



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA
DEPARTAMENTO DE INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE MICROCOMPUTADORAS

Manual de recursos y aplicaciones
PLATAFORMA RASPBERRY Pi

RUBÉN ANAYA GARCÍA
MOISES MELENDEZ REYES
ISRAEL RIVERA
ALBERTO TEMPLOS CARBAJAL

CIUDAD UNIVERSITARIA
PE108223

Material generado en colaboración con los docentes de la academia de la asignatura de Microcomputadoras, integrado por profesores de la Teoría y Laboratorio; así como de profesores participantes.

RUBÉN ANAYA GARCÍA
MOISES MELENDEZ REYES
ANTONIO SALVA CALLEJA
JOSE ANTONIO ARREDONDO GARZA
ANGELICA QUIÑONES JUAREZ
DIANA CRUZ HERNANDEZ
LUIS SERGIO DURÁN ARENAS
ROMÁN OSORIO COMPARAN
ALBERTO TEMPLOS CARBAJAL

Este material y la infraestructura generada fueron obtenidos a través de apoyo de **DGAPA** al proyecto **PAPIME PE108223**
*“Prácticas de Laboratorio de microcomputadoras
basadas en las plataformas Raspberry Pi”*

Introducción

Este manual proporciona información útil en la implementación del proyecto PE108223: “Prácticas de Laboratorio de Microcomputadoras: Basadas en las Plataformas Raspberry Pi”, en el que se presentan los procedimientos e información complementaria para la realización de las actividades solicitadas.

Contenido:

- 1. Plataformas Raspberry Pi.*
 - 2. Descarga, instalación de Raspberry SO.*
 - 3. Conexión remota.*
 - 4. Ensamblador y GDB.*
 - 5. IDE Code::Blocks.*
 - 6. Simulador en línea CPUlator.*
 - 7. Descarga e instalación de Thonny.*
 - 8. Programación de APP inventor; comunicación Bluetooth.*
 - 9. Control remoto de giro de un motor a pasos.*
 - 10. Control manual de posición angular de un motor a pasos.*
 - 11. Giroscopio y acelerómetro.*
 - 12. Control de un motor de CD.*
-

1

Raspberry Pi

Plataformas

La plataforma Raspberry ha sido diseñada por la Fundación Raspberry; con el paso del tiempo ha diseñado distintas versiones:

1. Raspberry Pi

Plataforma	SoC	Memoria	E/S	Conectividad
Raspberry Pi modelo B, B+	BCM2835	256 MB 512 MB	26	HDMI, 2 USB 2.0, audio, video RCA, Ethernet RJ45, SD Card, Alimentación micro USB
Raspberry Pi modelo A, A+	BCM2835	256 MB	26	HDMI, 2 USB 2.0, audio, video RCA, SD Card, Alimentación micro USB
Raspberry Pi 2 Modelo B	BCM2836, BCM2837	1 GB	40	HDMI, 4 USB 2.0, audio, video RCA, Ethernet RJ45, SD Card, Alimentación micro USB
Raspberry Pi 3 modelo B, B+	BCM2837 BCM2837b0	1 GB	40	HDMI, 2 USB 2.0, audio, video RCA, Ethernet RJ45, WiFi 2.4 GHZ, Bluetooth 4.1, Bluetooth BLE, SD Card, Alimentación micro USB
Raspberry Pi 4 modelo B	BCM 2711	1GB, 2GB, 4GB y 8GB	40	2 micro HDMI, 2 USB 2.0, 2 USB 3.0, audio, video RCA, Ethernet RJ45, WiFi 2.4/5 GHZ (120 Mb/s) SD Card, Alimentación micro USB
Raspberry Pi 4	BCM 2712	2GB, 4 GB, 8 GB Y 16 GB	40	2 micro HDMI, 2 USB 2.0, 2 USB 3.0, audio, video RCA, Ethernet RJ45, WiFi 2.4/5 GHZ (300 Mb/s) SD Card, Alimentación tipo C

Tabla 1. Características generales de las versiones microcomputadoras



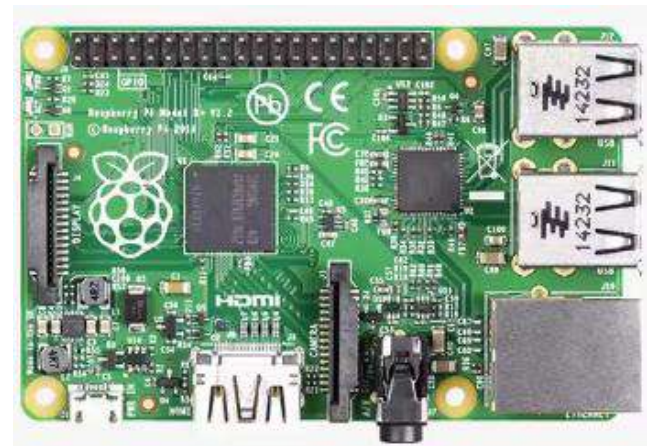
Raspberry Pi 5



Raspberry Pi 4



Raspberry Pi 3+



Raspberry Pi 1

Figura 1. Imágenes de las versiones de Raspberry Pi

2. Versiones Raspberry + Teclado

Plataforma	SoC	Memoria	E/S	Conectividad
Raspberry Pi 400	BCM 2711	4GB	40	2 micro HDMI, 2 USB 2.0, 2 USB 3.0, audio, video RCA, Ethernet RJ45, WiFi 2.4/5 GHZ (120 Mb/s) SD Card, Alimentación micro USB
Raspberry Pi 500	BCM 2712	8GB	40	2 micro HDMI, 2 USB 2.0, 2 USB 3.0, audio, video RCA, Ethernet RJ45, WiFi 2.4/5 GHZ (300 Mb/s) SD Card, Alimentación tipo C

Tabla 2. Versiones integradas



Raspberry Pi 500.



Raspberry Pi 400.

Figura 2. Imágenes de Raspberry integradas

3. Versiones reducidas (Zero).

Plataforma	SoC	Memoria	E/S	Conectividad
Raspberry Pi Zero	BCM2835	512 MB	40	Puerto mini HDMI, 2 puertos micro USB, slot tarjetas micro SD
Raspberry Pi Zero W	BCM2835	512 MB	40	Puerto mini HDMI, 2 puertos micro USB, slot tarjetas micro SD, WiFi 2.4 GHz (35 mb/s), Bluetooth 4.0, Bluetooth BLE
Raspberry Pi Zero 2W	RP3A0	512 MB	40	Puerto mini HDMI, 2 puertos micro USB, slot tarjetas micro SD, WiFi 2.4 GHz (35 mb/s), Bluetooth 4.2, Bluetooth BLE

Tabla 3. Características Raspberry Pi Zero



Raspberry Pi Zero



Raspberry Pi Zero W



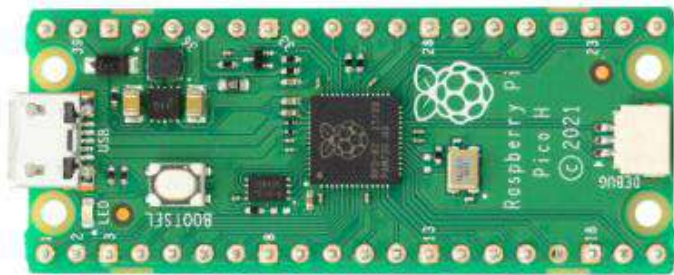
Raspberry Pi Zero 2W

Figura 3. Imágenes Raspberry Pi Zero

4. Versiones microcontrolador

Plataforma	SoC	Memoria	ROM	E/S	Conectividad
Raspberry Pi Pico	RP2040	264 MB	2 MB	40/26	-
Raspberry Pi Pico W	RP2040	264 MB	2 MB	40/26	WiFi 2.4 GHz (10 Mb/s), Bluetooth 5.2, Bluetooth BLE
Raspberry Pi Pico 2	RP2350	520 MB	4 MB	40/26	-
Raspberry Pi Pico 2 W	RP2350	520 MB	4 MB	40/26	WiFi 2.4 GHz (10 Mb/s), Bluetooth 5.2, Bluetooth BLE

Tabla 4. Características Raspberry Pi Pico



Raspberry Pi Pico.



Raspberry Pi Pico W.



Raspberry Pi Pico 2



Raspberry Pi Pico 2W

Figura 4. Imágenes Raspberry Pi Pico

2

Sistema Operativo Raspberry Pi

Raspberry Pi OS

La instalación requiere de:

1. Descarga del SO.
 2. Instalación de imagen.
 3. Instalación del SO en la unidad micro SD.
-

1.Descarga del SO.

La imagen del sistema operativo se puede descargar del sitio oficial de Raspberry Pi, existe versiones para Windows, MacOS y Ubuntu:

<https://www.raspberrypi.com/software/>

Una vez seleccionado el sistema operativo de su computadora, iniciará el proceso de descarga.



Figura 1. Descarga de la imagen del SO

Una vez concluido el proceso de descarga, tendrá a disposición el archivo ejecutable.

imager_1.8.5.exe	14/03/2025 01:26 p. m.	Aplicación	19,794 KB
------------------	------------------------	------------	-----------

Figura 2. Imagen del SO

2. Instalación Raspberry Pi Imager.

*Al Ejecutar el programa .exe, será confirmada la acción; presionar **Install** y esperar a concluir la instalación.*



Figura 3. Instalación de la imagen



Figura 4. Instalación Raspberry Pi Imager, concluido

3. Instalación de Raspberry Pi OS.

El entorno de la imagen tendrá la siguiente presentación:



Figura 5. Entorno del instalador

El proceso de instalación consta de las siguientes etapas; estas se muestran en las figuras 6 a la 17.

- a) Selección de la versión de Raspberry.*
- b) Selección de la versión del SO.*
- c) Selección de la unidad donde se ubica la memoria micro SD.*
- d) Configuraciones durante el proceso de instalación del SO, entre los que se encuentra:*
- e) Credenciales de identificación (nombre y contraseña).*
- f) Red de conexión alámbrica o inalámbrica.*
- g) Habilitación de comunicación SSH.*
- h) Confirmación de configuraciones.*

i) Instalar en Raspberry Pi.

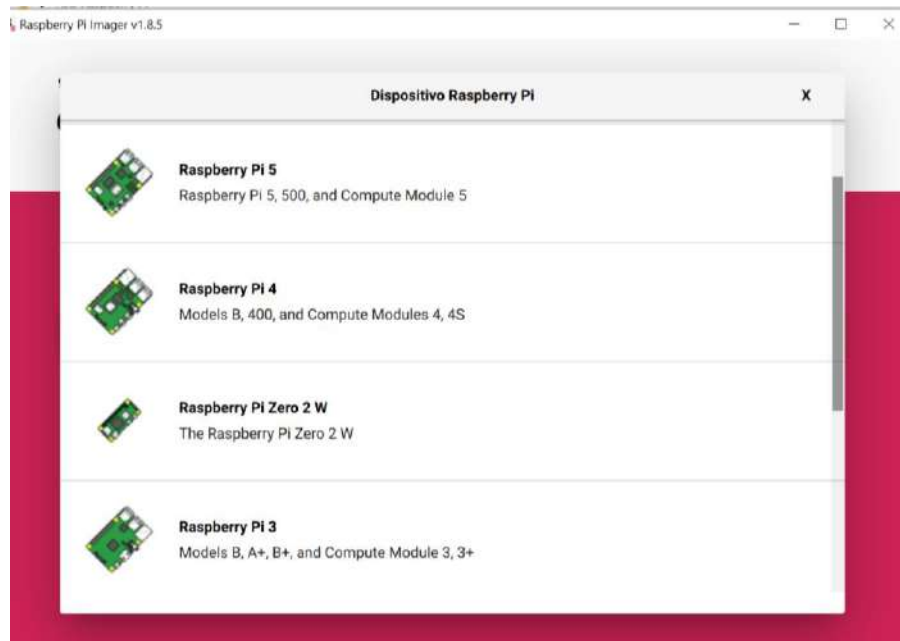


Figura 6. Selección versión Raspberry Pi

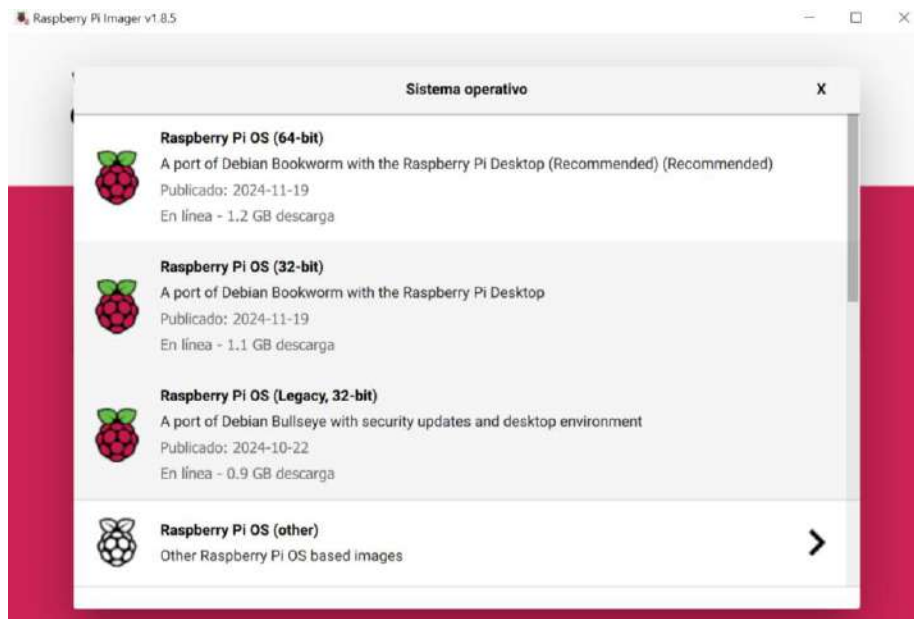


Figura 7. Selección del Sistema Operativo



Figura 8. Selección de la unidad micro SD

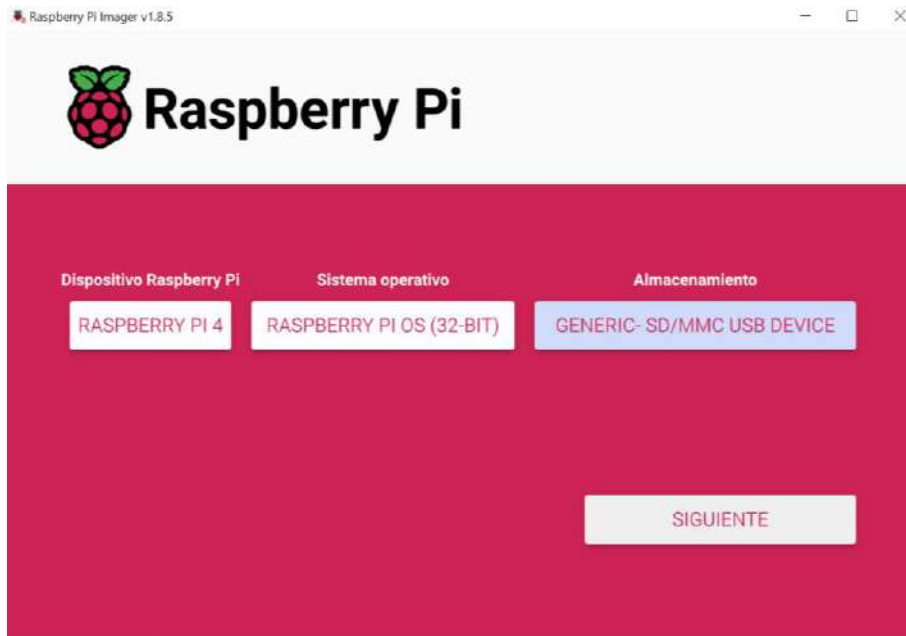


Figura 9. Configuraciones generales



Figura 10. Confirmación de personalización

Seleccionar editar ajustes para configurar claves de acceso a Raspberry, conexión inalámbrica, activar SSH y algunas configuraciones adicionales (figura 11).

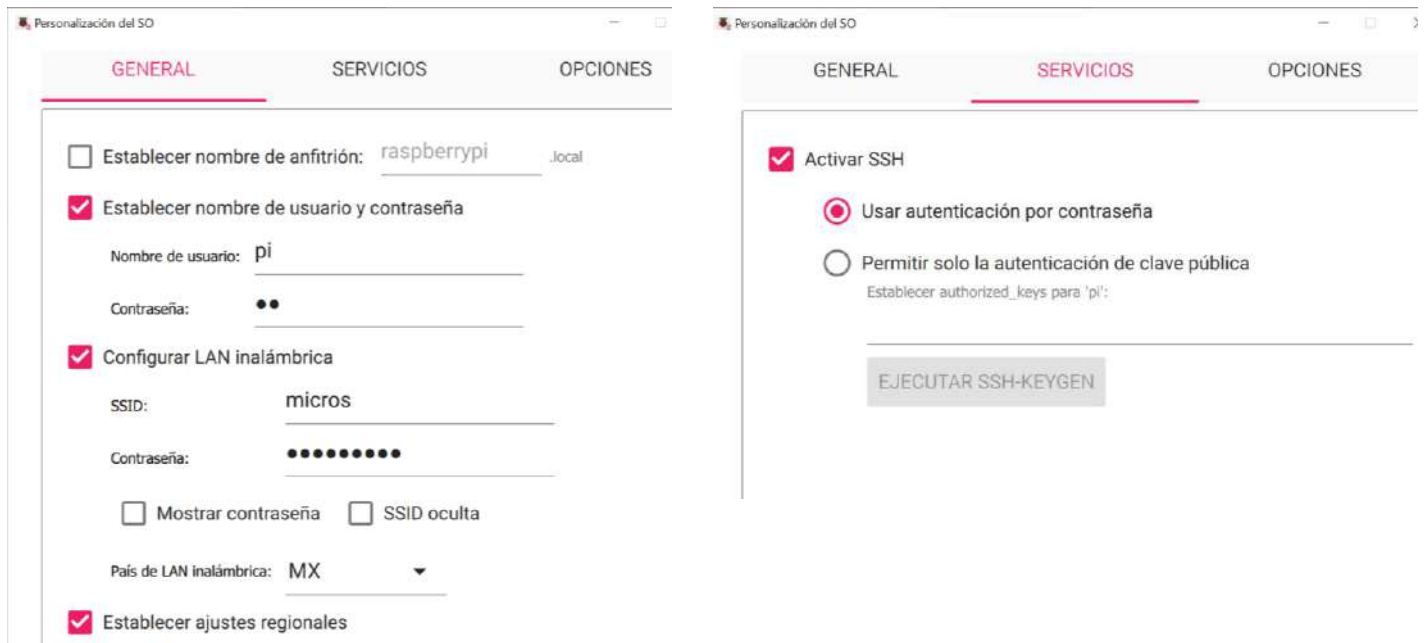




Figura 11. Personalizaciones iniciales en el SO

Cuando termine las configuraciones; se solicitará la confirmación.



