Lab 5 – Dealing with Interrupts

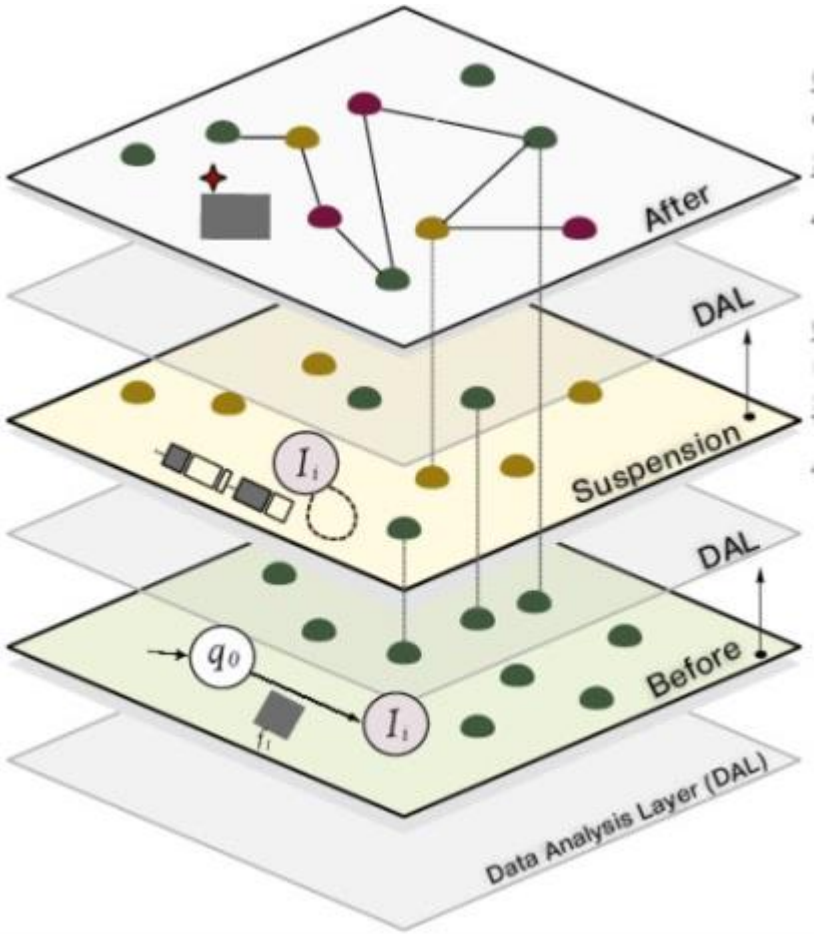
Anubhav Elhence

Lab 5 – Dealing with Interrupts

Anubhav Elhence

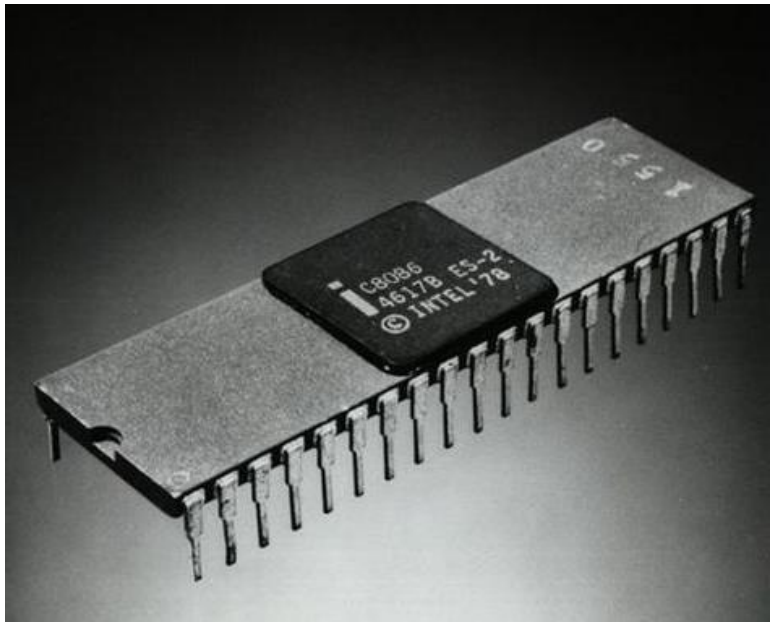
What are interrupts in MCU?

