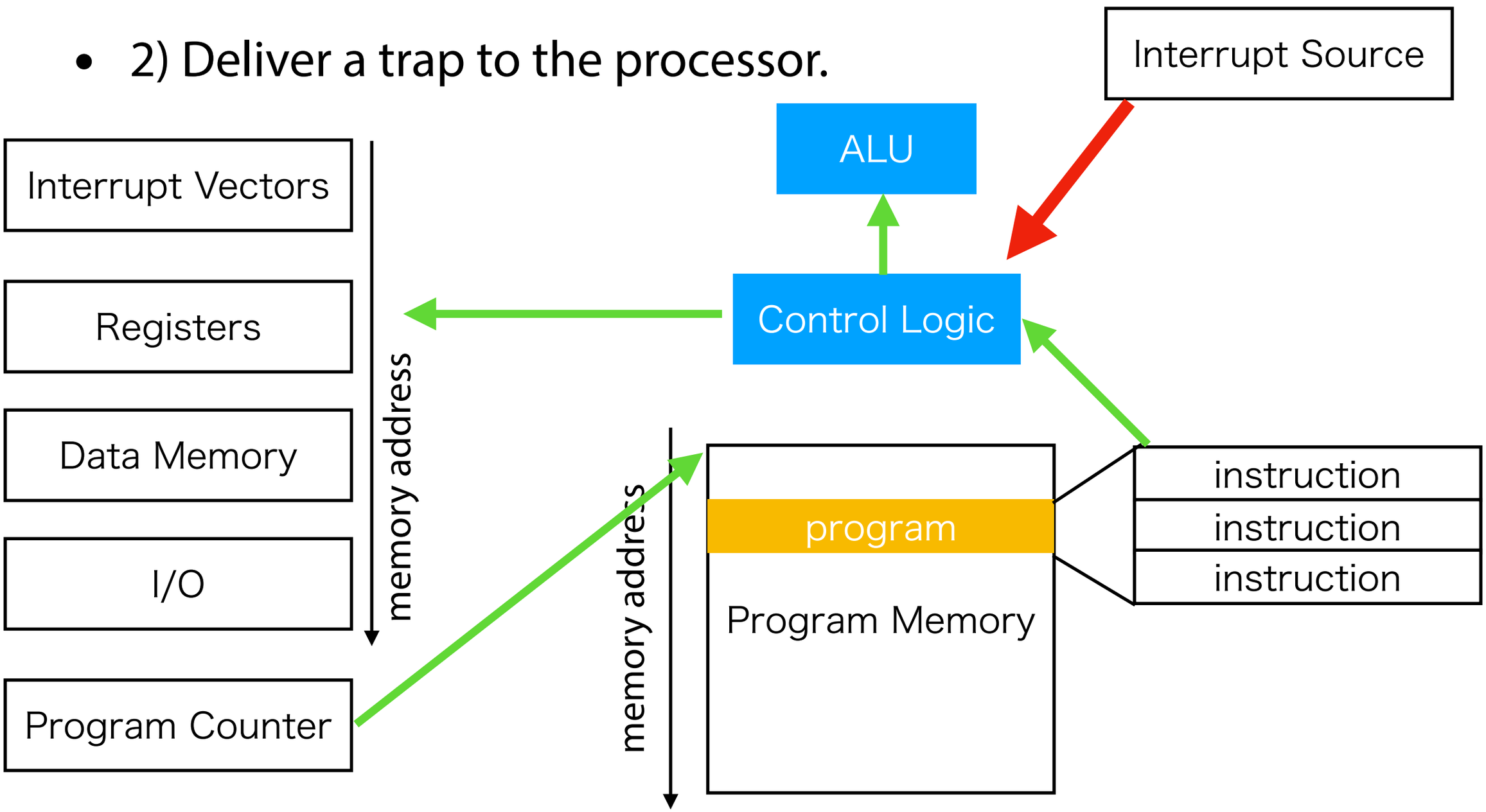
Interrupt Handler:
a function (instructions) at the address of the vector

