

# Módulo 3:

## Hardware y Software



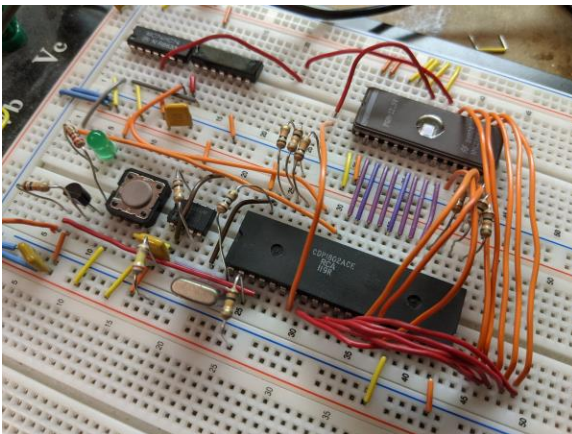
# Agenda

- Componentes Hardware
- Componentes Software
- Sistemas Operativos (OS)

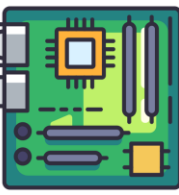


# Combinación de HW y SW

- Para diseñar un dispositivo IoT
  - Diseño HW
  - Diseño SW
  - Diseño para que HW y SW trabajen juntos
- Uso de Fichas técnicas
  - Dimensiones, pines I/O, parámetros electricos
  - Ejemplo: rectificador [KBPC3506](#)



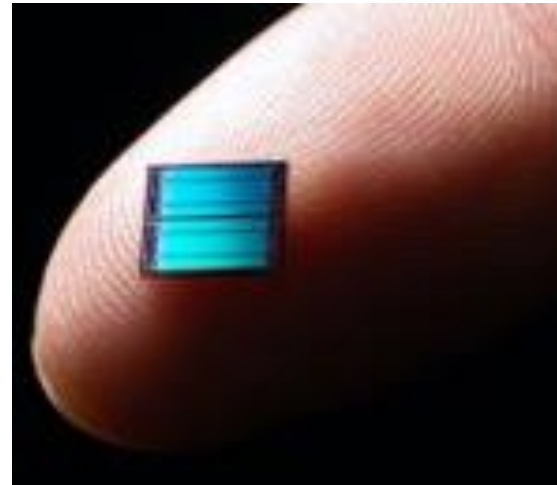
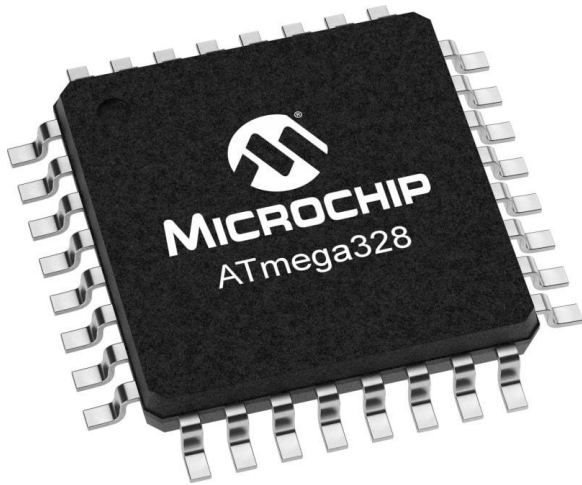
```
void setup() {  
    // initialize digital pin LED_BUILTIN as an output  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the positive voltage)  
    delay(1000); // wait for a second  
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off (LOW is the negative voltage)  
    delay(1000); // wait for a second  
}
```

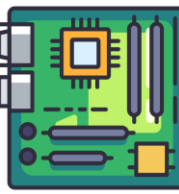


# Componentes Hardware

## Circuitos integrados

- Fabricados de material semiconductor (silicona)
- Protegidos por un paquete con un set de pines para acceder al chip
- Fichas técnicas complejas (+100 paginas)