- Interrupts in 8086 architecture or any other architecture refer to signals that interrupt the normal program execution of the processor and cause it to temporarily stop executing its current instructions to handle a specific event or condition. These events can be generated by external devices such as keyboard, mouse, or timer, or by internal conditions such as errors or exceptions.



What are interrupts in MCU?

- The 8086 microprocessor has two types of interrupts: hardware interrupts and software interrupts.
- Hardware interrupts are generated by external devices connected to the system, such as keyboard, mouse, or disk controller, to request the attention of the processor. The 8086 processor has 256 interrupt lines, numbered from 0 to 255, and each line is associated with a specific device. When an external device needs to send an interrupt to the processor, it sends a signal on its corresponding interrupt line. The processor then stops its current operation, saves its current state, and jumps to a specific interrupt handler routine to handle the interrupt.
- Software interrupts, also known as system calls, are generated by software programs to request the operating system to perform a specific operation or service. Software interrupts are triggered by executing a specific software instruction called an "INT" instruction. When the processor executes an INT instruction, it stops its current operation, saves its current state, and jumps to a specific interrupt handler routine defined by the operating system.

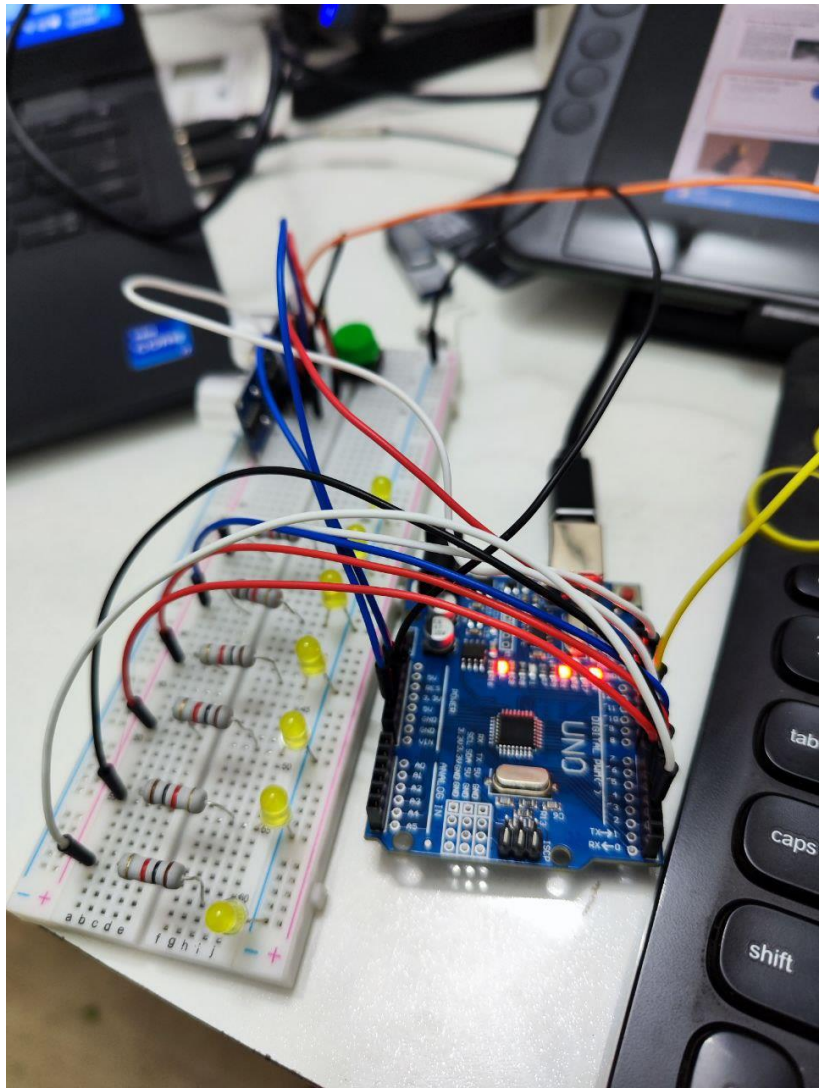


What are interrupts in MCU?