Interrupt Sources (event)

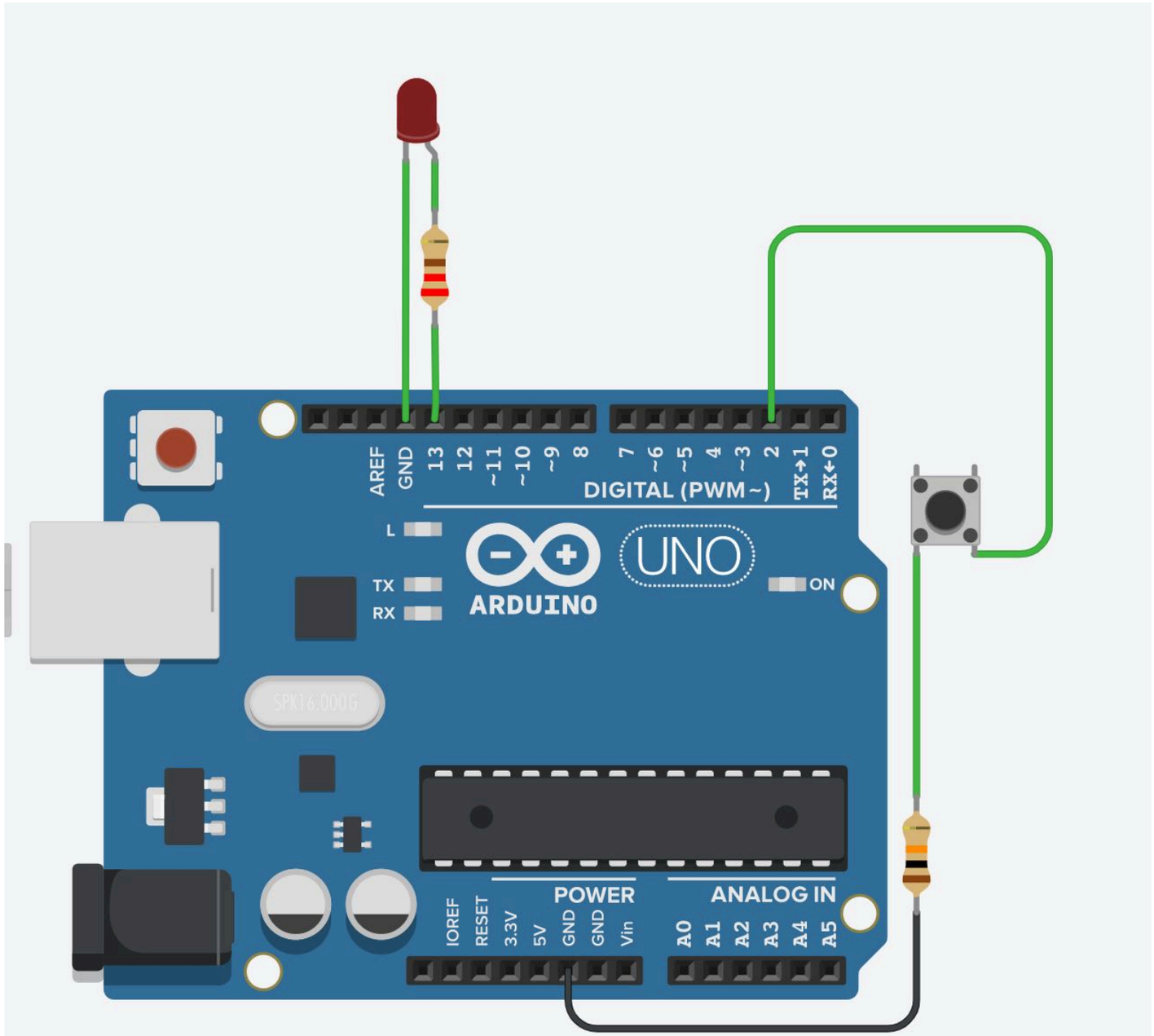
- The processor can accept a trap in various ways. Trap depends on the processor structure.
- Internal trap
  - **Timer**: general-purpose counter logic
  - **Exception**: divided-by-zero, privilege violation, special instruction, etc.
- External trap
  - Change of data on I/O interfaces (digital pin)