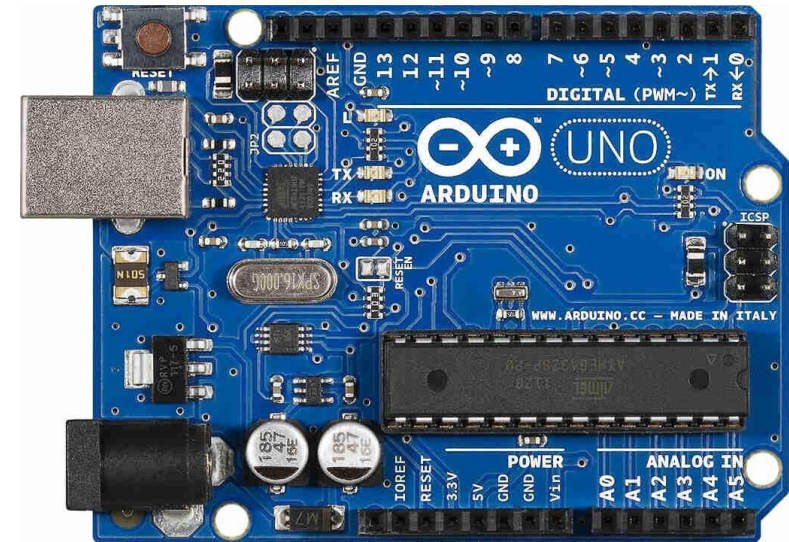


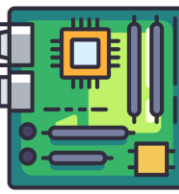
# Componentes Hardware

## Microcontroladores: Características

Pregunta: ¿Qué microcontrolador debemos escoger?

- Ancho de banda del datapath:
  - Numero de bits en un registro --> más bits más precisión
  - Arduino --> 8-bits
- Numero de pines I/O
- Rendimiento
  - Frecuencia del clock más lenta que un computador tradicional
  - Velocidad del procesador no siempre importante
- Temporizadores (Timers)
  - Para aplicaciones en tiempo real
  - La precisión es importante



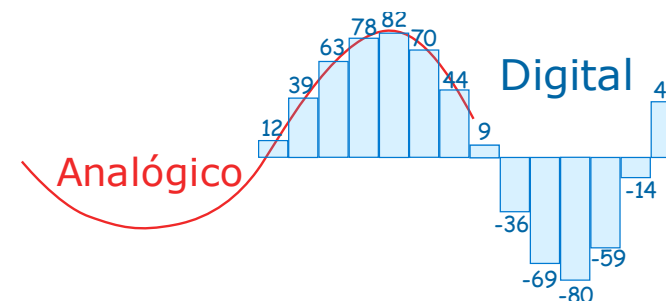
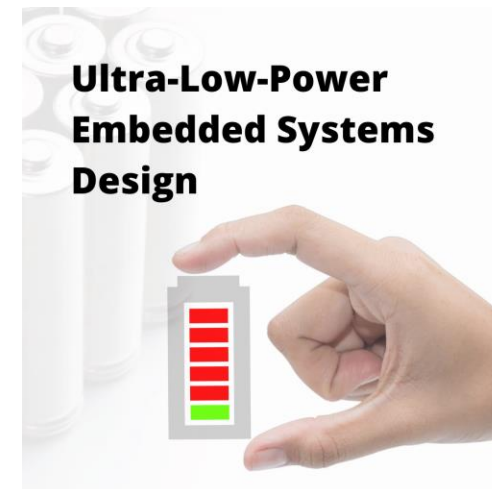


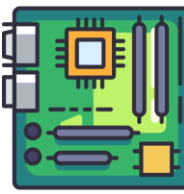
# Componentes Hardware

## Microcontroladores: Propiedades

Pregunta: ¿Qué microcontrolador debemos escoger?