- In summary, interrupts in 8086 architecture allow the processor to handle external events and internal conditions in a timely and efficient manner, without requiring the program to constantly check for them.

- Let's Look at a practical example of why interrupts are important.

What are interrupts in MCU?



- In summary, interrupts in 8086 architecture allow the processor to handle external events and internal conditions in a timely and efficient manner, without requiring the program to constantly check for them.
- Let's Look at a practical example of why interrupts are important.

Arduino Code without interrupt

led distance indicator | Arduino IDE 2.0.2

File Edit Sketch Tools Help

Arduino Uno

led distance indicator.ino

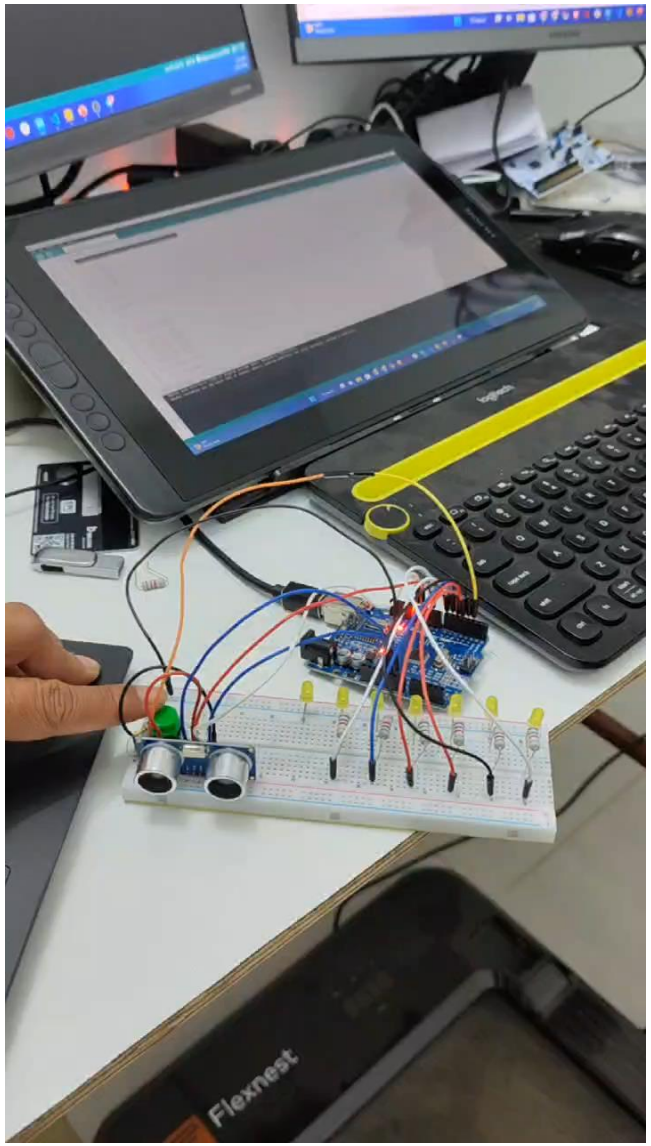
```
const int trig = 11;
const int echo = 12;

const int LED1 = 9;
const int LED2 = 3;
const int LED3 = 4;
const int LED4 = 5;
const int LED5 = 6;
const int LED6 = 7;
const int LED7 = 8;

int duration = 0;
int distance = 0;

int button=2;

int a;

void setup()
{
  pinMode(trig , OUTPUT);
  pinMode(echo , INPUT);

  pinMode(LED1 , OUTPUT);
  pinMode(LED2 , OUTPUT);
  pinMode(LED3 , OUTPUT);
  pinMode(LED4 , OUTPUT);
  pinMode(LED5 , OUTPUT);
  pinMode(LED6 , OUTPUT);
  pinMode(LED7 , OUTPUT);

  pinMode(button, INPUT);

  Serial.begin(9600);
```

```
void loop()
{

  a = digitalRead(button);
  if (a){
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, HIGH);
    digitalWrite(LED4, HIGH);
    digitalWrite(LED5, HIGH);
    digitalWrite(LED6, HIGH);
    digitalWrite(LED7, HIGH);
    delay(1000);
  }
  digitalWrite(trig , HIGH);
  delayMicroseconds(1000);
  digitalWrite(trig , LOW);


  duration = pulseIn(echo , HIGH);
  distance = (duration/2) / 28.5 ;
  Serial.println(distance);
```

```
  if ( distance <= 17 )
  {
    digitalWrite(LED5, HIGH);
  }
  else
  {
    digitalWrite(LED5, LOW);
  }
  if ( distance <= 20 )
  {
    digitalWrite(LED6, HIGH);
  }
  else
  {
    digitalWrite(LED6, LOW);
  }
  if ( distance <= 25 )
  {
    digitalWrite(LED7, HIGH);
  }
  else
  {
    digitalWrite(LED7, LOW);
  }

  delay(1000);
}
```