How to Use Interrupt

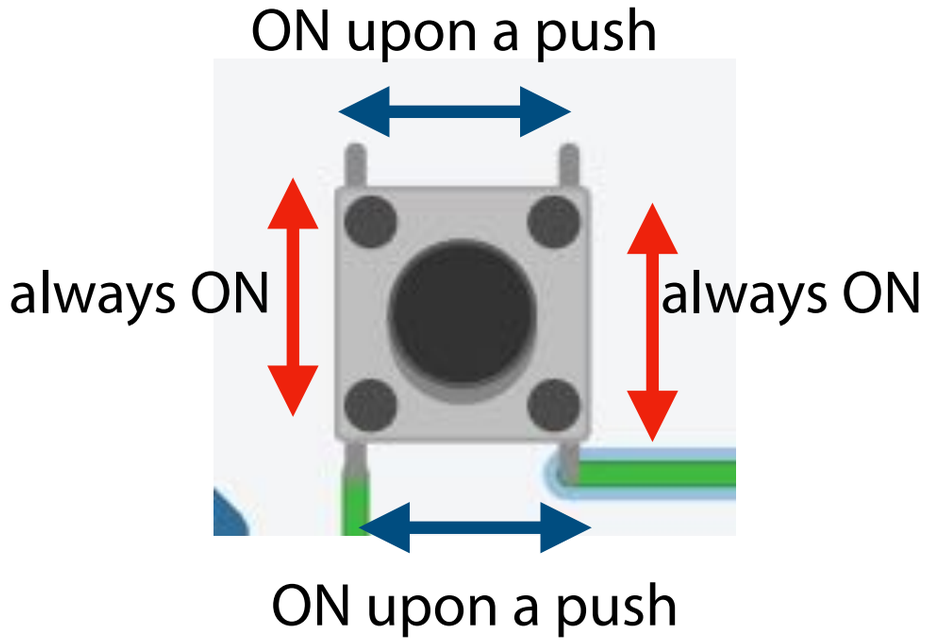
- 1) Register an interrupt handler to the interrupt vector.
- 2) Deliver a trap to the processor.



How to Use Interrupt



- A simple system which realizes that pushing the button makes the LED turn on and off.



Polling

```
int s = 0;

void setup()
{
  pinMode(13, OUTPUT);
  pinMode(2, INPUT_PULLUP);
}
```

```
void loop()
{
  if (digitalRead(2) == LOW)
    digitalWrite(13, (s++ % 2) ? LOW : HIGH);
  delay(100);
}
```

Check the button every 100ms

- A periodic check of the input device is called "polling".
- Difficult to handle another event or processing in loop().

How to Use Interrupt

```
int s = 0;

void setup()
{
  pinMode(13, OUTPUT);
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(2),
    on_push, FALLING);
}

void on_push()
{
  digitalWrite(13, (s++ % 2) ? LOW : HIGH);
}

void loop()
{
  delay(1000);
}
```

Interrupt sources must be pin 2 or pin 3 for this board.

attachInterrupt() registers an interrupt handler for a specific trap.

Interrupt handler for a button push event

How to Use Interrupt

```
attachInterrupt(digitalPinToInterrupt(2), on_push, FALLING);
```

The first parameter is trap number. digitalPinToInterrupt() is a function to convert the pin number to the trap number.