- ADC
  - Convierten señales análogas a digitales
- Modos de alimentación
  - Ahorro energético es clave
- Soporte de protocolos de comunicación
  - Interface con otros circuitos integrados I2C, UART, SPI etc

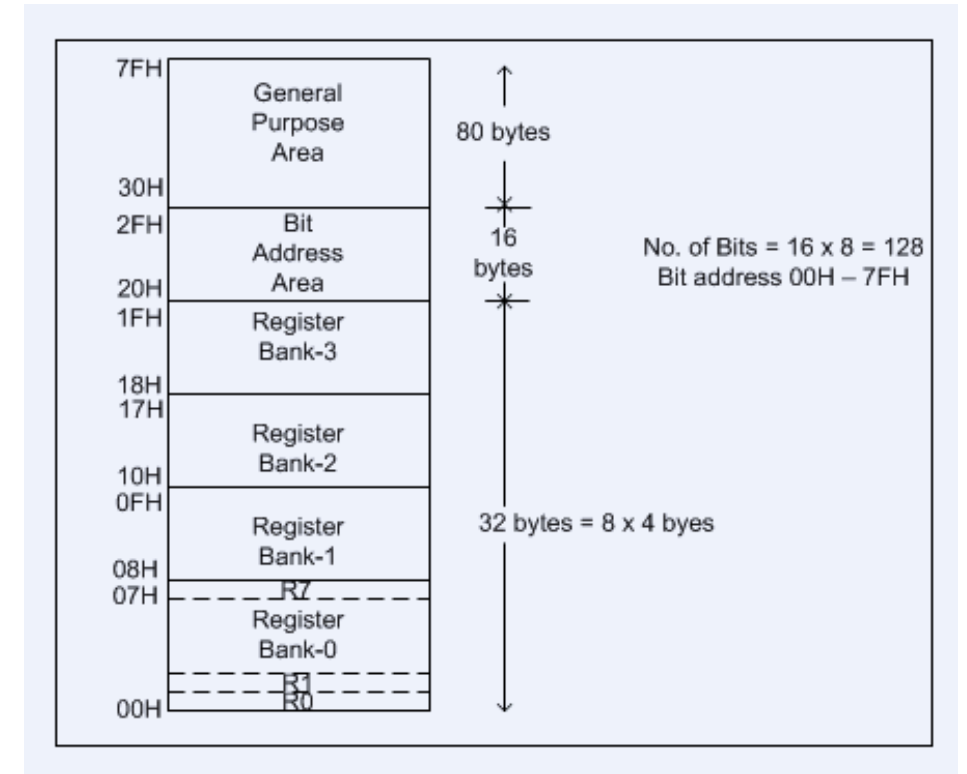




# Componentes Software

## Almacenamiento de datos en Microcontroladores

- Registros
  - Elemento más básico de almacenamiento
  - Espacio único en la memoria, guarda un solo valor
  - Son muy rápidos (tiempo de acceso menor a un clock cycle),
  - Cuestan mucho procesamiento
- Registros de propósito especial
  - Usados internamente por el microcontrolador
- Registros de propósito general
  - Registros que el programa puede usar
- Archivo de registros
  - Grupo de registros, cada uno con una dirección, actúan como una memoria
  - Solo se puede acceder uno o dos registros al mismo tiempo
  - Son muy rápidos (tiempo de acceso menor a un clock cycle), pero cuestan mucho procesamiento
  - Operadores de instrucciones (egg. ADD, registros de destinacion)
  - Usualmente de 32 registros







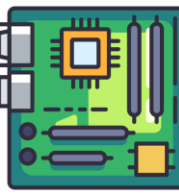
# Componentes Software

## Almacenamiento de datos en Microcontroladores

- Memoria cache
  - Mucho más grandes que un archivo de registros (1Mb)
  - Mas lentos que un archivo de registro (1 full clock cycle)
  - Cache de datos --> guarda los datos del programa
  - Cache de instrucciones --> guarda las instrucciones del programa (código)
- Memoria principal
  - Como un cache pero más grande, más lento y menor costo de procesado (Gb)
  - No está en el CPU (conectada vía system bus)
- Cuello de Botella Von Neumann
  - Memoria es más lenta que el CPU
  - Ejemplo: sumas dos variables (memoria 100 ciclos, ADD 1 ciclo)