Figura 12. Ventana de confirmación de personalizaciones

Será iniciado la instalación.

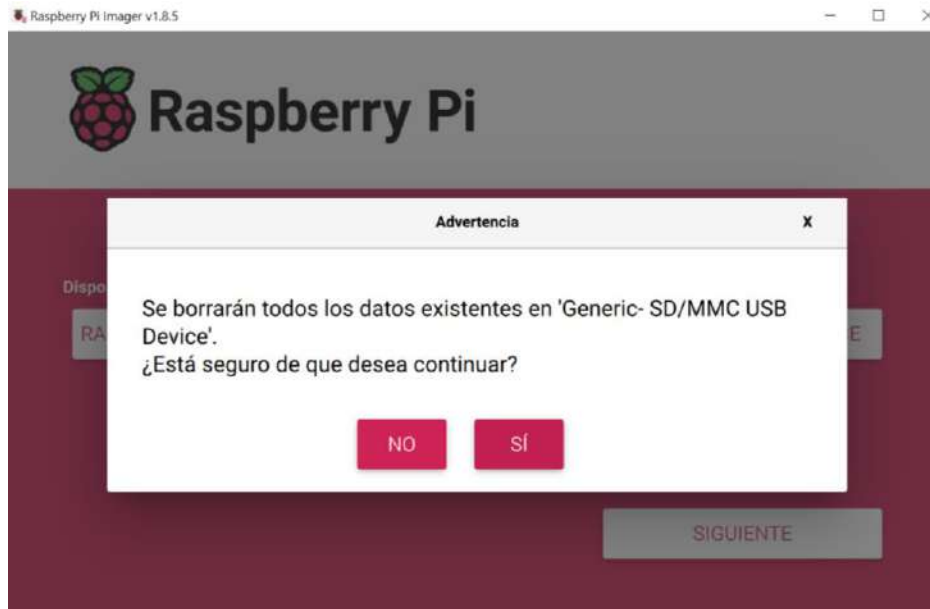


Figura 13. Confirmación de todo el proceso

Una vez configurado y confirmado, el sistema operativo será instalado en la unidad micro SD.



Figura 14. Ventana de preparación



Figura 15. Ventana del estado de instalación



Figura 16. Proceso terminado



Figura 17. Instalación completa

Como último paso en este proceso, será la extracción de la unidad micro SD de la computadora y conectarla a la Raspberry para concluir con la instalación en esa plataforma..

3

Raspberry Pi

Acceso Remoto

La conexión remota de Raspberry Pi usando un equipo de cómputo externo facilita en gran medida el uso de la plataforma, es un mecanismo de control cuando no se dispone de teclado, mouse o pantalla.

Entre los servicios de acceso remoto más empleados se encuentran:

-
- I. **SSH (Secure Shell)**; permite acceso a la terminal de Raspberry Pi.
 - II. **VNC (Virtual Network Computing)**; acceso al ambiente grafico de Raspberry Pi.
-

En ambos casos, es necesario conectarse a la red y conocer la dirección IP asignada a la tarjeta Raspberry Pi; existen varios métodos para identificarlo, podrá elegir el de su preferencia.

Contenido:

-
1. *Habilitar SSH.*
 2. *Conexión SSH.*
 3. *Habilitar VNC.*
 4. *Descargar y habilitar VNC Server.*
 5. *Conexión VNC Server.*
-

1. Activación SSH.

Se requiere habilitar en la Raspberry Pi el protocolo SSH; métodos:

- I. Al momento de configurar la instalación del sistema operativo.

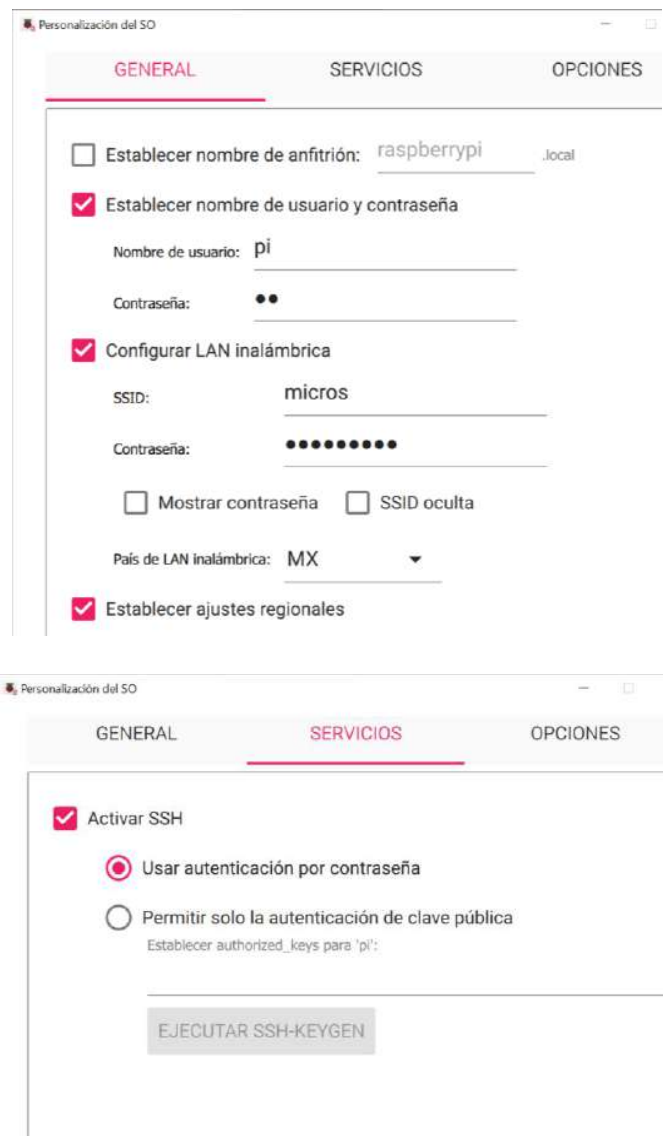


Figura 1. Activar SSH

- II. Una vez arrancado el SO, seleccionar **configuration** dentro de **preferences**.

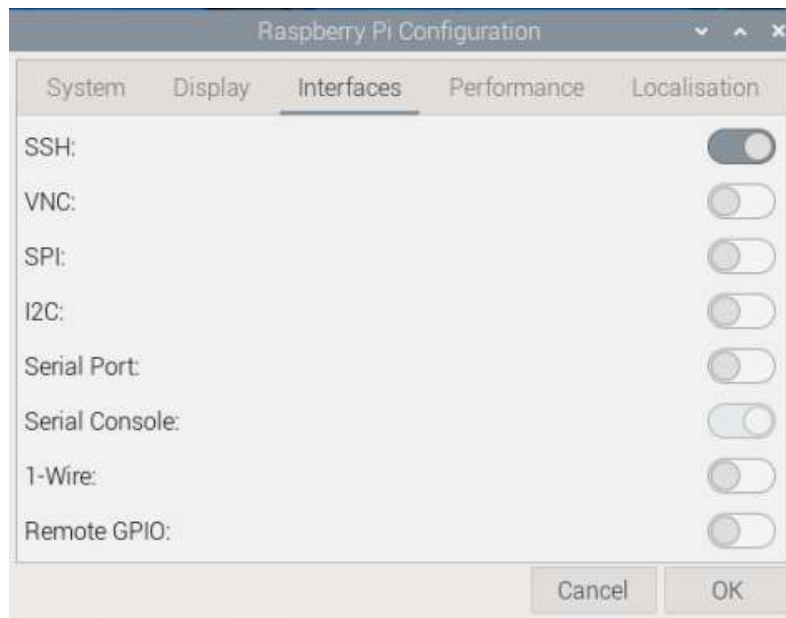
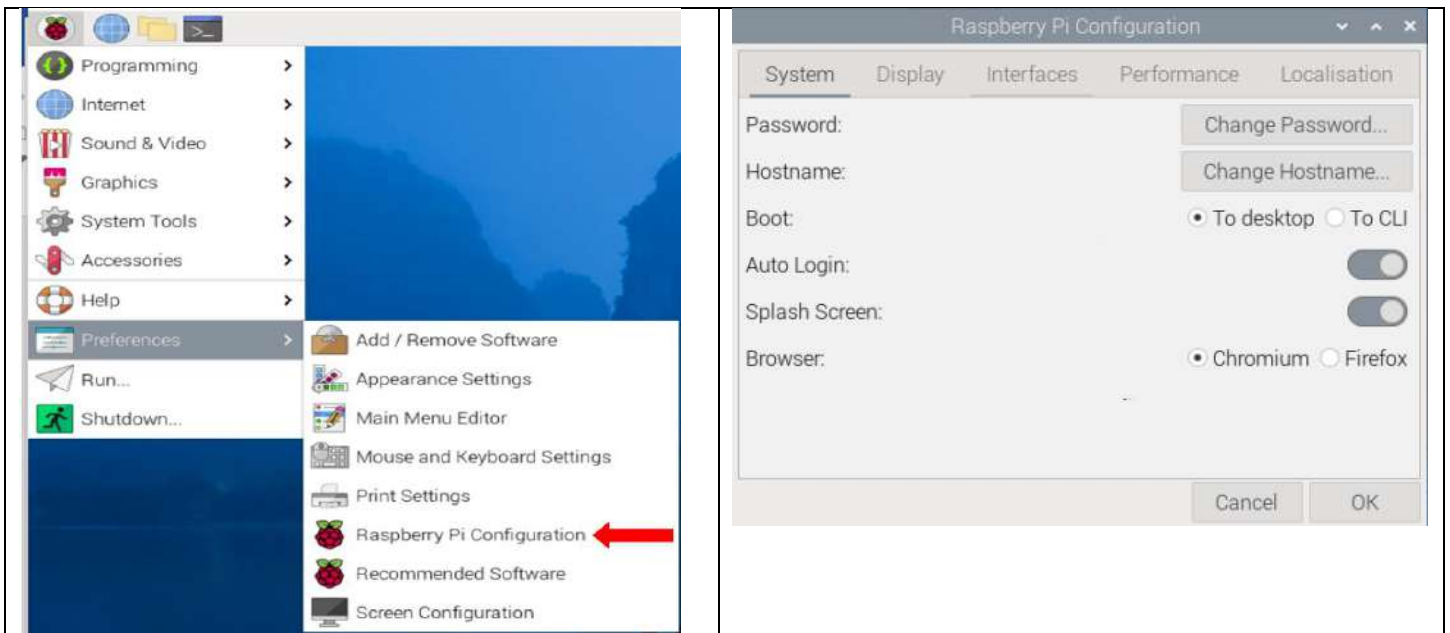


Figura 2. Configuración Raspberry Pi

2. Conexión SSH.

A. Conectar a través del SSH.

*Abrir la terminal (cmd) en Windows, escribir **ssh pi@ip_asignado** a la Raspberry Pi.*

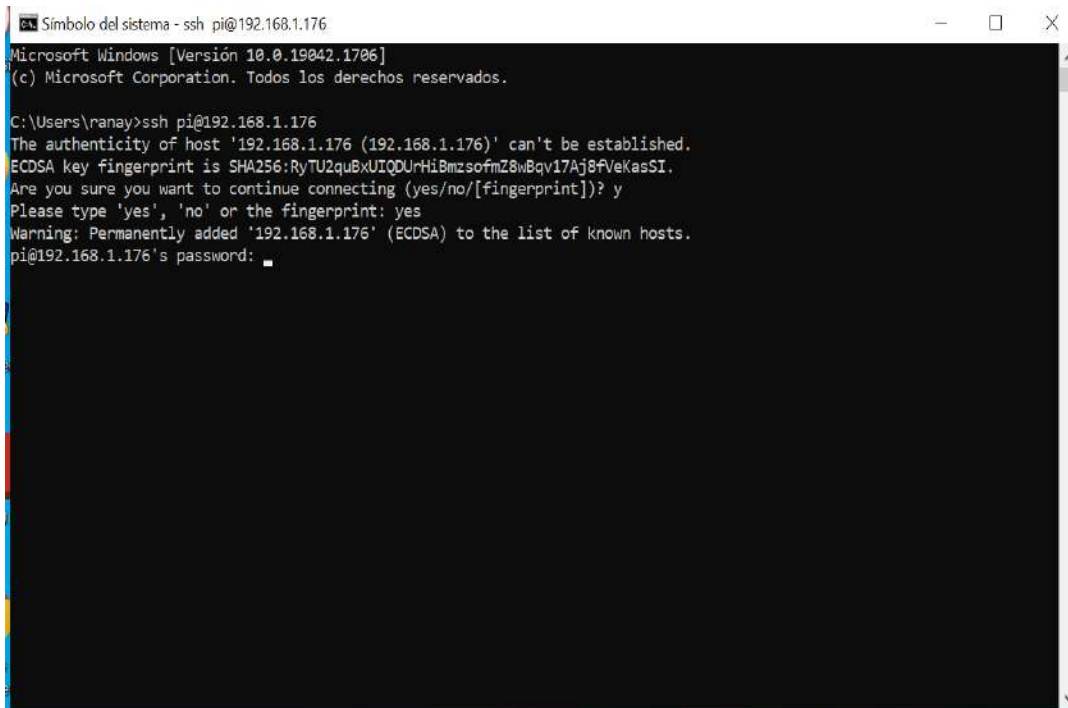


Figura 3. Conexión SSH

Solicitará las credenciales de acceso, al ser ingresados correctamente, tendrá el control de la plataforma (figura 6).

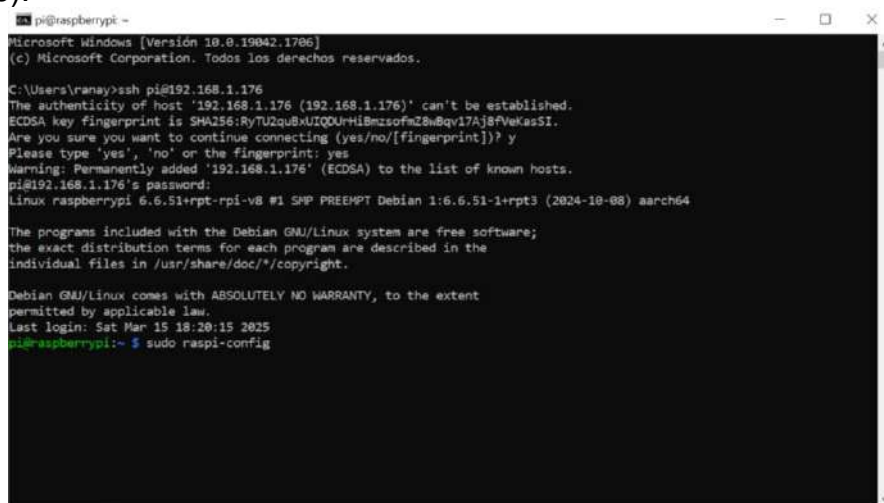


Figura 4. Control remoto

3.Habilitar VNC.

a. *Habilitar VNC; seguir el mismo procedimiento para SSH.*

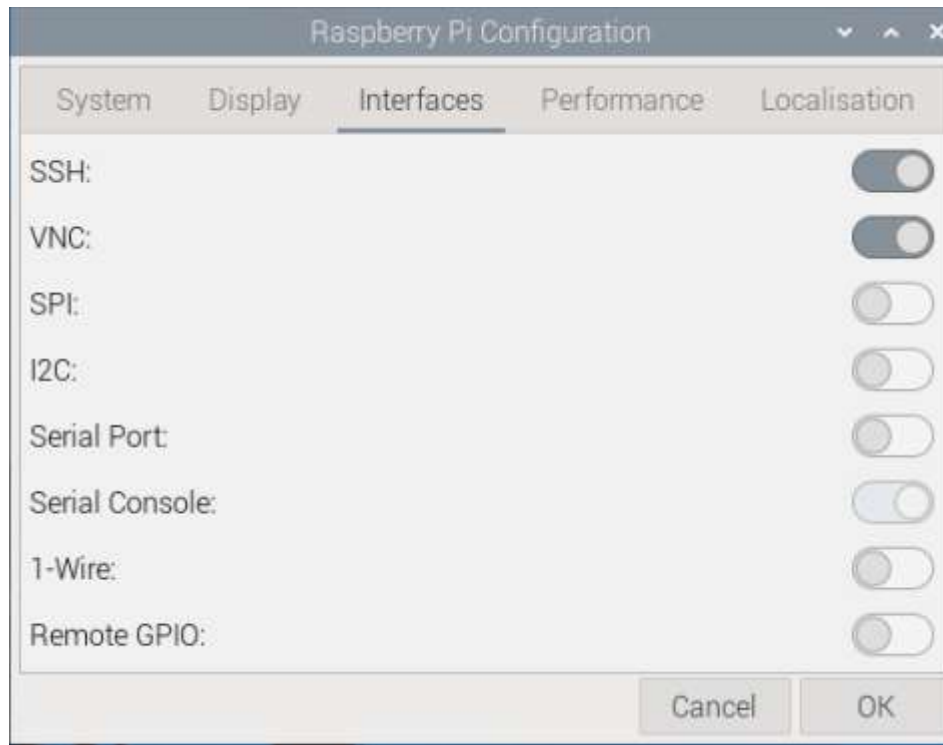


Figura 5. Habilitar VNC

b. *Configurar VNC a través de la conexión SSH.*

- i. *Usar procedimiento descrito en 1.1.*
 - ii. *Habilitar usando el menú de configuraciones.*
-

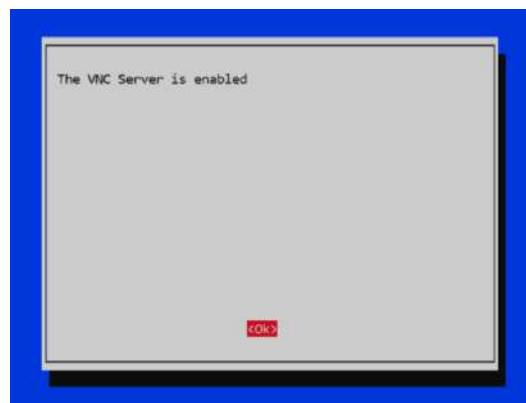
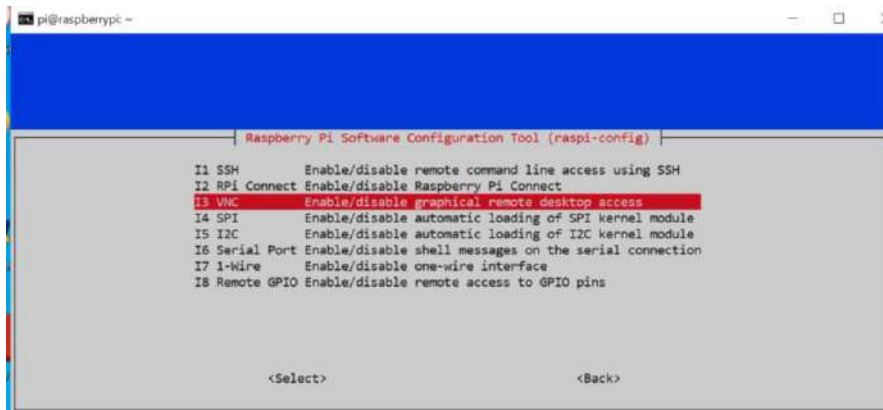
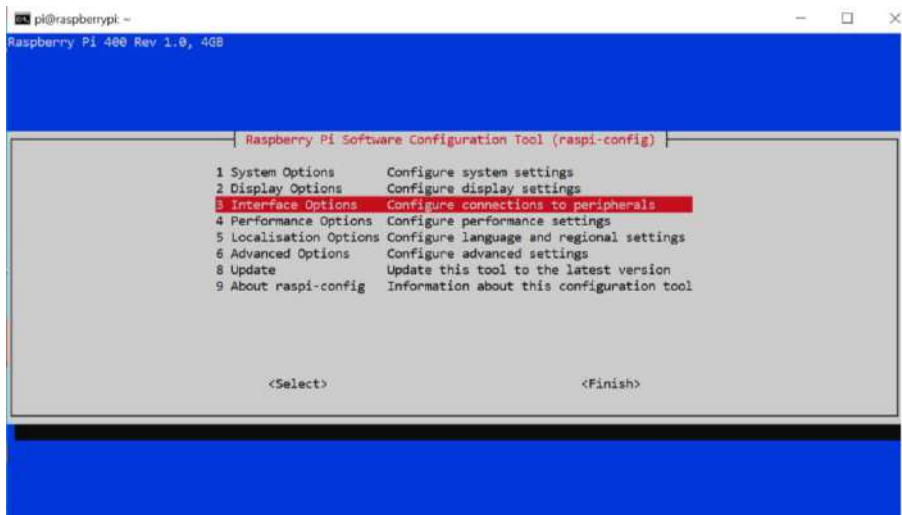


Figura 6. Habilitar server VNC

4. Descargar e instalar VNC Server.

Instalar VNC Server / Real VNC Viewer; disponible en línea;

<https://www.realvnc.com/en/connect/download/viewer/>.