A function pointer of the interrupt handler

Timing of I/O pin voltage change. RISING or FALLING

See https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

How to Use Interrupt

```
int s = 0;

void setup()
{
  pinMode(13, OUTPUT);
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(2),
    on_push, FALLING);
}

void on_push()
{
  digitalWrite(13, (s++ % 2) ? LOW : HIGH);
}

void loop()
{
  delay(1000);
}
```

- An additional execution context makes easy to add more processing in loop() function.

Run upon a button push event

Do nothing in loop()

Interrupt by Timer

- Timer is one of the useful interrupt sources.
- It is a free-running counter and the functionality varies depending on the board and processor. CTC(Clear-Timer-on-Compare) mode is popular.
- In CTC mode, a counter is incremented and reset at a predefined value. Upon the reset, a trap signal is delivered.
- Arduino Uno has 3 timers. Two 8-bit and one 16-bit timer.
- Blinker with 1s interval can be realized by using a timer.

How to Use a Timer

- Timer0, Timer1, and Timer2 are available. Timer0 is used for delay().
- To use them, configure special registers.

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B**

Cited from datasheet p.140

http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf

How to Use a Timer

- Timer1 in CTC mode will work like the following:
  - Increment the counter by 0.0625us (16MHz)
  - **Pre-scaler for the clock** can be configured. 8, 32, 64, 128, 256, or 1024 in **CS register**.
  - If the value is equal to **OCR1A register**, the counter will be reset to zero.
  - If **OCIE1A** of **TIMSK1 register** is set, a trap is delivered on a reset.
  - A trap happens every (CS x 0.0625us) x (OCR1A - 1) seconds.

- Again, this is specific to this processor.

How to Use a Timer

- Mode configuration: TCCR1A and TCCR1B

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

To set CTC mode, TCCR1A = 0 and TCCR1B = (1 << 3)

How to Use a Timer

- Pre-scaler configuration: TCCR1B

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

To set 1/256 prescaler, TCCR1B = (1 << 2)

How to Use a Timer

- Interrupt configuration: TIMSK1

16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

To enable traps, TIMSK1 = (1 << 1)

How to Use a Timer

```
void setup()
{
  TCCR1A  = 0;
  TCCR1B = (1 << WGM12) | (1 << CS12);
  OCR1A   = 31250 - 1;
  TIMSK1 |= (1 << OCIE1A);

  pinMode(13, OUTPUT);
}

ISR (TIMER1_COMPA_vect) {
  digitalWrite(13, !digitalRead(13));
}

void loop()
{
  delay(1000);
}
```

- Macros for register names are available as symbols: WGM12 = 3, CS12 = 2, OCIE1A = 1, etc.
- Interrupt handler for Timer1 must be declared as "ISR (TIMER1_COMPA_vect)" function, not using attachInterrupt().

How to Use a Timer

```
void setup()
{
  TCCR1A  = 0;
  TCCR1B = (1 << WGM12) | (1 << CS12);
  OCR1A   = 31250 - 1;
  TIMSK1 |= (1 << OCIE1A);
  pinMode(13, OUTPUT);
}

ISR (TIMER1_COMPA_vect) {
  digitalWrite(13, !digitalRead(13));
}

void loop()
{
  delay(1000);
}
```

1/256

0.0625us x 256 = 16us

16us x 31250 = 0.5s

Summary

- **Single thread programming model**
  - A processor has only one PC in general.
  - A program in C/C++ programming language runs a single function
  - "polling" is a normal way to handle I/O devices

- **Interrupt**
  - make execution context switching possible. You can have multiple execution contexts for each processing.
  - Multiple "interrupt sources" can be configured
  - Useful for asynchronous event handling