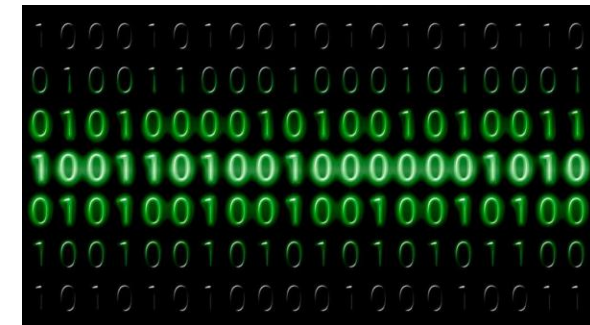




# Componentes Software

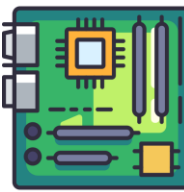
## Traducción del SW

- Lenguaje de Maquina (Machine Language)
  - Conjunto de instrucciones simples en binario (1 y 0)
  - Microcontrolador no entiende directamente C, C++, JAVA
  - Ejemplo: procesador Intel entiende lenguaje de maquina Intel x86
  - No se puede leer (facilmente)
- Lenguaje Ensamblador (Assembly Language)
  - Se puede leer (entender)
  - Instrucciones basicas como ADD, R1, R2
  - No insturcciones complejas (e.g. for loops)
  - Una instrucción de ensamblador es una instrucción de maquina (mapeados)
- Lenguajes de Alto Nivel
  - C/C++, JAVA, etc.
  - Facil de usar, leer y entender