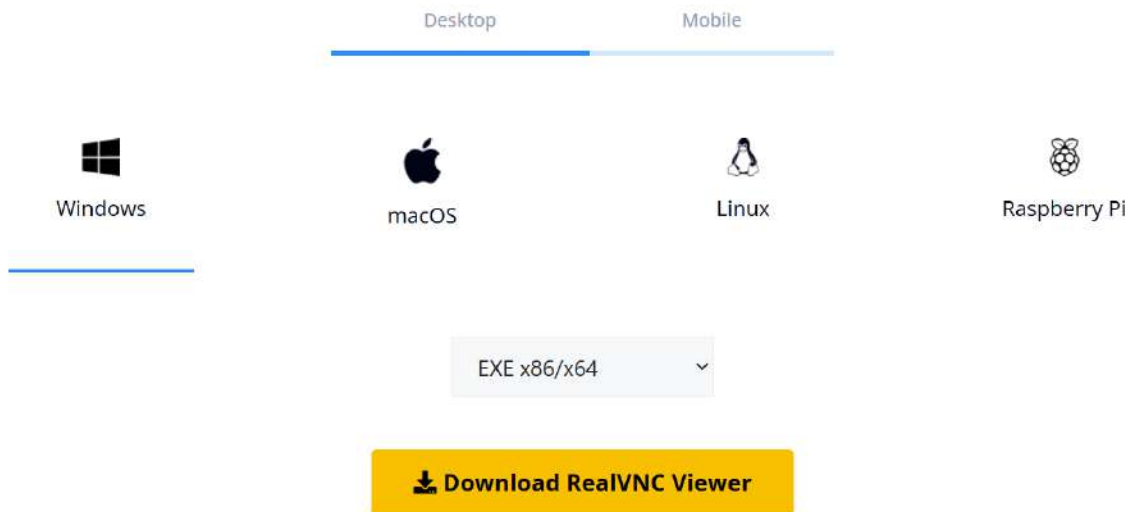


Figura 7. Descarga RealVNC Viewer

-
- i. *Seleccionar el sistema operativo de su equipo de cómputo.*
 - ii. *Esperar la descarga e instalar.*
-

5. Conexión con el servidor VNC.

Ejecutar RealVNC Viewer para conectar a la Raspberry; introducir la dirección IP; se solicitarán las credenciales de acceso.

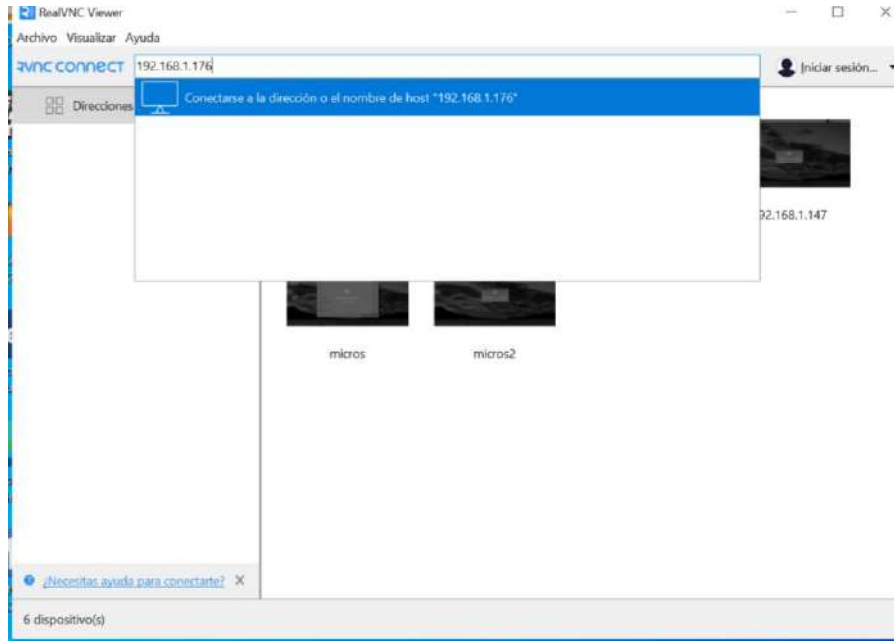


Figura 8. Ingreso IP.

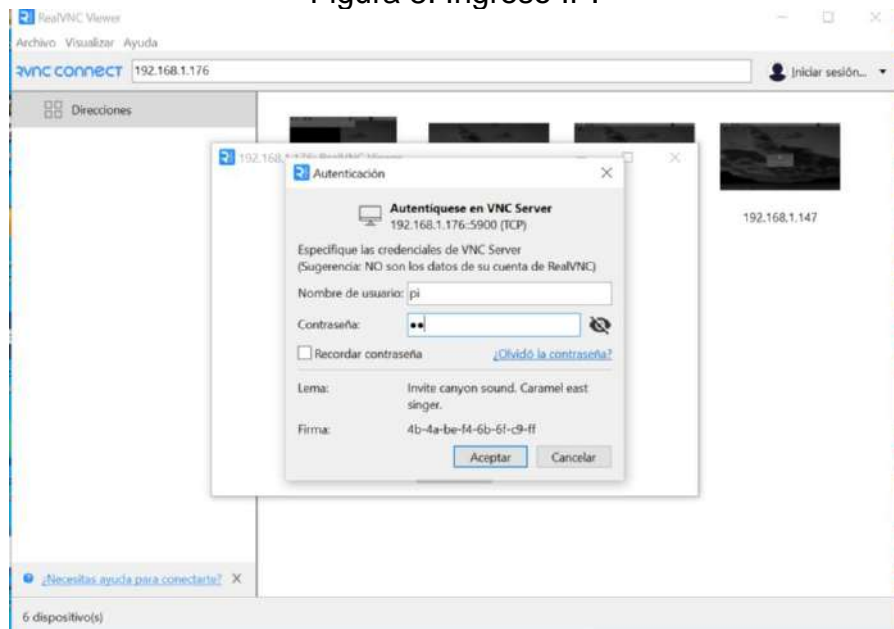


Figura 9. Ingreso de credenciales para control remoto

En esta etapa, tendrá control remoto de la Raspberry Pi.

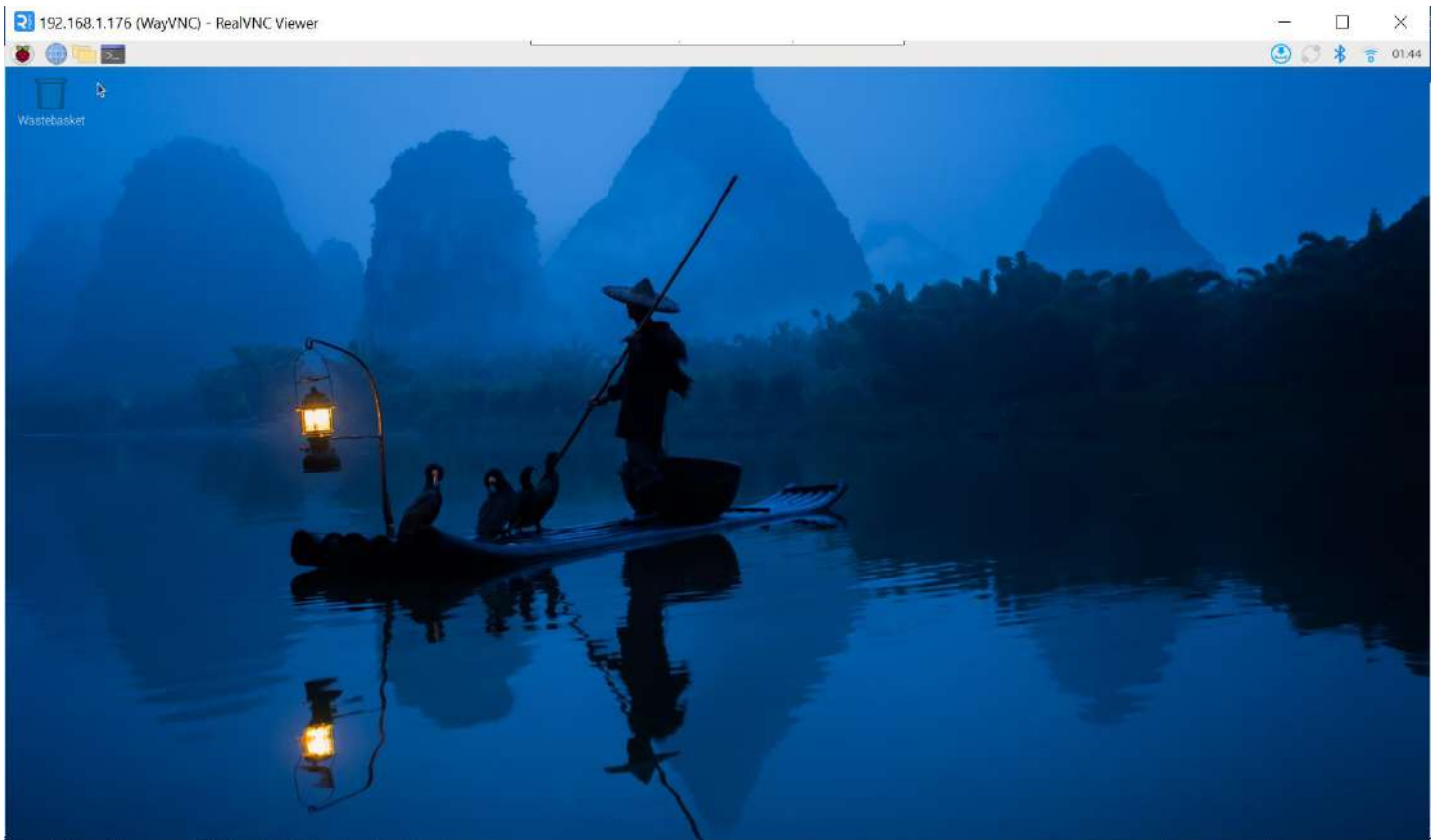


Figura 10. Escritorio del Sistema Operativo

4

Raspberry Pi

Ensamblador, Ligador y Debugger GDB

Editar y ensamblar un programa se puede realizar por varios mecanismos, a continuación, se estudiará el procedimiento que permitirá emplear los recursos del paquete GNU GCC Compiler y servicios del sistema operativo.

A. Editor de texto.

Existen una gran variedad de editores para captura, entre los sugeridos se encuentran:

1. Editor vi.

Es un editor que se ejecuta sobre una terminal.

-
- *Se invoca desde la línea de comandos de la consola:*
-

◆ *vi NombrePrograma.s*

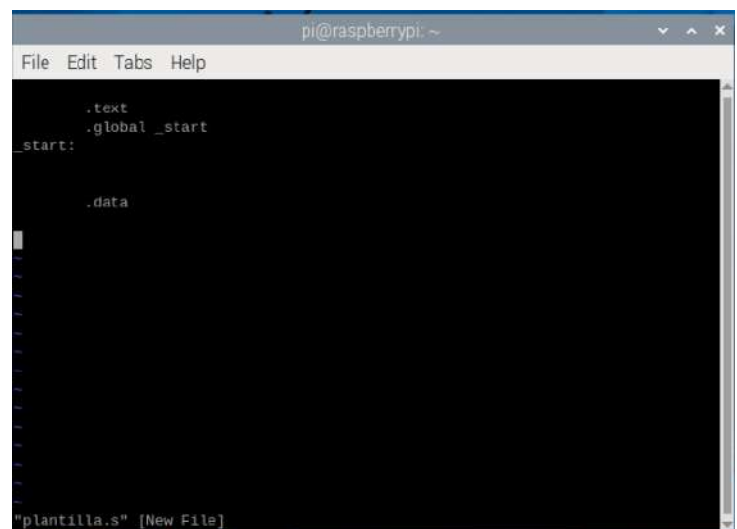


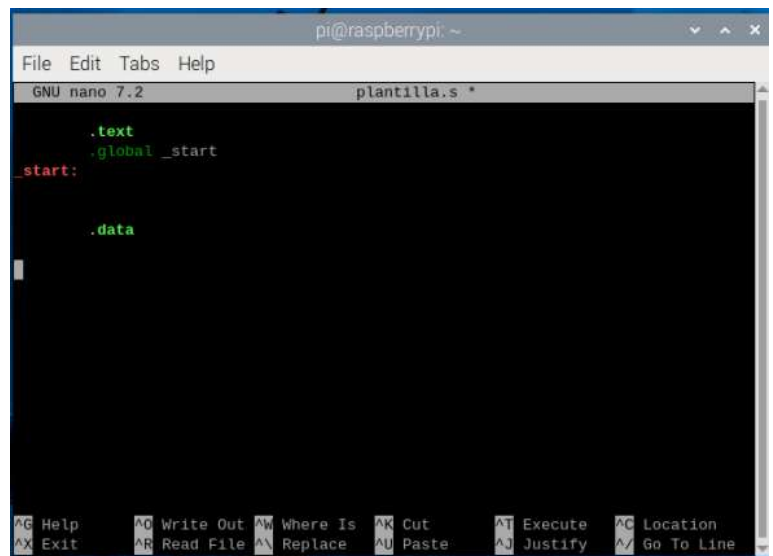
Figura 1. Editor vi

2. Editor nano.

Es un editor que se ejecuta sobre una terminal.

- *Se invoca desde la línea de comandos de la consola.*

◆ *nano NombrePrograma.s*



```
pi@raspberrypi: ~
File Edit Tabs Help
GNU nano 7.2 plantilla.s *
.text
.global _start
start:
.data
AG Help  AG Write Out  AW Where Is  AK Cut       AT Execute  AC Location
AX Exit  AR Read File  AW Replace  AU Paste    AJ Justify  AV Go To Line
```

Figura 2. Editor nano

3. Editor Geany.

Es un ambiente gráfico; se ubica en el software de programación del Sistema Operativo de Raspberry Pi.

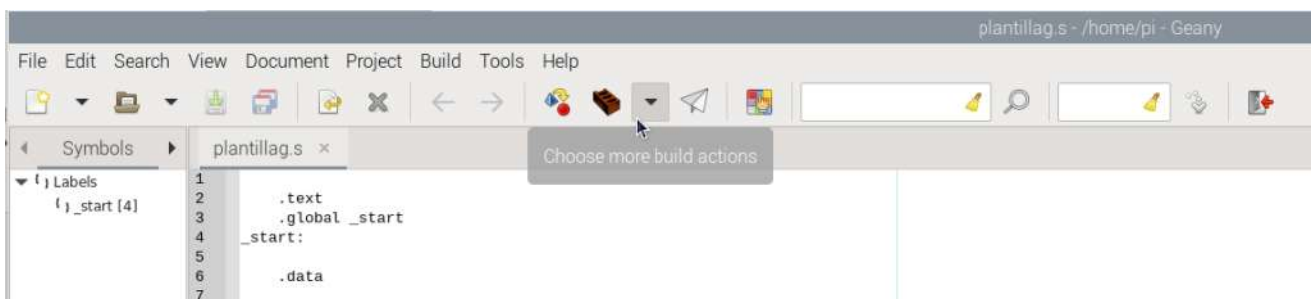


Figura 3. Editor Geany

El procedimiento a seguir independientemente de editor elegido:

1. *Editar el programa.*
2. *Ensamblar*
3. *Ligar*
4. *Ejecutar en línea o con el debugger gdb.*

a. Edición de un programa.

Escribir el código empleando el editor de su preferencia, guardar con la extensión .s.

```

GNU nano 7.2 e51.s
      .text
      .global _start
_start: mov r0,#5
        mov r1,#0x01
        subs r3,r0,r1
        beq igual
        bne diferente
igual:  mov r0,#1
        ldr r1,=texto1
        mov r2,#30
        mov r7,#4
        svc 0
        b fin
diferente: mov r0,#1
        ldr r1,=texto2
        mov r2,#33
        mov r7,#4
        svc 0
fin:    mov r0,r3
        mov r7,#1
        svc 0
      .data
texto1: .ascii "Datos iguales ... resultado = "
texto2: .ascii "Datos diferentes ... resultado = "

```

Terminal window title: pi@raspberrypi: ~/D.. e1.s - /home/pi/Desk..
 File Edit Tabs Help
 GNU nano 7.2 e51.s
 ^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
 ^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line M-E Redo M-6 Copy

Figura 4. Código; ensamblador Raspberry Pi

b. Ensamblado.