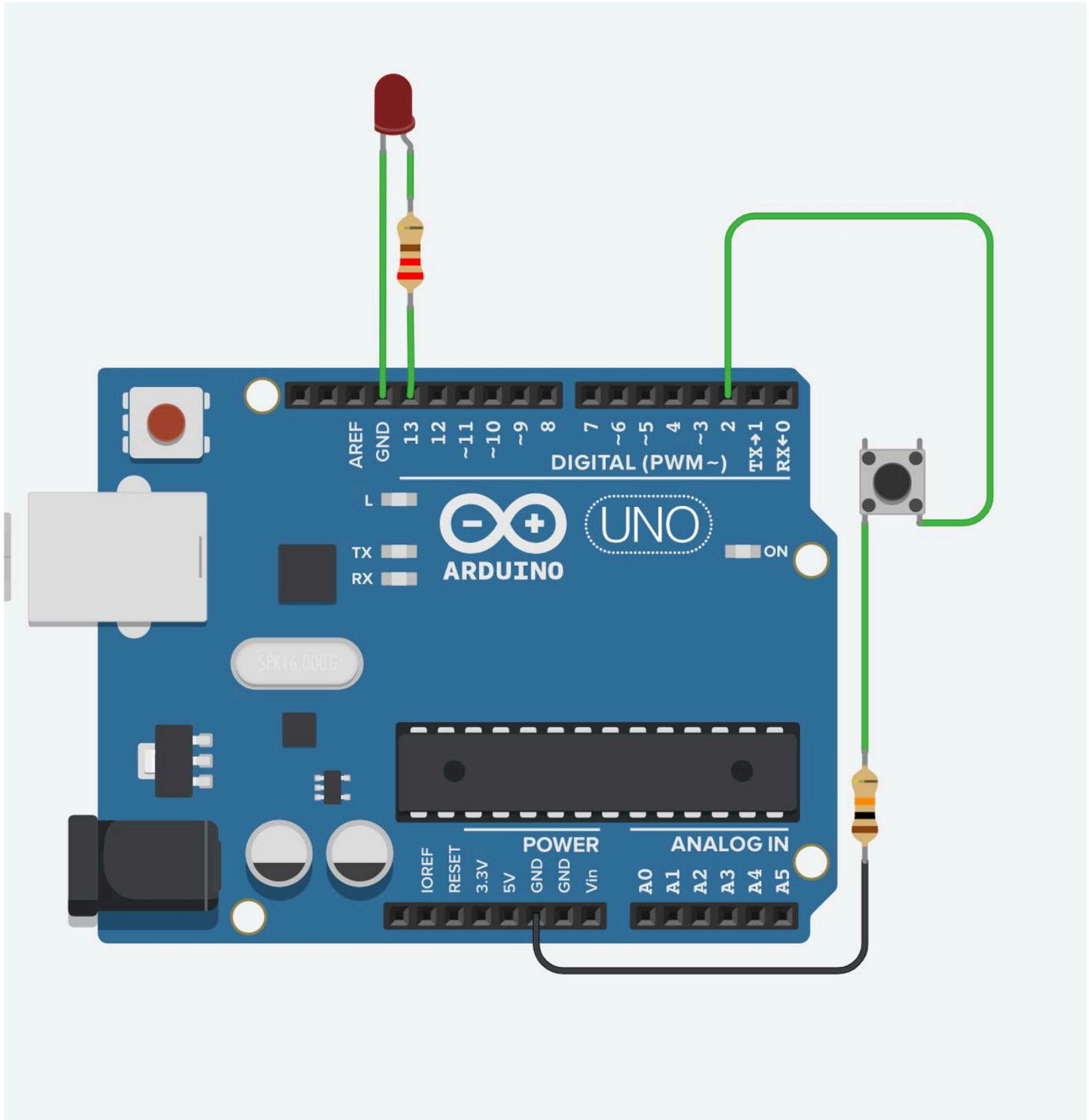
Assignment

- **Prepare your design for the following exercises:**
  - 6-A) A LCD signage (page 29 on October 18)
  - 6-B) LCD version of a temperature sensing system (page 25 on October 25)
- **Put your designs into TinkerCAD, and name them "DESIGN-6A(*student id*)" and "DESIGN-6B(*student id*)," respectively.**
- **Submit your entry on ScombZ (instructions will be ready tomorrow). Your designs on TinkerCAD will be evaluated.**
- **Use your nickname to log in**
- **Submission due is 23:59 on November 14 (JST)**