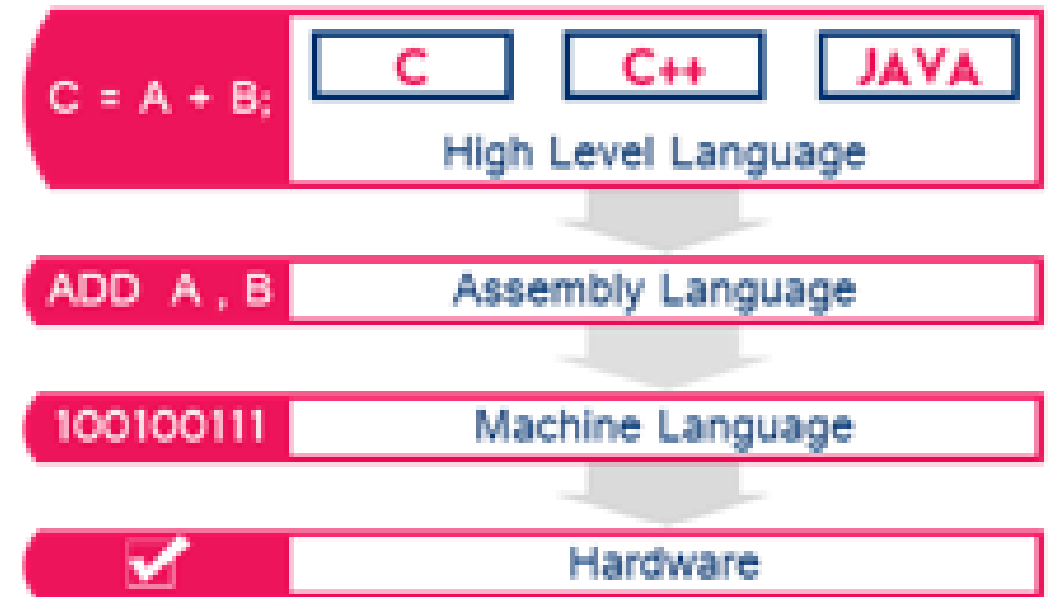
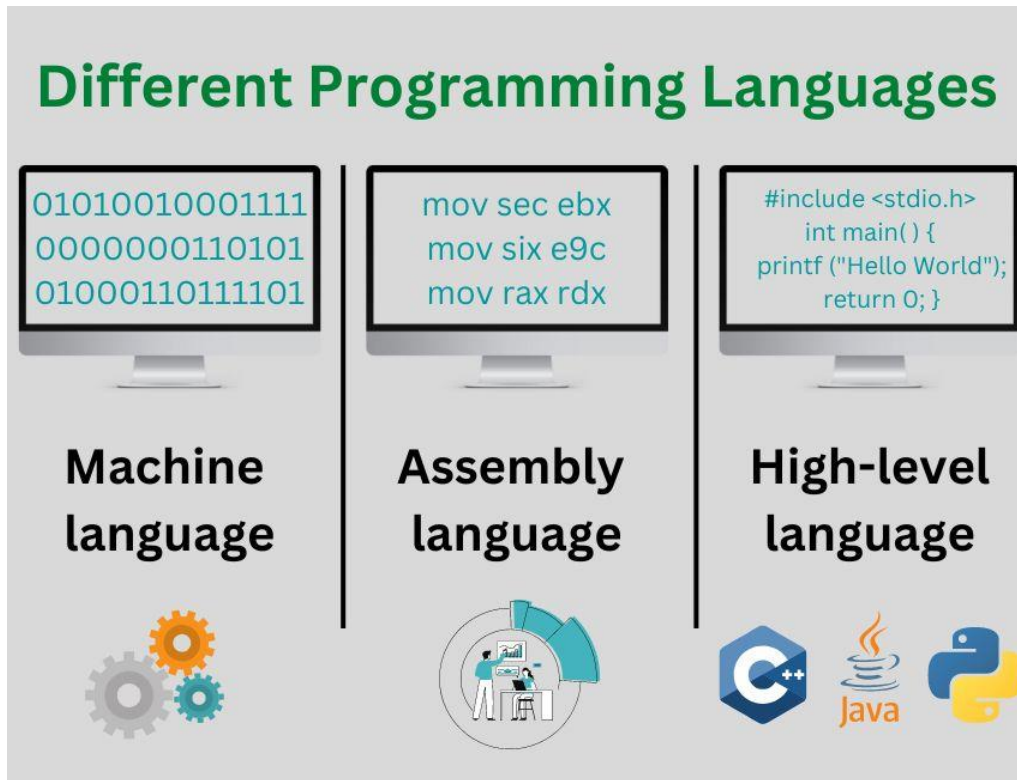
```
1 section .text
2     global _start
3
4 _start:
5     mov     edx,len
6     mov     ecx,msg
7     mov     ebx,1
8     mov     eax,4
9     int     0x80
10
11     mov     eax,1
12     int     0x80
13
14 section .data
15 msg db 'Hola, mundo!', 0xa
16 len equ $ - msg
```