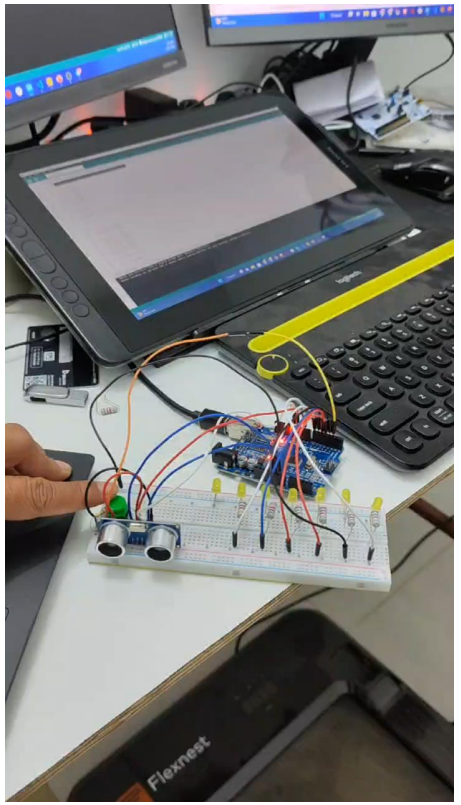

Anubhav Elhence and Dr. Vinay Chamola

Let's have a performance Check



Let's have a performance Check

- Clearly it is very laggy.
- The button press is an unexpected event, AND OUR CODE IS NOT ALWAYS READY TO handle the unexpected event.
- Therefor, it has to be modelled as interrupt.
- Let's do that.



Button press captured as an Interrupt

```
const int trig = 11;
const int echo = 12;

const int LED1 = 9;
const int LED2 = 3;
const int LED3 = 4;
const int LED4 = 5;
const int LED5 = 6;
const int LED6 = 7;
const int LED7 = 8;

int duration = 0;
int distance = 0;

int buttonPin=2;

int a;
volatile boolean buttonState = false;
```

```
void setup()
{
  pinMode(trig , OUTPUT);
  pinMode(echo , INPUT);

  pinMode(LED1 , OUTPUT);
  pinMode(LED2 , OUTPUT);
  pinMode(LED3 , OUTPUT);
  pinMode(LED4 , OUTPUT);
  pinMode(LED5 , OUTPUT);
  pinMode(LED6 , OUTPUT);
  pinMode(LED7 , OUTPUT);

  pinMode(buttonPin, INPUT);
  attachInterrupt(digitalPinToInterrupt(buttonPin), blink1, FALLING);


  Serial.begin(9600);

}
```