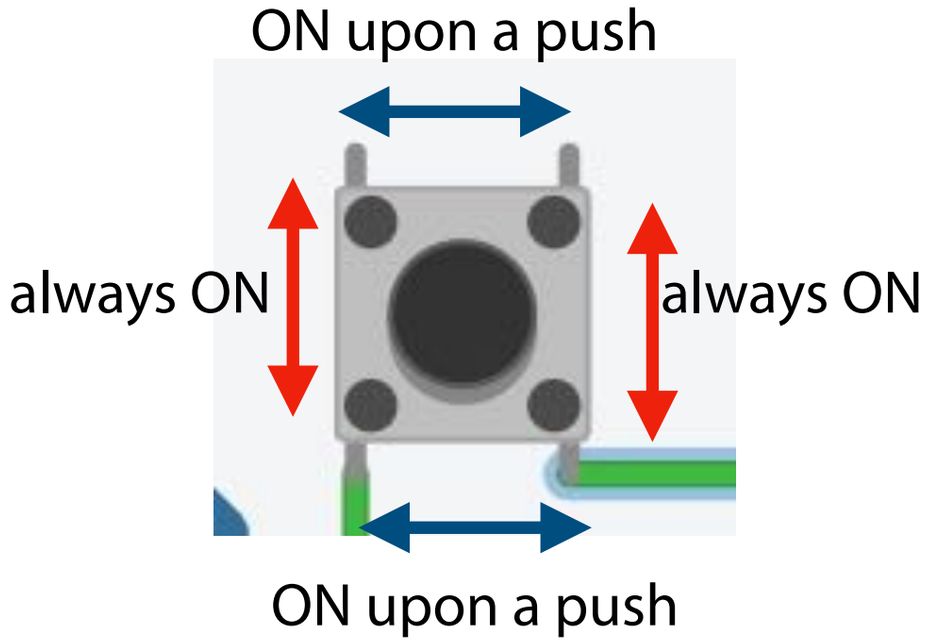
Today's Projects

- a) Push button blinker on page 21 of this slides.
- b) Blinker with 0.5s interval by timer interrupts.
- c) Temperature sensor and 1.5s interval blinker by using timer interrupts.

(a)