Generar el código objeto, usando el ensamblador *as*.

as -o NombrePrograma.o NombrePrograma.s

```
pi@raspberrypi:~/Desktop/Practicas $ as -o ejemplo.o ejemplo.s
```

Figura 5. Ensamblado

c. Ligado.

Generar el código ejecutable; se invoca el ligador *ld*.

ld -o NombrePrograma NombrePrograma.o

```
pi@raspberrypi:~/Desktop/Practicas $ ld -o ejemplo ejemplo.o
```

Figura 6. Ligado

En caso de no existir errores el proceso de ensamblado y ligado, no será desplegado alguna información de salida; en caso contrario enviará los mensajes de error, para su atención.

d. Ejecución.

Con el código ejecutable *NombrePrograma* (no tendrá extensión), se ejecutará el programa y se comprobará el resultado obtenido, deberá teclear:

./NombrePrograma ; echo \$?

```
pi@raspberrypi:~/Desktop/Practicas $ ./ejemplo ; echo $?
Datos diferentes ... resultado = 4
```

Figura 7. Ejecución

Por lo tanto, el procedimiento completo se muestra en la figura 8:

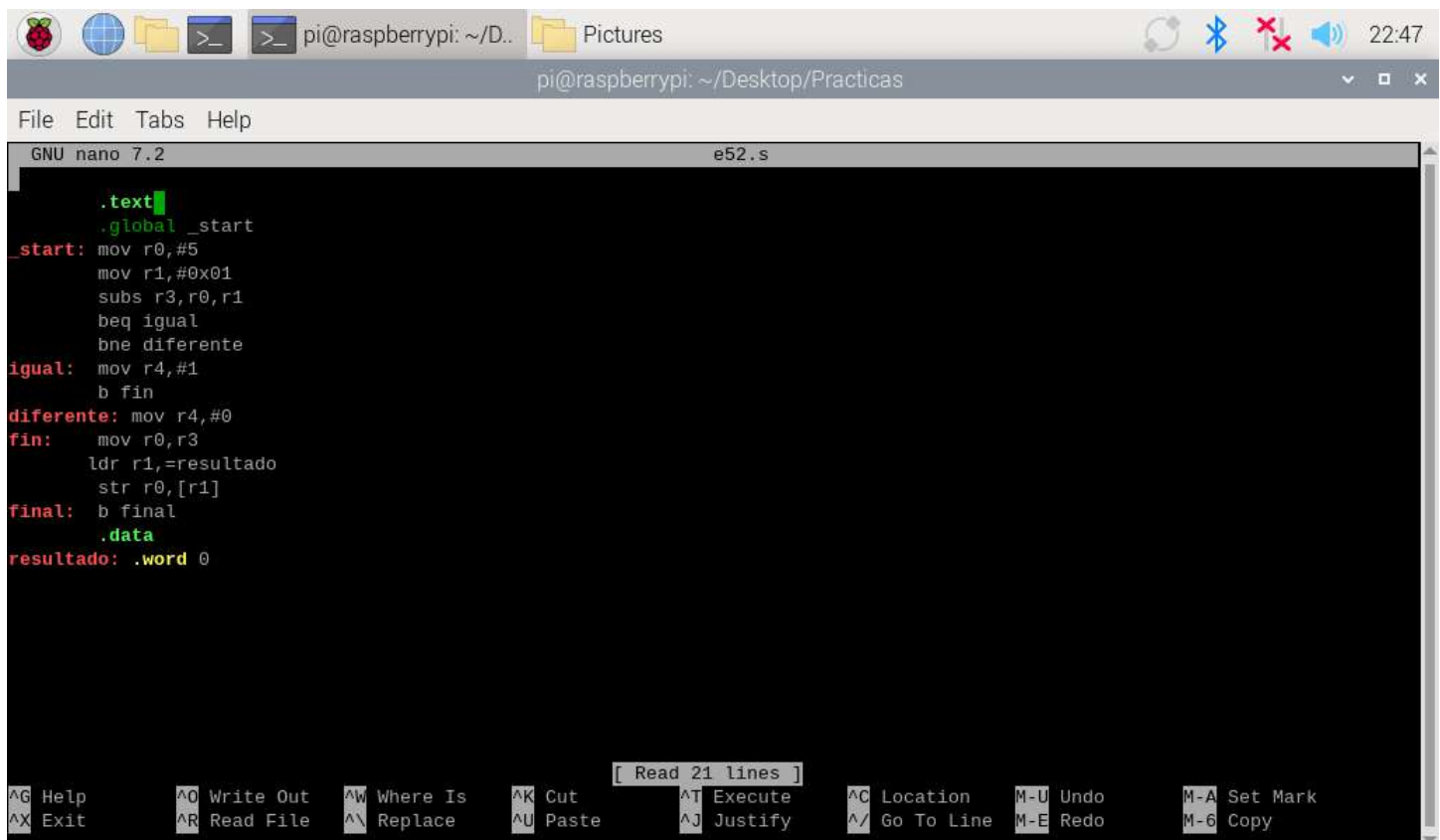
```
pi@raspberrypi:~/Desktop/Practicas $ nano ejemplo.s
pi@raspberrypi:~/Desktop/Practicas $ as -o ejemplo.o ejemplo.s
pi@raspberrypi:~/Desktop/Practicas $ ld -o ejemplo ejemplo.o
pi@raspberrypi:~/Desktop/Practicas $ ./ejemplo ; echo $?
Datos diferentes ... resultado = 4
pi@raspberrypi:~/Desktop/Practicas $
```

Figura 8. Etapas

En caso que un programa no haga uso de los servicios, librerías y prestaciones del GNU GCC Compiler de manera directa, será más sencillo la ejecución por pasos del programa mediante el uso de GDB.

B. Ejemplo.

Para explicar el uso de GDB; se implementará el procedimiento previo con el siguiente ejemplo.



```
GNU nano 7.2 e52.s
.text
.global _start
_start: mov r0,#5
        mov r1,#0x01
        subs r3,r0,r1
        beq igual
        bne diferente
igual:  mov r4,#1
        b fin
diferente: mov r4,#0
fin:    mov r0,r3
        ldr r1,=resultado
        str r0,[r1]
final:  b final
.data
resultado: .word 0

[ Read 21 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo     M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line M-E Redo     M-6 Copy
```

Figura 9. Código ejemplo

Generando el código ejecutable.

```
pi@raspberrypi:~/Desktop/Practicas $ as -g -o e52.o e52.s
pi@raspberrypi:~/Desktop/Practicas $ ld -o e52 e52.o
```

Figura 10. Proceso de ensamblado y ligado

*Notar que en el proceso de ensamblado se ha agregado el parámetro **-g** para habilitar requerimientos de gdb.*

*Para conocer el código generado, se recomienda revisar el contenido del programa objeto *.o con el comando:*

objdump -s -d NombrePrograma.o

```
Contents of section .text:
0000 0500a0e3 0110a0e3 013050e0 0000000a .....0P.....
0010 0100001a 0140a0e3 000000ea 0040a0e3 .....@.....@..
0020 0300a0e1 04109fe5 000081e5 feffffea .....
0030 00000000 ....
Contents of section .data:
0000 00000000 ....
Contents of section .debug_line:
0000 3b000000 03001c00 00000201 fb0e0d00 ;.....
0010 01010101 00000001 00000100 6535322e .....e52.
0020 73000000 00000005 02000000 00152f2f s.....//
0030 2f2f2f2f 2f2f2f2f 2f2c0202 000101 //...../,....
Contents of section .debug_info:
0000 22000000 02000000 00000401 00000000 ".....
0010 00000000 34000000 00000000 06000000 ....4.....
0020 21000000 0180 !.....
Contents of section .debug_abbrev:
0000 01110010 06110112 01030e1b 0e250e13 .....%..
0010 05000000 ....
Contents of section .debug_aranges:
0000 1c000000 02000000 00000400 00000000 .....
0010 00000000 34000000 00000000 00000000 ....4.....
Contents of section .debug_str:
0000 6535322e 73002f68 6f6d652f 70692f44 e52.s./home/pi/D
0010 65736b74 6f702f50 72616374 69636173 esktop/Practicas
0020 00474e55 20415320 322e3430 00 .GNU AS 2.40.
```

```

Contents of section .ARM.attributes:
 0000 41110000 00616561 62690001 07000000  A....aeabi.....
 0010 0801                                     ..

Disassembly of section .text:

00000000 <_start>:
 0:  e3a00005      mov     r0, #5
 4:  e3a01001      mov     r1, #1
 8:  e0503001      subs   r3, r0, r1
 c:  0a000000      beq    14 <igual>
10:  1a000001      bne    1c <diferente>

00000014 <igual>:
14:  e3a04001      mov     r4, #1
18:  ea000000      b      20 <fin>

0000001c <diferente>:
1c:  e3a04000      mov     r4, #0

00000020 <fin>:
20:  e1a00003      mov     r0, r3
24:  e59f1004      ldr     r1, [pc, #4]    @ 30 <final+0x4>
28:  e5810000      str     r0, [r1]

0000002c <final>:
2c:  eaffffffe     b      2c <final>
30:  00000000      .word  0x00000000

```

Figura 11. Salida código objeto

En caso de ejecutar el programa, no genera salida a la consola y se mostrará el contenido de la figura 12.

```

pi@raspberrypi:~/Desktop/Practicas $ ./e52 ; echo $?
^C
pi@raspberrypi:~/Desktop/Practicas $ █

```

Figura 12. Ejecución

C. GDB GNU Debugger.

Se invocará al GNU debugger escribiendo: ***gdb NombrePrograma***

pi@raspberrypi:~\$ gdb e52

```
pi@raspberrypi:~/Desktop/Practicas $ gdb e52
GNU gdb (Raspbian 13.1-3) 13.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from e52...
(gdb) █
```

Figura 13. Prompt del debugger gdb

Aparecerá el entorno mostrado en la figura 13; notar que el debugger queda listo para recibir comandos mediante su *prompt*: (***gdb***).

Comandos *gdb* más usados.

Comando	Abreviación	Descripción	Ejemplo																																				
break	b	<i>Establece un breakpoint.</i>	b 5																																				
run	r	<i>Ejecución del programa en un solo paso o hasta encontrar un break point.</i>	r																																				
list	l	<i>Despliega código.</i>	l																																				
info registers	ir	<i>Despliega información de registros.</i>	ir																																				
disassemble	disas	<i>Muestra código desensamblado.</i>	disas																																				
stepi	stepi	<i>Ejecuta una o más instrucciones.</i>	stepi stepi 10																																				
continue	cont	<i>Continúa la ejecución de un programa de manera continua, hasta encontrar otro breakpoint; en caso de quedar en un ciclo infinito, salir con Ctrl-C.</i>	c																																				
examine memory /nfs address address	x /nfs address	<p><i>Examina el contenido de memoria; donde:</i></p> <table border="1"> <thead> <tr> <th colspan="2">n Número localidades.</th> </tr> </thead> <tbody> <tr> <td>F</td> <td>Formato de despliegue:</td> </tr> <tr> <td>o</td> <td>Octal</td> </tr> <tr> <td>x</td> <td>Hexadecimal</td> </tr> <tr> <td>d</td> <td>Decimal</td> </tr> <tr> <td>u</td> <td>Unsigned dec.</td> </tr> <tr> <td>t</td> <td>Bit.</td> </tr> <tr> <td>f</td> <td>Float.</td> </tr> <tr> <td>a</td> <td>Address.</td> </tr> <tr> <td>h</td> <td>Halfword.</td> </tr> <tr> <td>w</td> <td>Word.</td> </tr> <tr> <td>g</td> <td>Giant (8 bytes).</td> </tr> <tr> <td>s</td> <td>Size</td> </tr> <tr> <td>b</td> <td>Byte.</td> </tr> <tr> <td>h</td> <td>Halfword.</td> </tr> <tr> <td>w</td> <td>Word.</td> </tr> <tr> <td>g</td> <td>Giant (8 bytes).</td> </tr> <tr> <td colspan="2">Address.</td> </tr> </tbody> </table>	n Número localidades.		F	Formato de despliegue:	o	Octal	x	Hexadecimal	d	Decimal	u	Unsigned dec.	t	Bit.	f	Float.	a	Address.	h	Halfword.	w	Word.	g	Giant (8 bytes).	s	Size	b	Byte.	h	Halfword.	w	Word.	g	Giant (8 bytes).	Address.		x/4xw 0x1000
n Número localidades.																																							
F	Formato de despliegue:																																						
o	Octal																																						
x	Hexadecimal																																						
d	Decimal																																						
u	Unsigned dec.																																						
t	Bit.																																						
f	Float.																																						
a	Address.																																						
h	Halfword.																																						
w	Word.																																						
g	Giant (8 bytes).																																						
s	Size																																						
b	Byte.																																						
h	Halfword.																																						
w	Word.																																						
g	Giant (8 bytes).																																						
Address.																																							
quit	q	<i>Salir de gdb.</i>	q																																				

Tabla 1. Comandos gdb

Ejemplos de uso:

- a. Listar el programa.

```
Reading symbols from e52...
(gdb) l
1
2     .text
3     .global _start
4     _start: mov r0, #5
5           mov r1, #0x01
6           subs r3, r0, r1
7           beq igual
8           bne diferente
9     igual: mov r4, #1
10          b fin
(gdb) l
11     diferente: mov r4, #0
12     fin:     mov r0, r3
13           ldr r1, =resultado
14           str r0, [r1]
15     final:  b final
16     .data
17     resultado: .word 0
18
19
20
(gdb) █
```

Figura 14. Comando l

- b. Establecer breakpoint.

```
(gdb) b 4
Breakpoint 1 at 0x100074: file e52.s, line 4.
```

Figura 15. Comando b

- c. Ejecutar hasta el breakpoint.

```
(gdb) r
Starting program: /home/pi/Desktop/Practicas/e52
Breakpoint 1, _start () at e52.s:4
4     _start: mov r0, #5
```

Figura 16. Comando r

d. Desplegar contenido de registros.

```
(gdb) i r
r0          0x0          0
r1          0x0          0
r2          0x0          0
r3          0x0          0
r4          0x0          0
r5          0x0          0
r6          0x0          0
r7          0x0          0
r8          0x0          0
r9          0x0          0
r10         0x0          0
r11         0x0          0
r12         0x0          0
sp          0xffffef100    0xffffef100
lr          0x0          0
pc          0x10074      0x10074 <_start>
cpsr       0x10          16
fpscr      0x0          0
tpidruro   <unavailable>
(gdb) █
```

Figura 17. Comando i r

e. Desplegar programa desensamblado.

```
(gdb) disas
Dump of assembler code for function _start:
=> 0x00010074 <+0>:      mov     r0, #5
    0x00010078 <+4>:      mov     r1, #1
    0x0001007c <+8>:      subs   r3, r0, r1
    0x00010080 <+12>:     beq    0x10000 <igual>
    0x00010084 <+16>:     bne    0x10000 <diferente>
End of assembler dump.
```

Figura 18. Comando disas

f. Desplegar programa desensamblado indicando inicio y final.

g.

```
(gdb) disas 0x10074, 0x100a0
Dump of assembler code from 0x10074 to 0x100a0:
=> 0x00010074 <_start+0>:      mov     r0, #5
    0x00010078 <_start+4>:      mov     r1, #1
    0x0001007c <_start+8>:      subs   r3, r0, r1
    0x00010080 <_start+12>:     beq    0x10000 <igual>
    0x00010084 <_start+16>:     bne    0x10000 <diferente>
    0x00010088 <igual+0>:      mov     r4, #1
    0x0001008c <igual+4>:      b      0x10000 <fin>
    0x00010090 <diferente+0>:    mov     r4, #0
    0x00010094 <fin+0>:        mov     r0, r3
    0x00010098 <fin+4>:        ldr     r1, [pc, #4] @ 0x100a4 <final+4>
    0x0001009c <fin+8>:        str     r0, [r1]
```

Figura 19. Comando disas dir_inc dir_fin

h. Ejecutar una instrucción.

```
(gdb) stepi
5          mov r1, #0x01
```

Figura 20. Comando stepi

i. Mostrar resultado de la ejecución.

```
(gdb) i r
r0          0x5          5
```

Figura 21. Comando i r

j. Continuar ejecución continua del programa.

```
(gdb) cont
Continuing.
^C
Program received signal SIGINT, Interrupt.
final () at e52.s:15
15      final: b final
```

Figura 22. Comando cont

k. Desplegar registros.

```
(gdb) i r
r0          0x4          4
r1          0x110a8       69800
r2          0x0          0
r3          0x4          4
r4          0x0          0
r5          0x0          0
r6          0x0          0
r7          0x0          0
r8          0x0          0
r9          0x0          0
r10         0x0          0
r11         0x0          0
r12         0x0          0
sp          0xffffef100    0xffffef100
lr          0x0          0
pc          0x100a0       0x100a0 <final>
cpsr       0x20000010     536870928
fpscr      0x0          0
```

Figura 23. Comando i r

l. Mostrar memoria.

```
(gdb) x /8xw 0x110a8
0x110a8:      0x00000004      0x00001141      0x61656100      0x01006962
0x110b8:      0x00000007      0x00000108      0x0000001c      0x00000002
```

Figura 24. Comando x

m. Salir de gdb.

Comando: **q** (*quit*)

5

Raspberry Pi

Entorno de Desarrollo Integrado Code::Blocks

El IDE Code::blocks permite el uso del paquete GNU GCC Compiler para escribir, compilar y ejecutar programas escritos en ensamblador dentro de la plataforma Raspberry.

I. Instalación.

Una vez corriendo el sistema operativo en la Raspberry, se requiere de realizar lo siguiente:

a. *Actualizar los paquetes:*

```
sudo apt-get update
```

b. *Instalar Code::Blocks*

```
sudo apt-get install codeblocks
```

II. Abrir Code:Blocks.

Una vez concluido el proceso de instalación, encontrará al IDE Code::Blocks en el grupo de programación.