Button press captured as an Interrupt

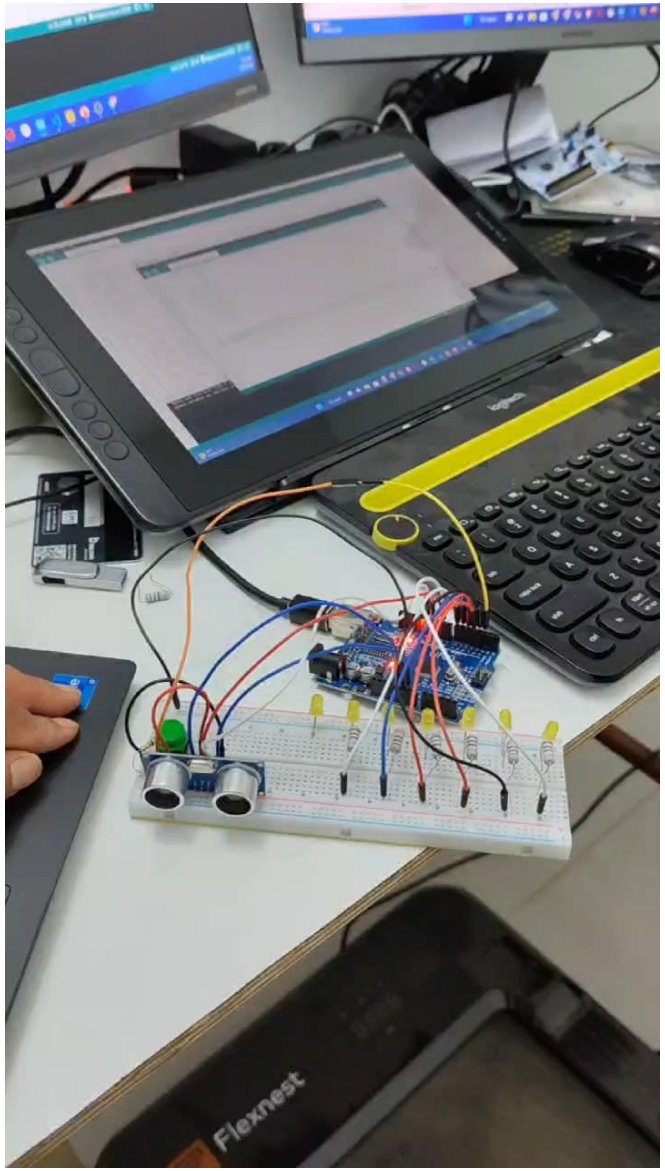
```
const int trig = 11;
const int echo = 12;

const int LED1 = 9;
const int LED2 = 3;
const int LED3 = 4;
const int LED4 = 5;
const int LED5 = 6;
const int LED6 = 7;
const int LED7 = 8;

int duration = 0;
int distance = 0;

int buttonPin=2;

int a;
volatile boolean buttonState = false;
```

```
void setup()
{
  pinMode(trig , OUTPUT);
  pinMode(echo , INPUT);

  pinMode(LED1 , OUTPUT);
  pinMode(LED2 , OUTPUT);
  pinMode(LED3 , OUTPUT);
  pinMode(LED4 , OUTPUT);
  pinMode(LED5 , OUTPUT);
  pinMode(LED6 , OUTPUT);
  pinMode(LED7 , OUTPUT);

  pinMode(buttonPin, INPUT);
  attachInterrupt(digitalPinToIn

  Serial.begin(9600);

}
```

```
void loop()
{

 while (buttonState){
   digitalWrite(LED1, HIGH);
   digitalWrite(LED2, HIGH);
   digitalWrite(LED3, HIGH);
   digitalWrite(LED4, HIGH);
   digitalWrite(LED5, HIGH);
   digitalWrite(LED6, HIGH);
   digitalWrite(LED7, HIGH);
   Serial.println("INSIDE BUTTONState");
 }


 digitalWrite(trig , HIGH);
 delayMicroseconds(1000);
 digitalWrite(trig , LOW);


 duration = pulseIn(echo , HIGH);
 distance = (duration/2) / 28.5 ;
 Serial.println(distance);


 if ( distance <= 5 )
 {
   digitalWrite(LED1, HIGH);
 }
 else
 {
```