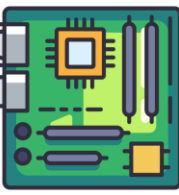




# Componentes Software

## Traducción del SW

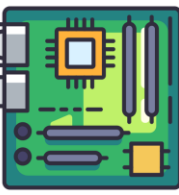




# Componentes Software

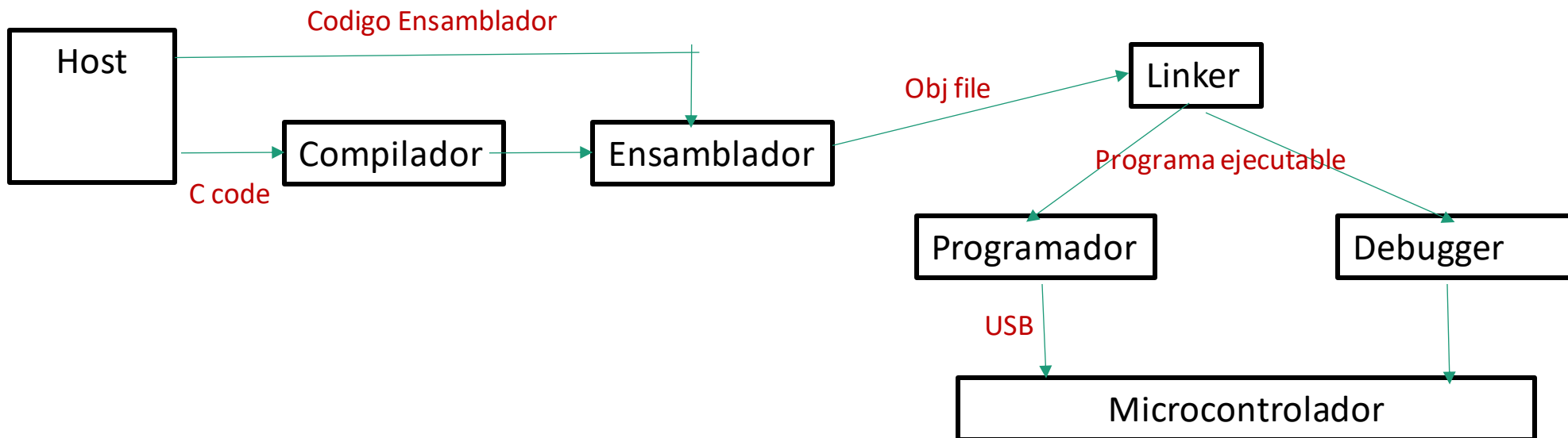
## Dos formas de traducción: Compilación vs Interpretación

- Compilación (C/C++) (Arduino)
  - Traduce las instrucciones una vez antes de correr el código (ahorra tiempo)
  - El resultado es un ejecutable (.exe) --> lenguaje de maquina
    - No hay necesidad de compilar de nuevo
  - El programador se encarga de los detalles
- Interpretación (Python)
  - Traducción ocurre en cada ejecución (lento)
  - Mas amigable con el programador
  - El interpretador no es perfecto
- Ejemplo
  - En C se debe declarar variables para que el compilador entienda que tipo de variable está usando, en Python no se necesita, el Interpretador se encarga de averiguar a que tipo de variable se refiere el programador



# Componentes Software

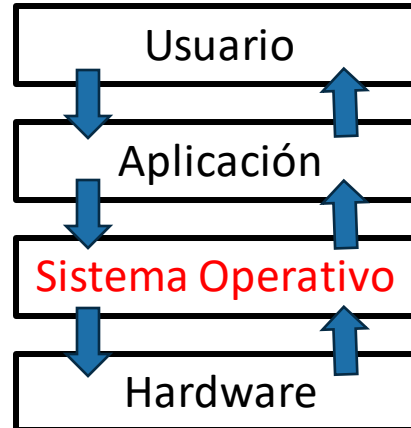
Cadena de herramientas SW (toolchain)



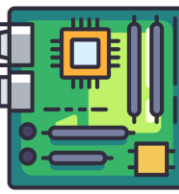


# Sistemas Operativos

- Es una capa extra entre el programa y el HW



- No siempre presentes en los dispositivos IoT.
  - Arduino no tiene OS



# Sistemas Operativos

- Se encarga de los detalles de interactuar con el hardware, maneja otros programas
- Permite correr muchas aplicaciones al mismo tiempo
  - En realidad, no están corriendo al mismo tiempo, están alternando muy rápido
- En su mayoría tienen una interface amigable
  - Ejemplos: Windows, iOS
- Los OS del IoT no tienen interfaz amigable
  - Linea de comandos





# Sistemas Operativos

- Ejemplo:
  - Video streaming
  - Joystick
  - Detección de obstáculos
  - Frenado automático