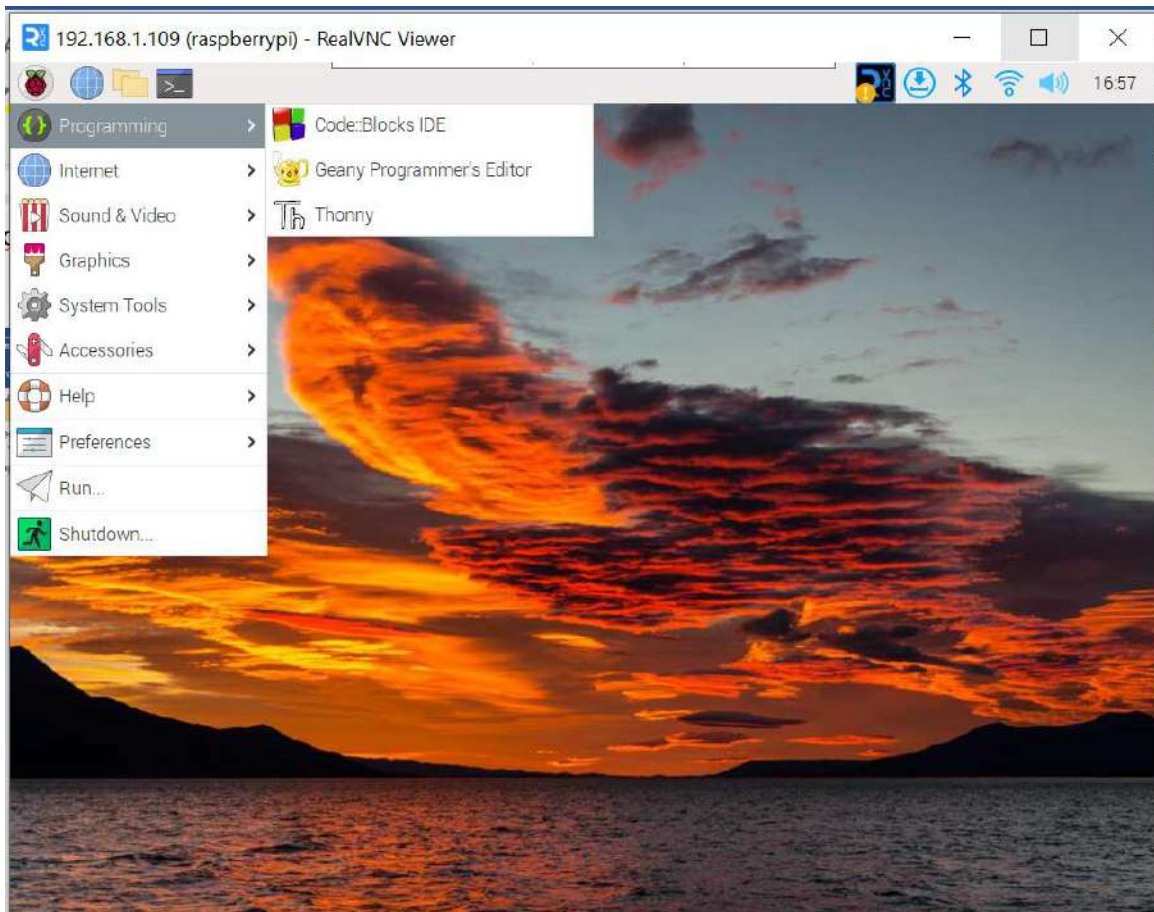


Figura 1. Ejecución Code::Blocks

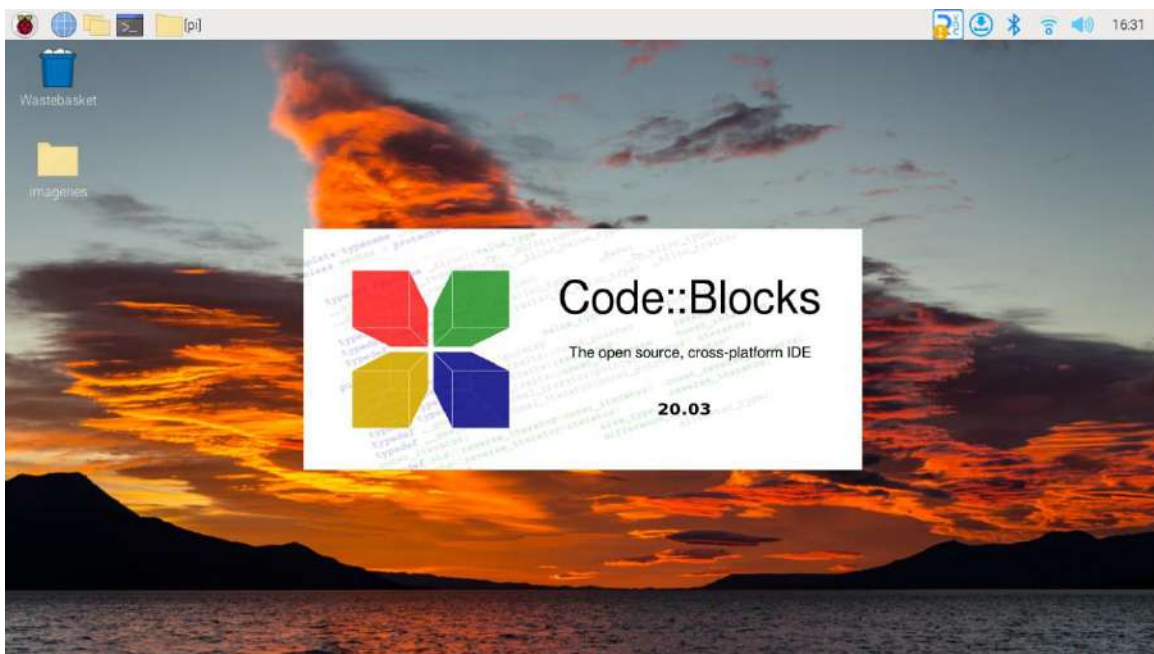


Figura 2. Etapa de arranque

El proceso para desarrollar programas en ensamblador consiste en:

1. Creación del proyecto.
2. Eliminar el archivo **main.c** del proyecto.
3. Agregar el archivo en ensamblador **nom_prog.s**.
4. Compilar el programa.
5. Ejecución del programa.

1. Crear proyecto nuevo.

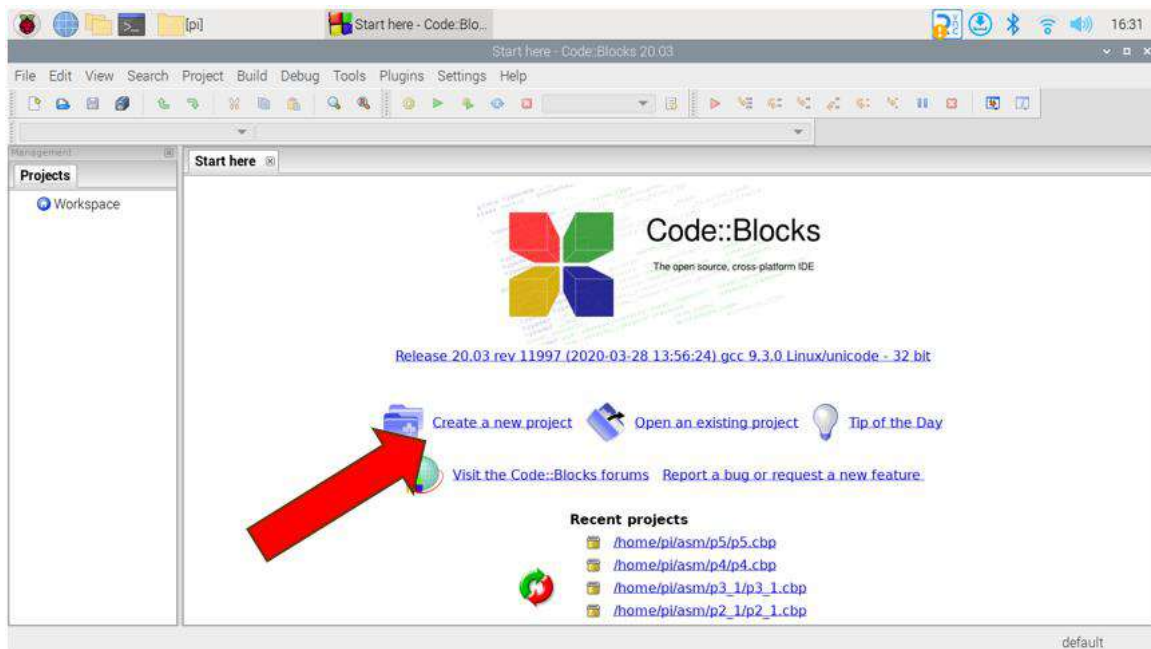


Figura 3. Creación de proyecto nuevo

1.1. Configurar proyecto.

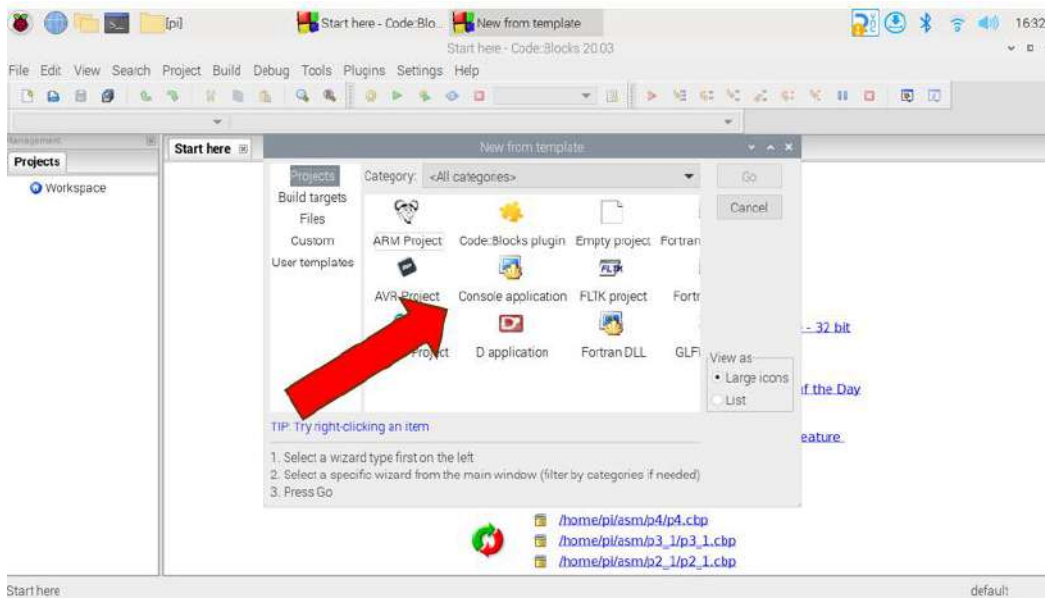


Figura 4. Seleccionar Console application

1.2. Seleccionar el lenguaje de programación.

Será utilizado el compilador de C, por lo tanto, seleccionar lenguaje C; entonces presionar Next.

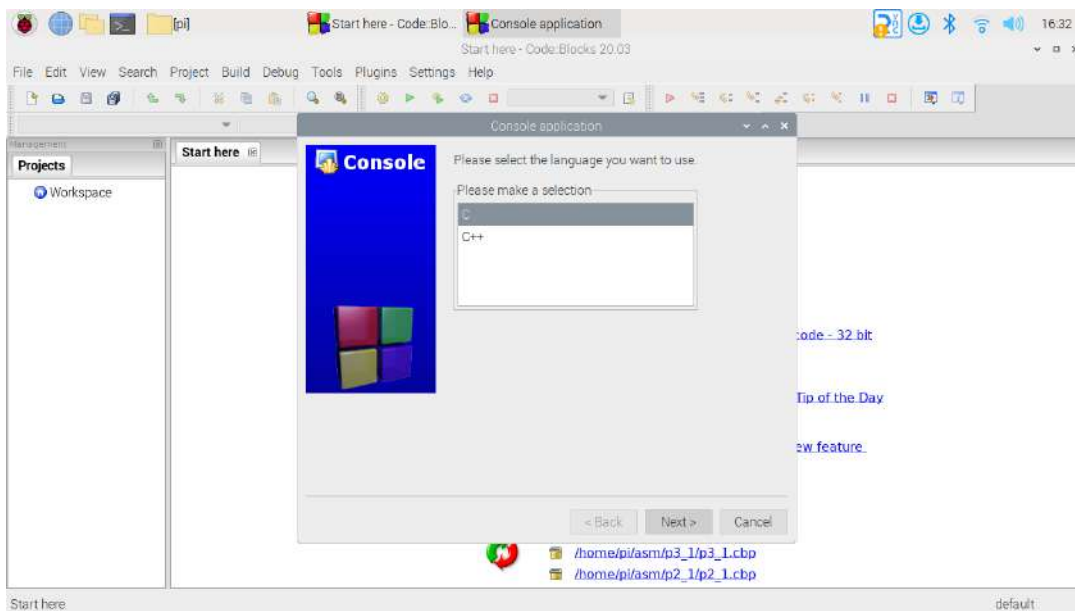


Figura 6. Configurar lenguaje

1.3. Nombrar el proyecto.

Indicar el nombre y ubicación del proyecto, continuar seleccionando Next.

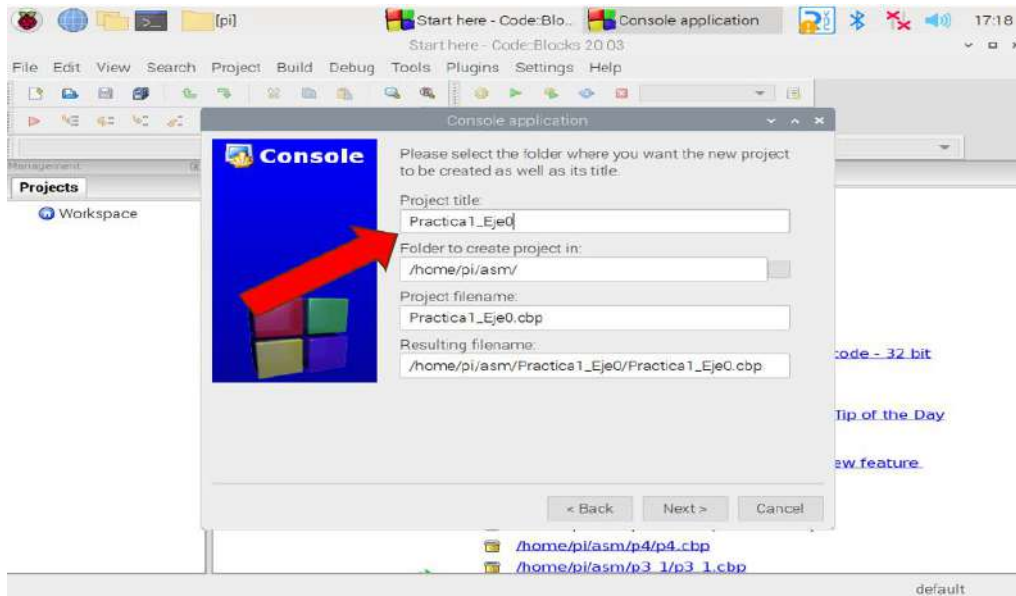


Figura 7. Nombrar proyecto

1.4. Mantener la configuración por defecto, solo presionar Finish.

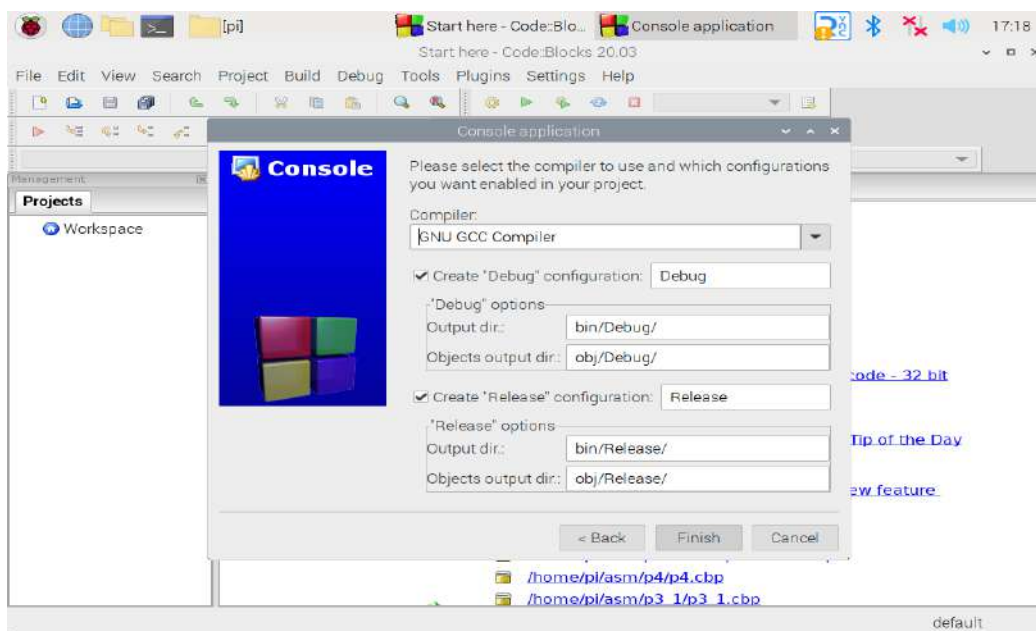


Figura 8. Concluir configuración del proyecto

La creación del proyecto genera el programa fuente de tipo C, con la plantilla mostrada en la figura 9.

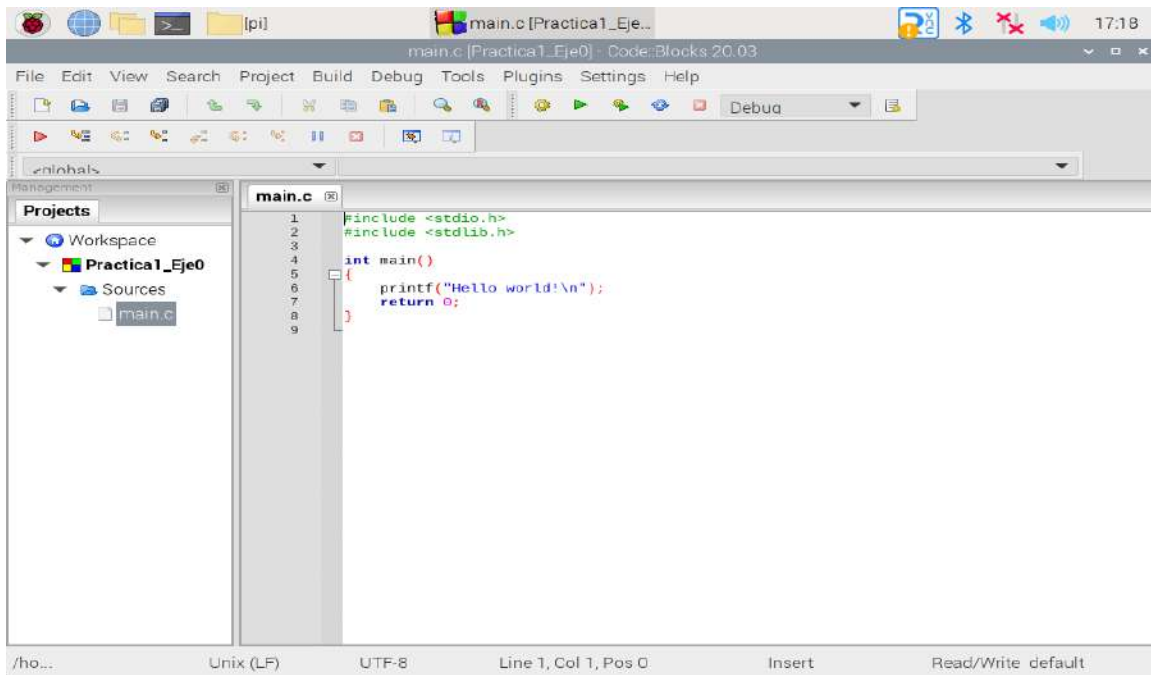


Figura 9, Proyecto completo

2. Eliminar programa main.c.

El programa main.c debe de ser eliminado, para ser intercambiado por el programa en ensamblador; presionar el botón secundario en el archivo main.c y seleccionar eliminar.