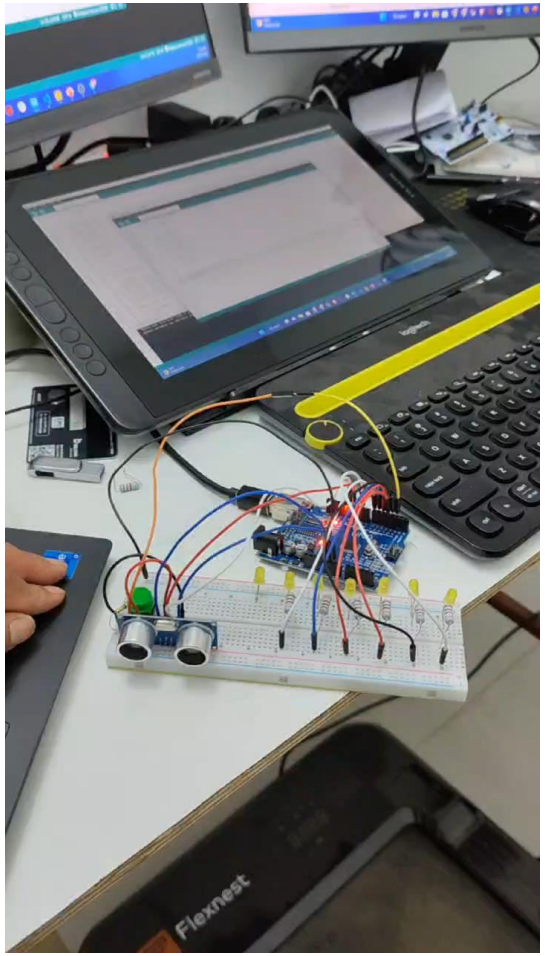
```
  }
  else
  {
    digitalWrite(LED7, LOW);
  }

}

void blink1()
{
  Serial.println("HIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII");
  Serial.println("HIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII");
  Serial.println("HIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII");
  Serial.println("HIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII");
  Serial.println("HIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII");
  Serial.println("HIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII");
  buttonState = !buttonState;

}
```

Button press captured as an Interrupt



What did you observe now?

- Faster Response Time: Interrupts allow the microcontroller to respond to external events almost instantaneously, as soon as they occur. This is because the microcontroller can suspend the current task and immediately jump to the interrupt service routine (ISR) to handle the event. In contrast, polling requires the microcontroller to continuously check for the event, which can result in longer response times.
- Reduced CPU Load: Interrupts can significantly reduce the CPU load by allowing the microcontroller to focus on other tasks when there are no events to handle. In contrast, polling requires the CPU to continuously check for events, which can waste CPU cycles and slow down the system.
- Simpler Code: Interrupts can simplify the code by allowing the programmer to write more modular and reusable code. The code for handling the event can be separated into an ISR, which is called only when needed, while the main program can focus on other tasks. This can make the code more readable, maintainable, and easier to debug.
- Precise Timing: Interrupts can be used to achieve precise timing in real-time applications. For example, an interrupt can be used to generate a periodic pulse or to measure the duration of a signal. This level of precision is difficult to achieve with polling.
- Energy Efficiency: Interrupts can improve energy efficiency by allowing the microcontroller to enter a low-power sleep mode when there are no events to handle. This can significantly reduce the power consumption of the system, especially in battery-powered devices.



Now Let's Understand Interrupts Theoretically with Dr. Vinay Chamola

Follow Along Example:

- Testing the prompts really quick
- **Input a character from the keyboard (STDIN) with Echo**
- **Input a character from the keyboard (STDIN) without Echo**

Follow Along Example:

- Testing the prompts really quick
- **Input a character from the keyboard (STDIN) with Echo**

```
MOV AH, 01h                    ; AH -01 parameter for INT 21h
INT 21h
```

- **Input a character from the keyboard (STDIN) without Echo**

```
MOV AH, 08h                    ; AH -08 parameter for INT 21h
INT 21h
```

- **Input a string from keyboard (STDIN)**

Input a string from keyboard (STDIN)

week5_c1.asm > ...

```
1    .model tiny
2    .486
3    .data
     1 reference
4    max1 db 32
     0 references
5    act1 db ?
     0 references
6    inp1 db 32 dup(0)
7    .code
8    .startup
9
10       lea DX, max1
11       mov ah, 0ah
12       int 21h
13
14   .exit
     0 references
15   end
```