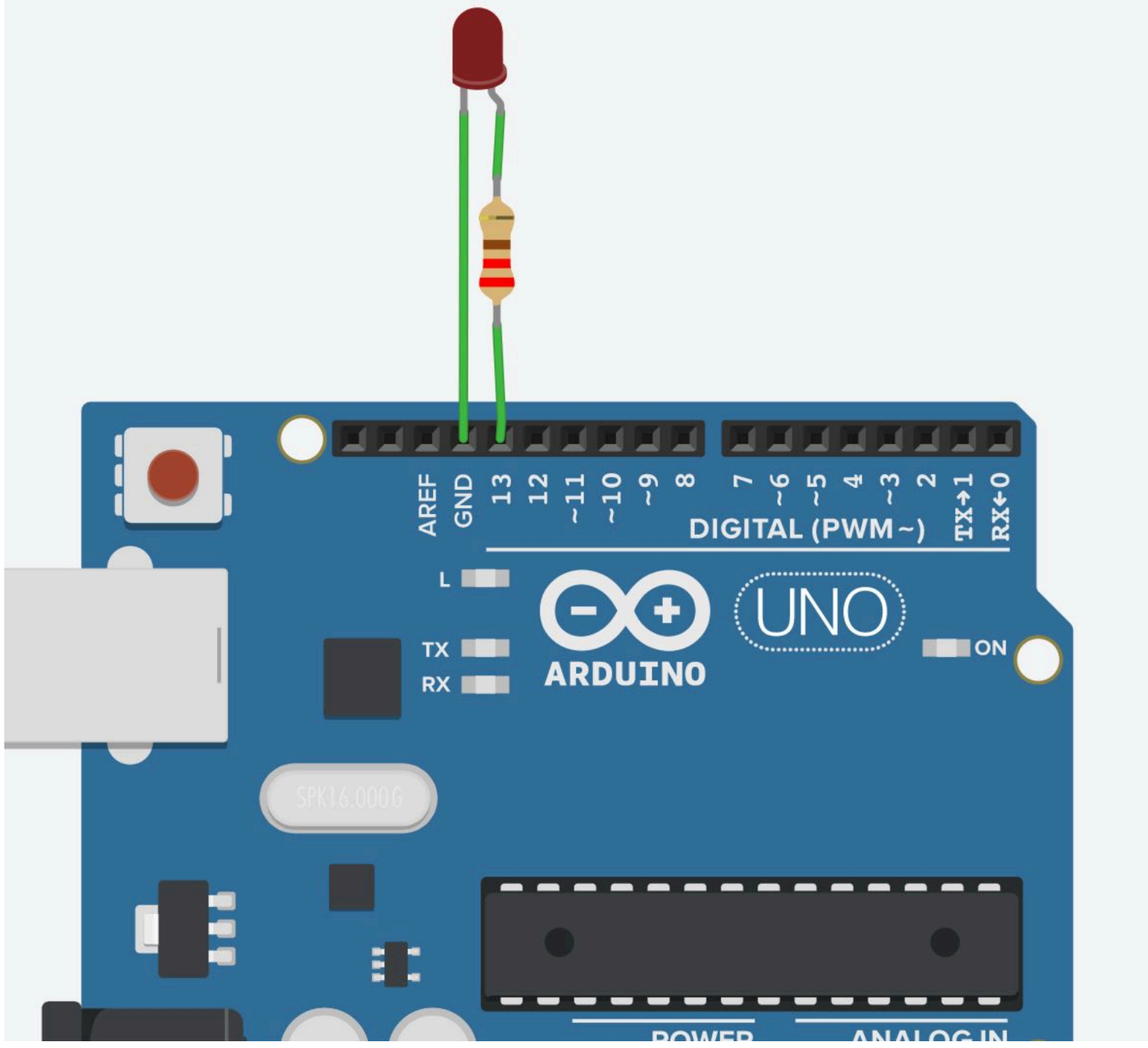


- A simple system which realizes that pushing the button makes the LED turn on and off.