3. Agregar archivo fuente *.s.

Seleccionar File y nuevo para agregar al proyecto; hará la solicitud mostrada en la figura 10.

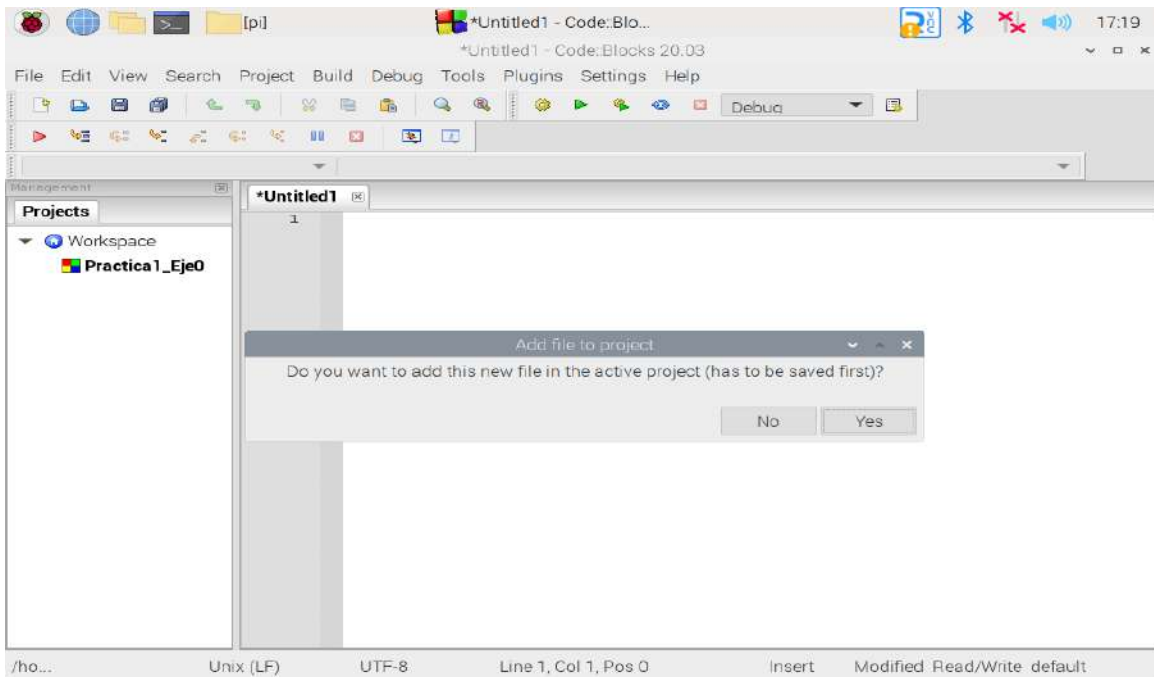


Figura 10. Confirmación al agregar el programa fuente al proyecto

Indicar el nombre del programa; este debe tener la extensión .s; salvar.

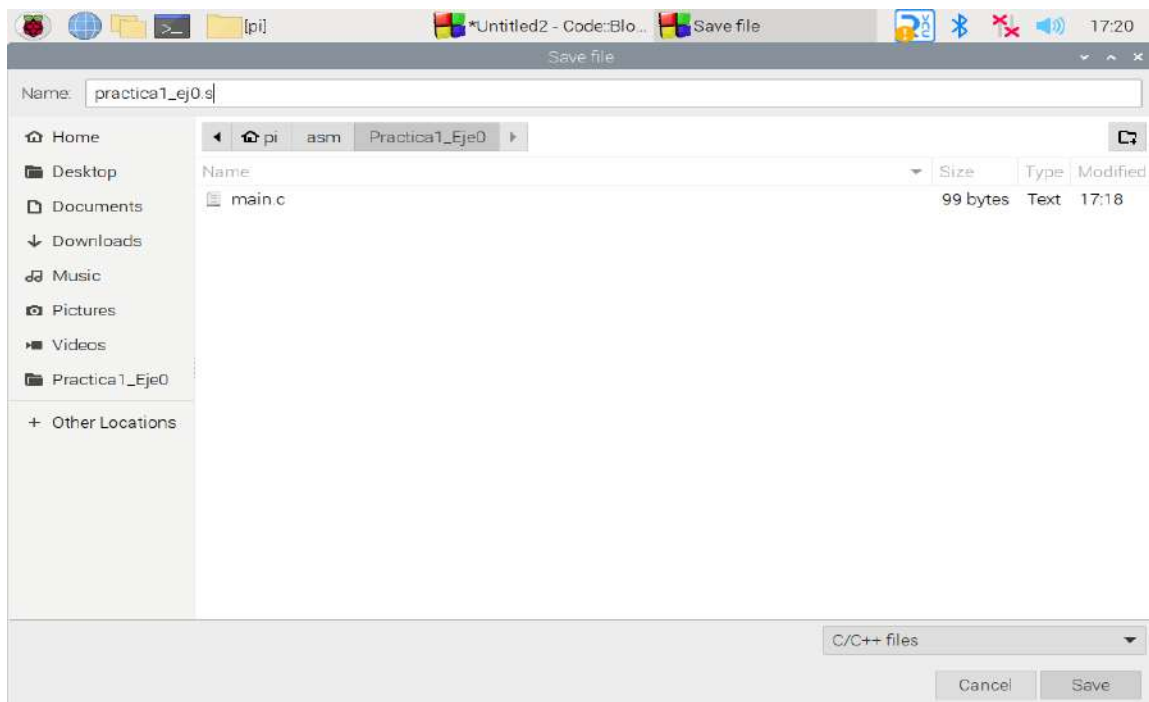


Figura 11. Nombre del archivo fuente

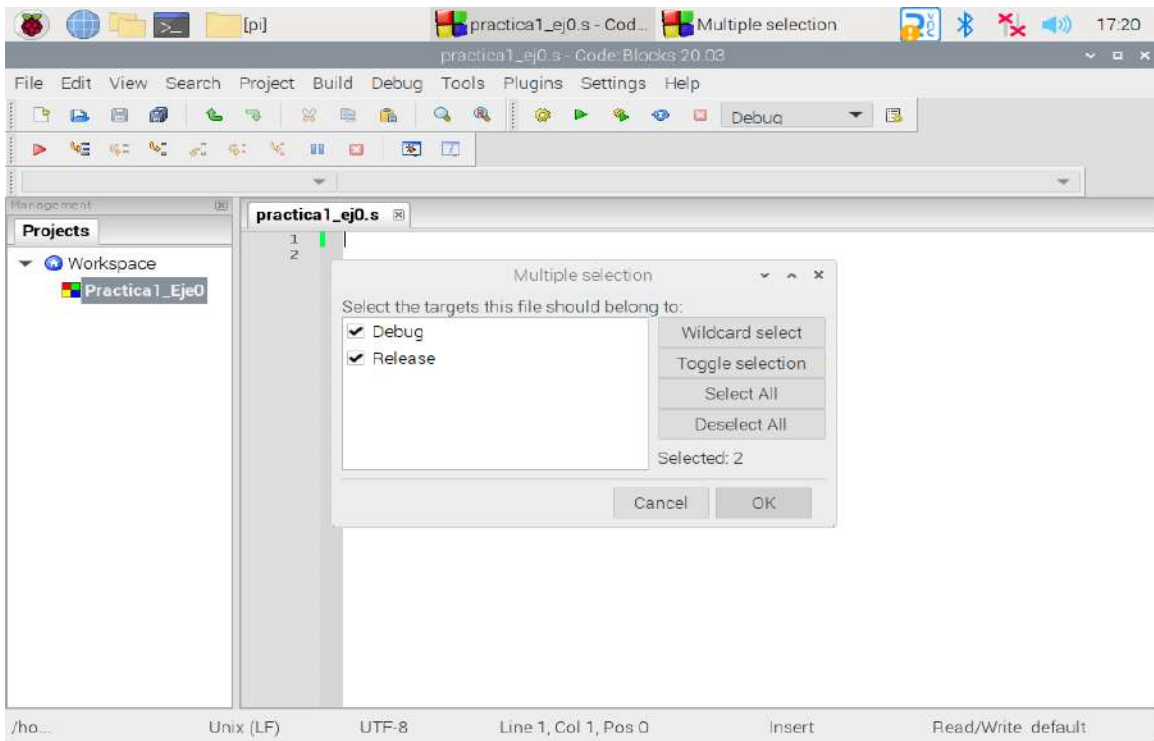


Figura 12. Mantener la configuración inicial

Se abrirá el entorno para edición del programa fuente.

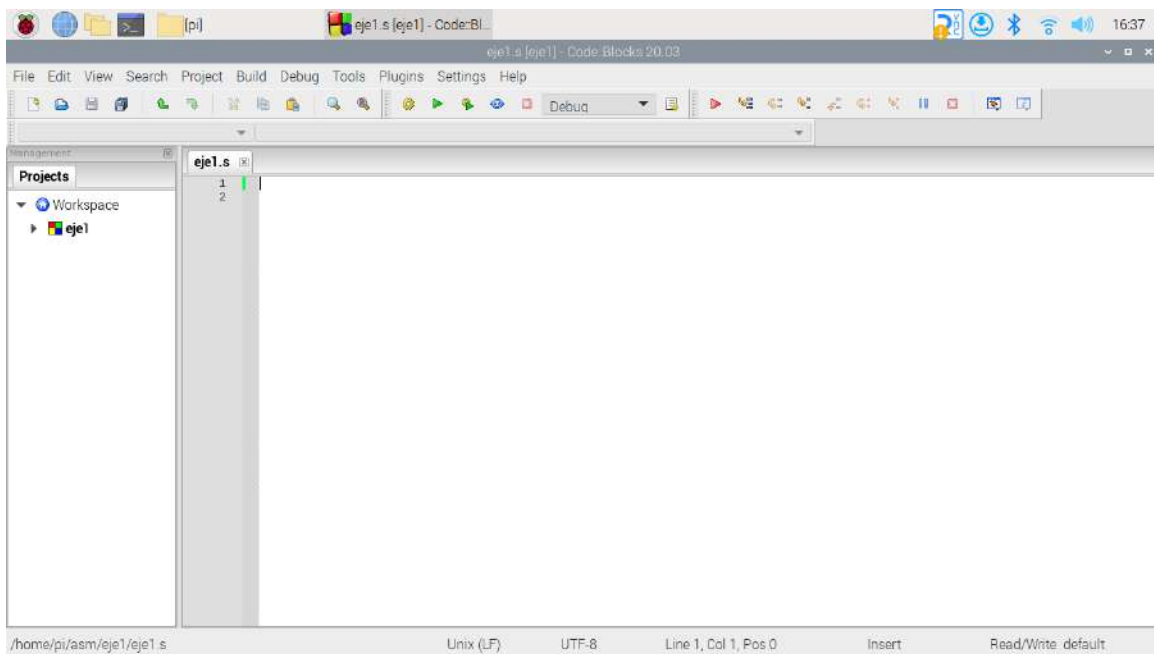


Figura 13. Entorno de edición

El siguiente paso, será la escritura del programa en ensamblador.

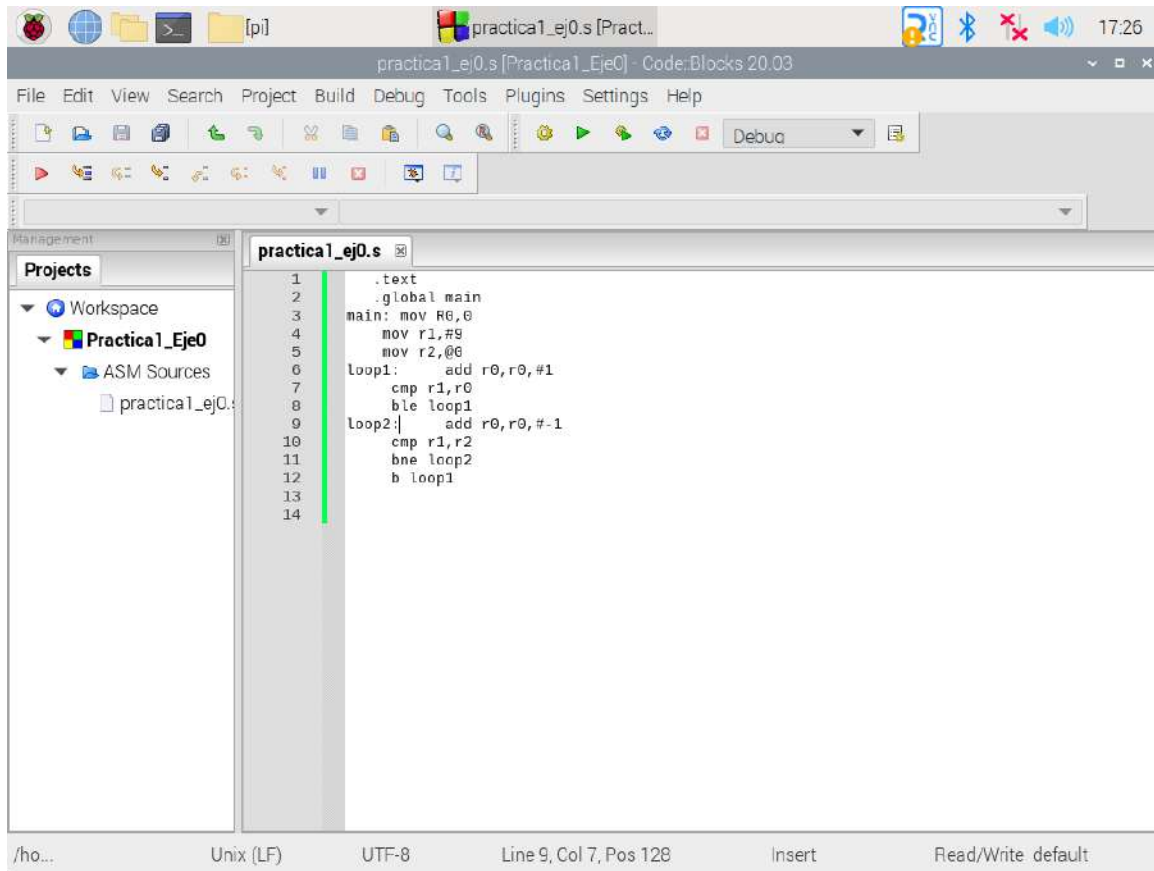


Figura 14. Programa fuente

4. Compilar el proyecto.

Terminado la escritura del programa fuente; proceder a compilar el programa;

presionar  (Compile).