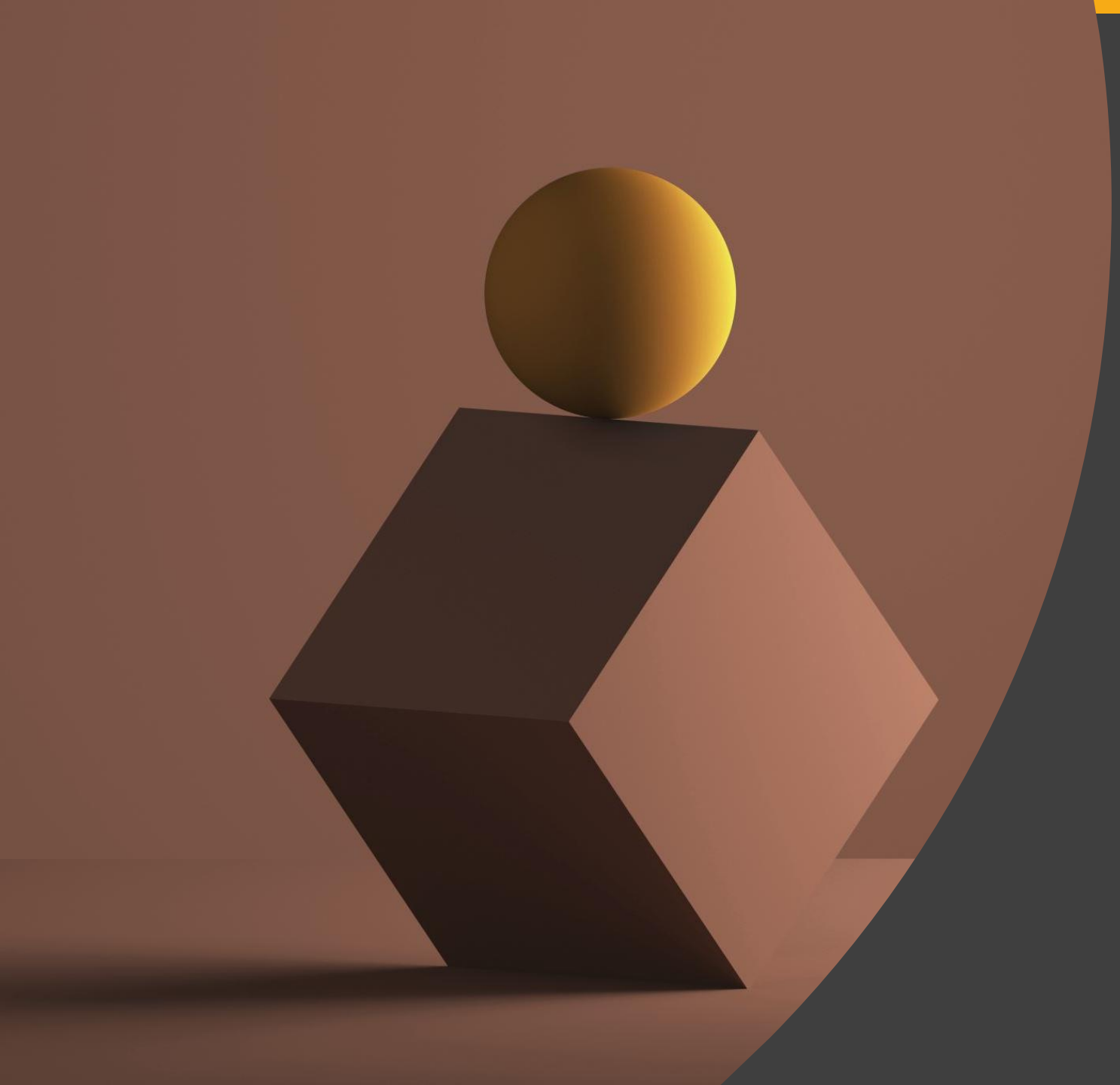
- After the interrupt, act 1 will contain the number of characters read, and the characters themselves will start at inp1 The characters will be terminated by a carriage return (ASCII code 0Dh), although this will not be included in the count (Note: this will not be included in the ACT1 but you have to count Enter also when you are specifying it in max1)

- **Output a string from keyboard (STDIN)**

Output a string from keyboard (STDIN)

```
.data
str1 db 'HELLO$'                    ; all strings must terminate with '$' ASCII value (24h)

.code
.startup

LEA DX, str1
MOV AH, 09h
INT 21h
```

- When interrupt is executed the string "HELLO" will be displayed on screen. Remove the '$' sign. What happens



Thankyou

Anubhav Elhence and Dr. Vinay Chamola