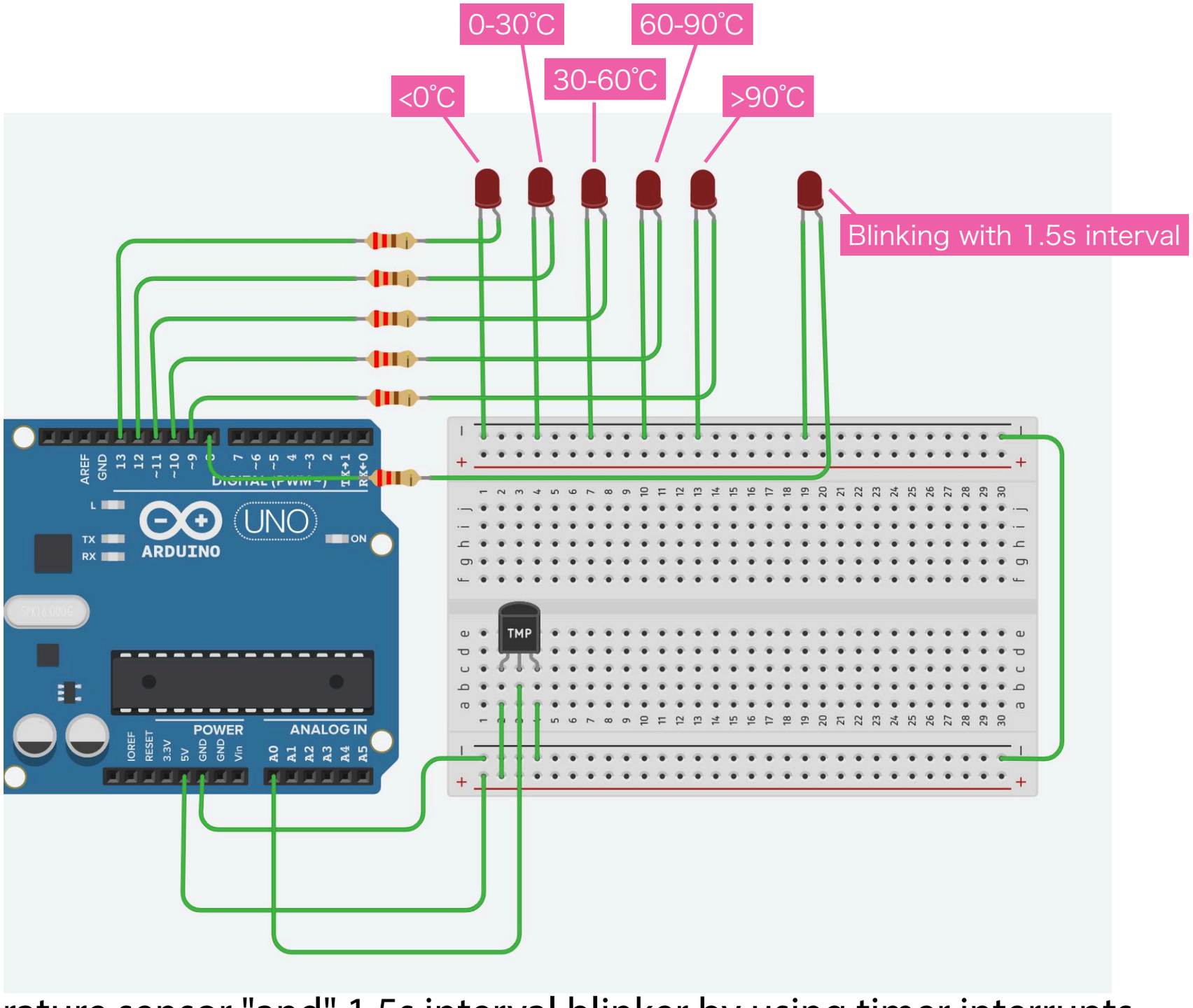


(b)

Blinker with 0.5s interval by timer interrupts.



(c)



Temperature sensor "and" 1.5s interval blinker by using timer interrupts.

(c) in Single Thread Model

```
/* Temperature sensor with 5 LEDs */

int pos = 9;
int s = 0;
float temp;

void setup()
{
  int i;

  for (i = 8; i <= 13; i++)
    pinMode(i, OUTPUT);
  Serial.begin(9600);
}
```

```
void loop()
{
  int i;

  temp = (analogRead(A0) * 5.0 / 1024.0 - 0.5)
           * 100.0;
  Serial.println((int)temp);
  if ((int)temp < 0) {
    pos = 13;
  } else if ((int)temp < 30) {
    pos = 12;
  } else if ((int)temp < 60) {
    pos = 11;
  } else if ((int)temp < 90) {
    pos = 10;
  } else {
    pos = 9;
  }
  for (i = 9; i<= 13; i++) {
    digitalWrite(i, LOW);
  }
  digitalWrite(pos, HIGH);
  digitalWrite(8, (s++ % 2) ? HIGH : LOW);
  delay(1000);
}
```

Reading temperature

Set 5 LEDs

LED Blinker

Time for Your Project

- Feel free to discuss with your friends
- If you have a question, ask the teaching assistant or just speak up.

Conclusions

- Event-driven execution context is an essential tool for complex, asynchronous event handling.

- **Next week**:
  Communication between two processors (or more)
  Do not forget to submit your design before the deadline

- **Homework**:
  Finish your projects (not for evaluation)