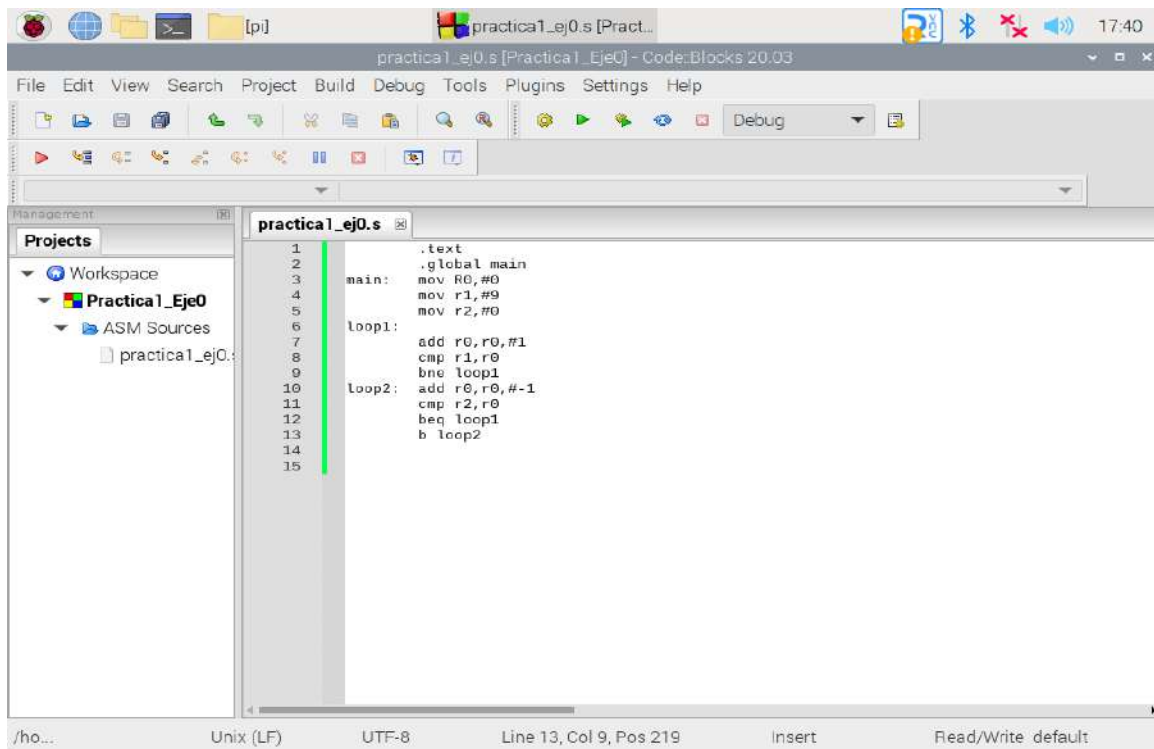

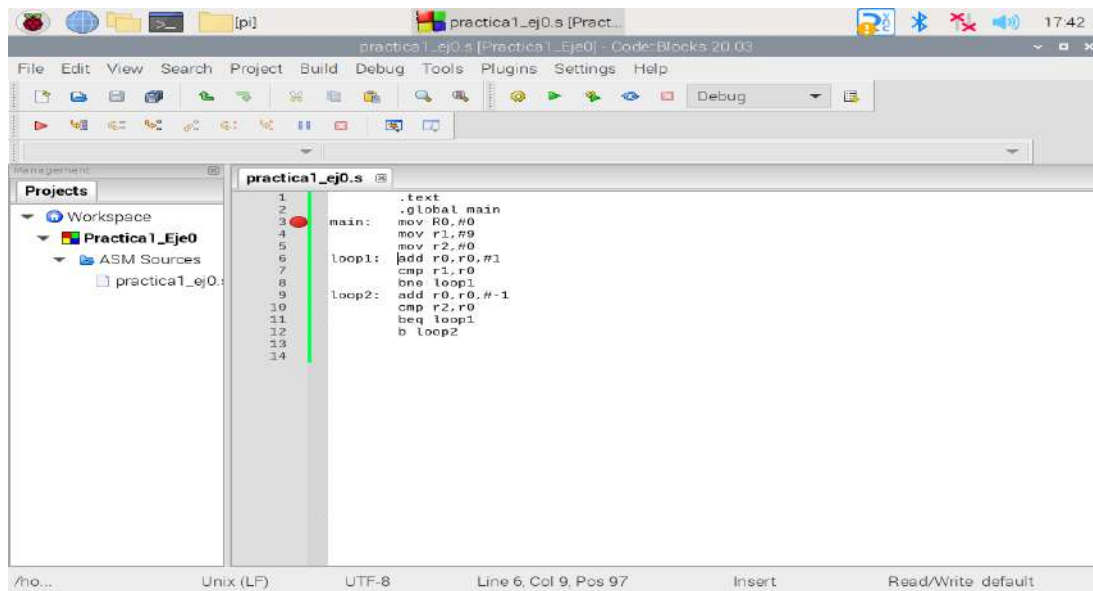



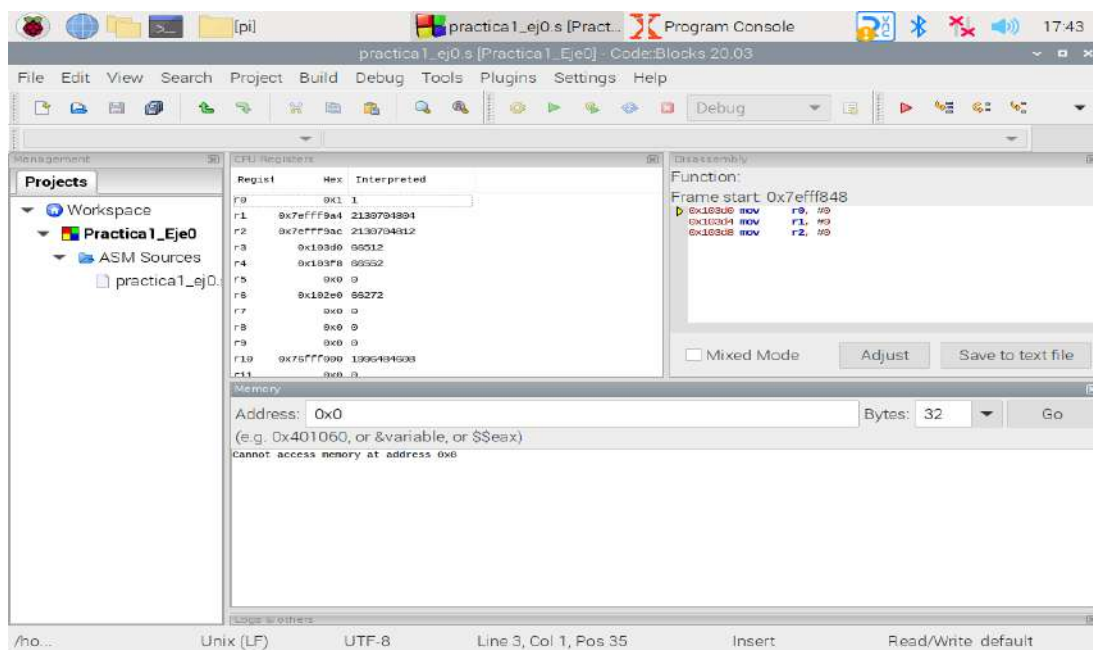
Figura 15. Proceso de compilación

En caso de no existir error, quedará listo para la ejecución, se requiere habilitar un break point en la primera instrucción del programa (colocarse del lado derecho del número de línea de esta, aparecerá un círculo rojo ( main: mov R0,#0)).

Figura 16. Ubicación del *break point*

5. Ejecución.

Presionar el comando  (*Debug/Stop*), para iniciar la ejecución del programa; Code::Block genera las pantallas para ejecución por pasos, mostrará parte del código la ventana del desensamblador, que será actualizado conforme se avance en el proceso.

Figura 17. Estado inicial del *Debugger*

Conociendo la dirección de asignación del programa, podrá ingresarla para ver su comportamiento durante la ejecución (Address:).

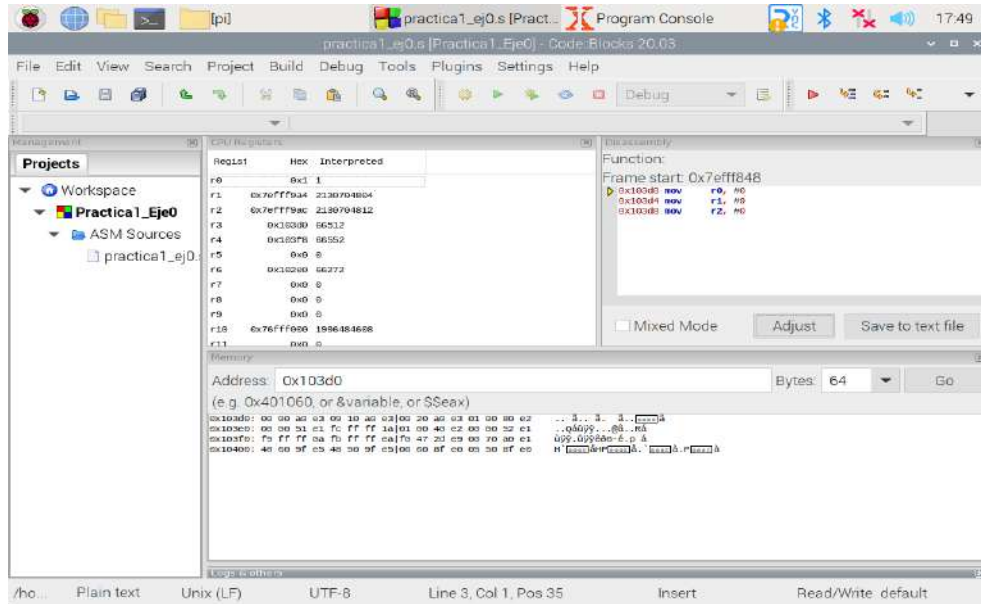


Figura 18. Estado actual de los registros y memoria

Presionar , para ejecución del programa por pasos y así comprobar el funcionamiento del programa.

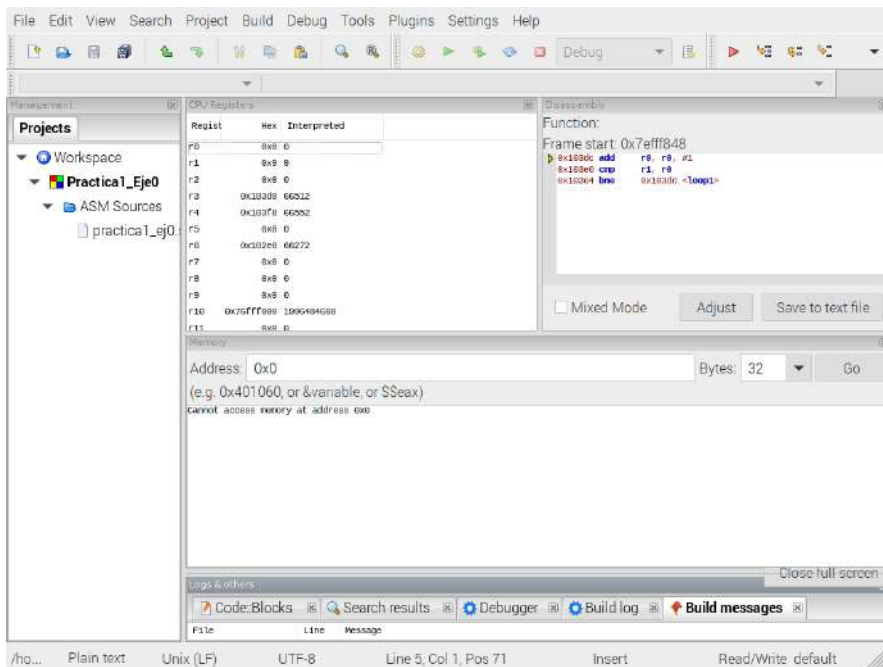


Figura 19. Resultado de la ejecución

6

CPULator

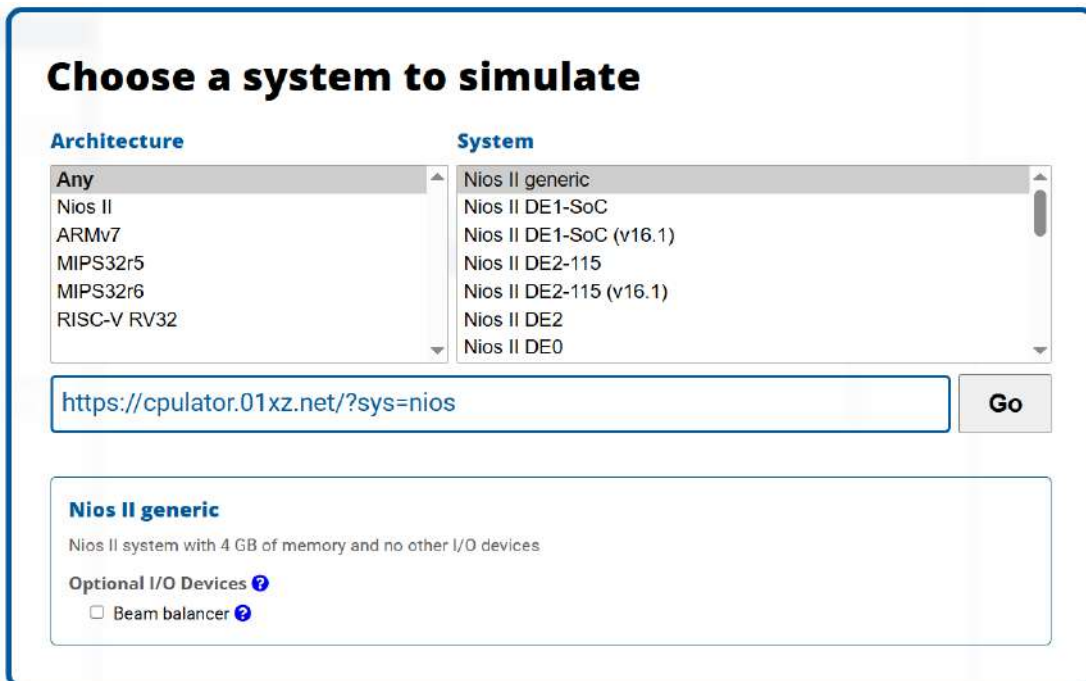
Simulador en línea

Es una herramienta útil para la escritura y depuración en línea de programas en ensamblador, aplica para diferentes versiones de arquitecturas. Será de interés las arquitecturas ARM, son asignadas las primeras direcciones de memoria para ubicar el código de programa y los datos.

El acceso al simulador lo puede realizar desde cualquier navegador, escribiendo cpulator; se selecciona la opción:

<https://cpulator.01xz.net/>

Aparecerá la pantalla de inicio en la que se seleccionará la arquitectura y el sistema a utilizar.



The screenshot shows the CPULator web interface with the title "Choose a system to simulate". It features two dropdown menus: "Architecture" and "System". The "Architecture" menu is set to "Any" and lists options: Nios II, ARMv7, MIPS32r5, MIPS32r6, and RISC-V RV32. The "System" menu is set to "Nios II generic" and lists options: Nios II DE1-SoC, Nios II DE1-SoC (v16.1), Nios II DE2-115, Nios II DE2-115 (v16.1), Nios II DE2, and Nios II DE0. Below the menus is a text input field containing the URL "https://cpulator.01xz.net/?sys=nios" and a "Go" button. At the bottom, there is a section for "Nios II generic" with a description "Nios II system with 4 GB of memory and no other I/O devices" and an "Optional I/O Devices" section with a checkbox for "Beam balancer".

Figura 1. Simulador CPULator

Raspberry Pi está diseñada usando la arquitectura **ARM**; por lo que se podrán simular programas en ensamblador seleccionando el sistema genérico de la arquitectura **ARMv7**.

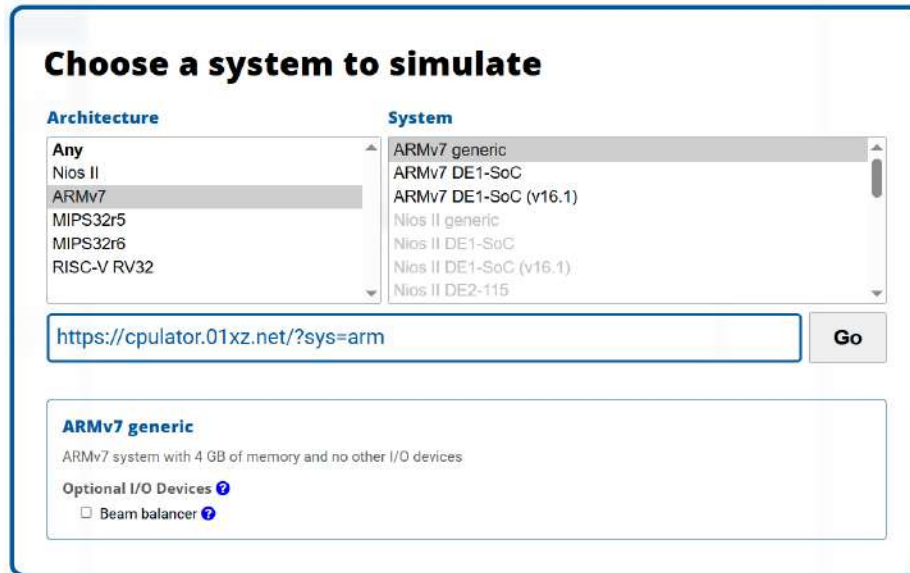


Figura 2. Selección; arquitectura ARM7

I. Descripción CPUlator

Una vez seleccionada la arquitectura, se presenta el entorno de trabajo.

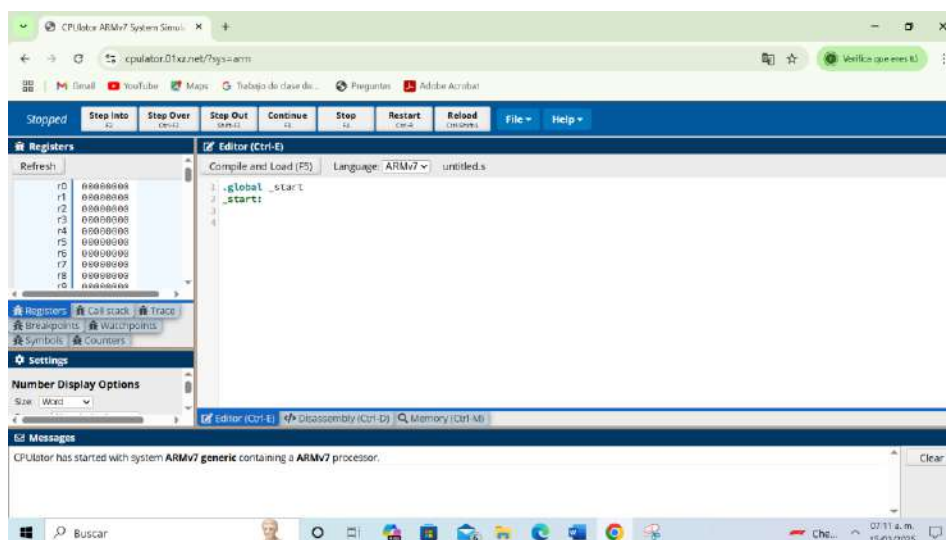


Figura 3. Entorno CPUlator

Se identifican las distintas zonas del entorno; entre los que se encuentran:

-
1. *Control de simulación.*
 2. *Área de edición.*
 3. *Área de desensamblado.*
 4. *Área de memoria.*
 5. *Área de registros.*
 6. *Área de configuración.*
 7. *Área de mensajes de salida.*
-

1.- Control de simulación. En la parte superior se encuentran los íconos para control del proceso de simulación y el manejo de archivos.



Figura 4. Control de simulación

2.- Área de edición. Ubicar en la zona del editor para escribir un programa, de manera automática aparece el formato del programa en ensamblador. El simulador no requiere la directiva **.text** (inicio del código); en caso de usar **datos** deberá incluir la zona de ellos, mediante **.data**.

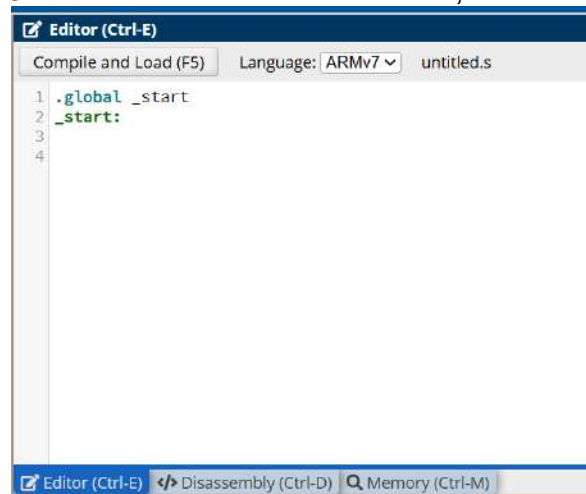


Figura 5. Editor CPUlator

3.- Área de desensamblado: Mostrará el programa en el formato original, la construcción en formato pseudoensamblador, las direcciones de memoria donde el entorno asignará a las instrucciones y los códigos de las instrucciones; además dará seguimiento al proceso de simulación.

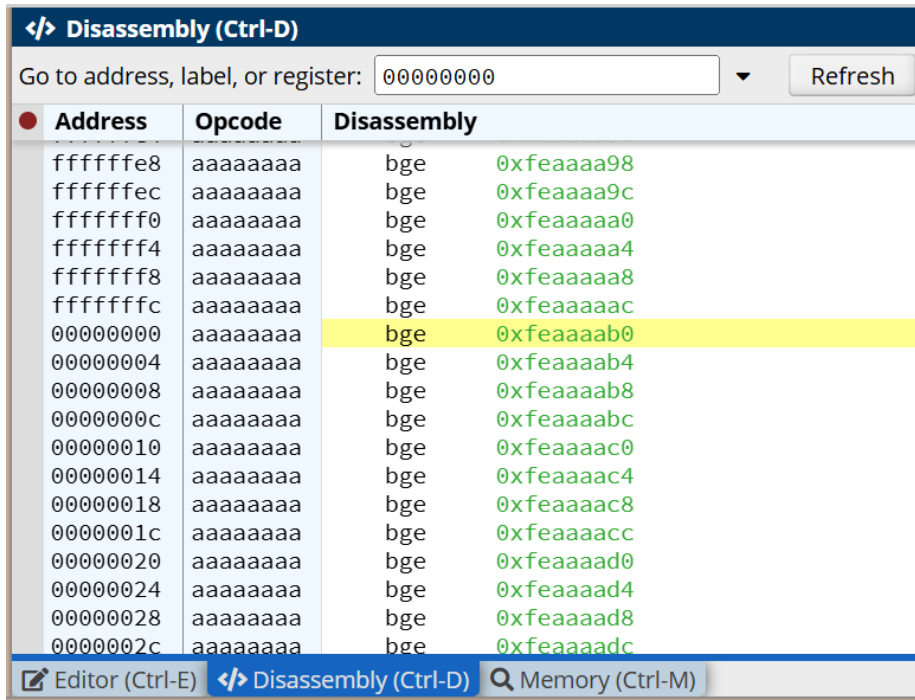


Figura 6. Ventana del desensamblador

5. **Área de Memoria.** Podrá ver y modificar el contenido de memoria.
- 6.

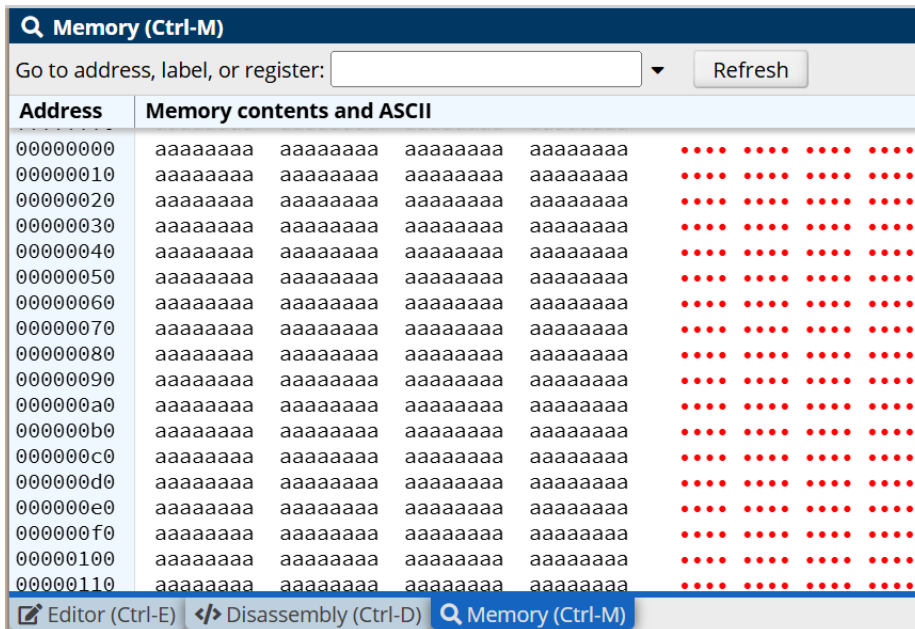


Figura 7. Despliegue y control de memoria

5.- Área de registros. Permite visualizar y modificar el contenido de los registros de la arquitectura.

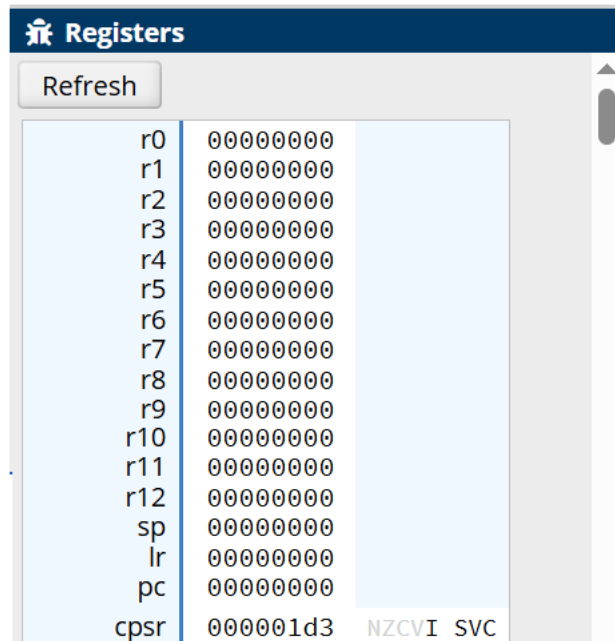


Figura 8. Registros

6.- Área de configuraciones. Configura opciones de despliegue, simulación y manejo de datos.

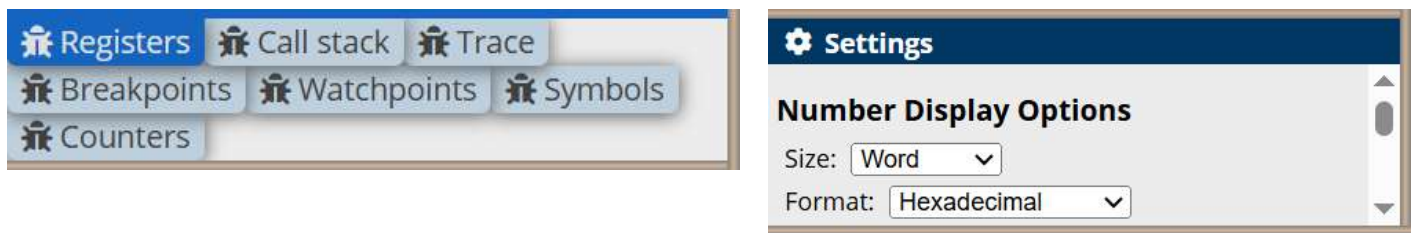


Figura 9. Configuración del IDE

7.- Zona de mensajes. Mostrará información del proceso de compilado.



Figura 10. Ventana de mensajes

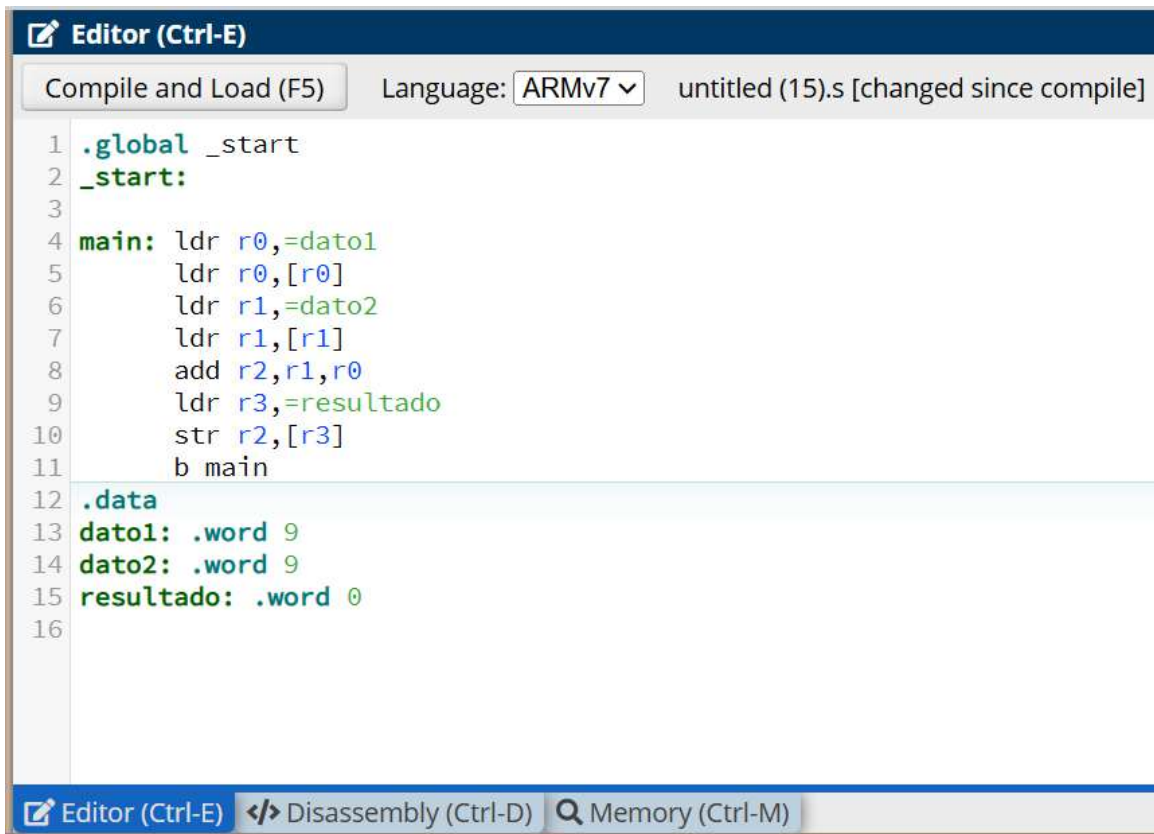
II. Uso de CPUlator.

El proceso consta de:

-
- A. Escritura del programa.
 - B. Compilar el programa.
 - C. Simular el programa.
 - D. Revisar la ejecución.
-

A. Escritura del programa.

Siguiendo la estructura del programa en ensamblador, usar el editor de CPUlator.



```
1  .global _start
2  _start:
3
4  main: ldr r0,=dato1
5        ldr r0,[r0]
6        ldr r1,=dato2
7        ldr r1,[r1]
8        add r2,r1,r0
9        ldr r3,=resultado
10       str r2,[r3]
11       b main
12
13  .data
14  dato1: .word 9
15  dato2: .word 9
16  resultado: .word 0
```

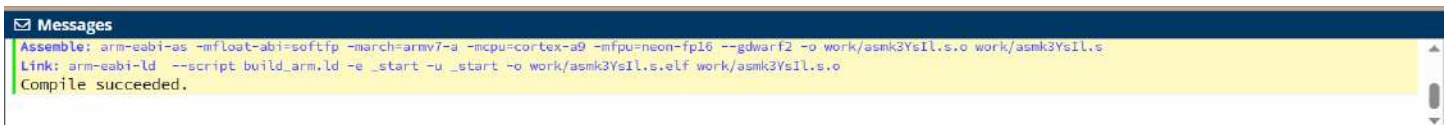
Figura 11. Programa formal

B. Compilar el programa.



Figura 10. Botón para compilar

Al finalizar el proceso de compilado, generará el estado de mismo.

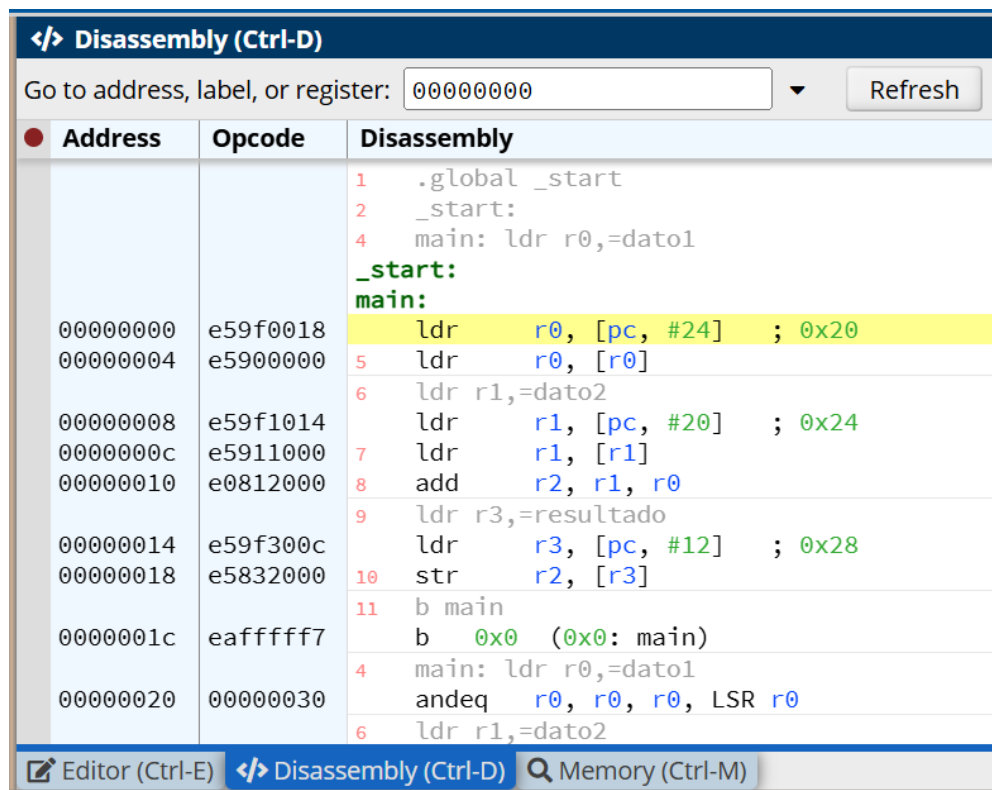


```

Messages
Assemble: arm-eabi-as -mfloat-abi=softfp -march=armv7-a -mcpu=cortex-a9 -mfp=neon-fp16 --gdwarf2 -o work/asmk3Ys1L.s.o work/asmk3Ys1L.s
Link: arm-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmk3Ys1L.s.elf work/asmk3Ys1L.s.o
Compile succeeded.
  
```

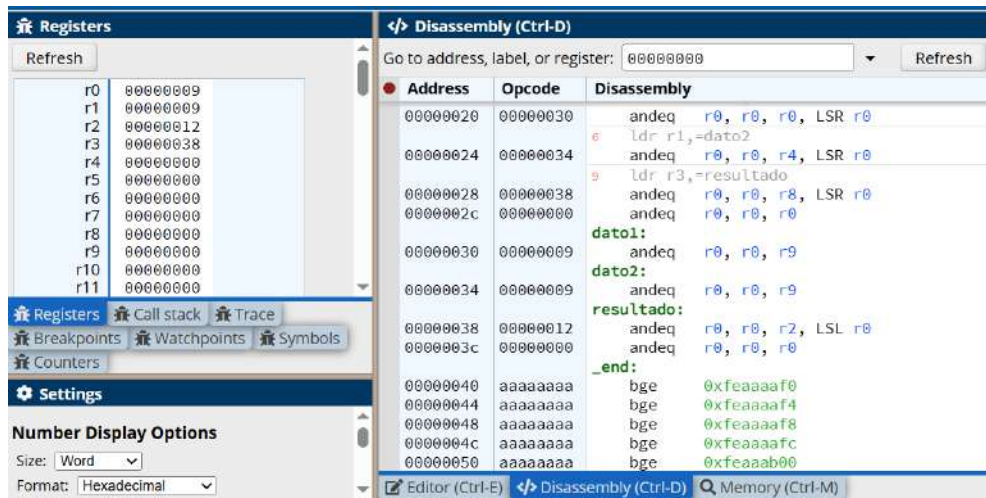
Figura 12. Mensaje final

De manera automática mostrará el programa en el área de desensamblado y quedará listo para iniciar la simulación.



Address	Opcode	Disassembly
		1 .global _start
		2 _start:
		4 main: ldr r0,=dato1
		_start:
		main:
00000000	e59f0018	ldr r0, [pc, #24] ; 0x20
00000004	e5900000	5 ldr r0, [r0]
		6 ldr r1,=dato2
00000008	e59f1014	ldr r1, [pc, #20] ; 0x24
0000000c	e5911000	7 ldr r1, [r1]
00000010	e0812000	8 add r2, r1, r0
		9 ldr r3,=resultado
00000014	e59f300c	ldr r3, [pc, #12] ; 0x28
00000018	e5832000	10 str r2, [r3]
		11 b main
0000001c	eafffff7	b 0x0 (0x0: main)
		4 main: ldr r0,=dato1
00000020	00000030	andeq r0, r0, r0, LSR r0
		6 ldr r1,=dato2

a. Área de código



b. Área de datos

Figura 13. Programa ensamblado

Así mismo podrá ver el contenido de memoria; se identifican los códigos de las instrucciones y el área de datos.

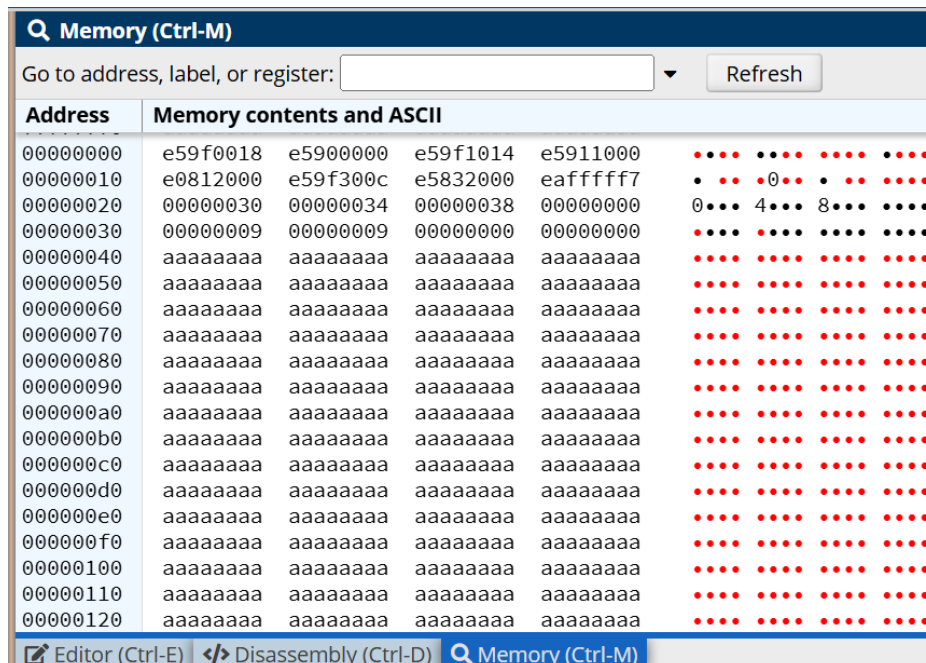



Figura 14. Contenido de memoria

C. Simulación.

Para iniciar el proceso de simulación presionar ; podrá ver la ejecución paso a paso, con lo que se comprobará el funcionamiento del programa.

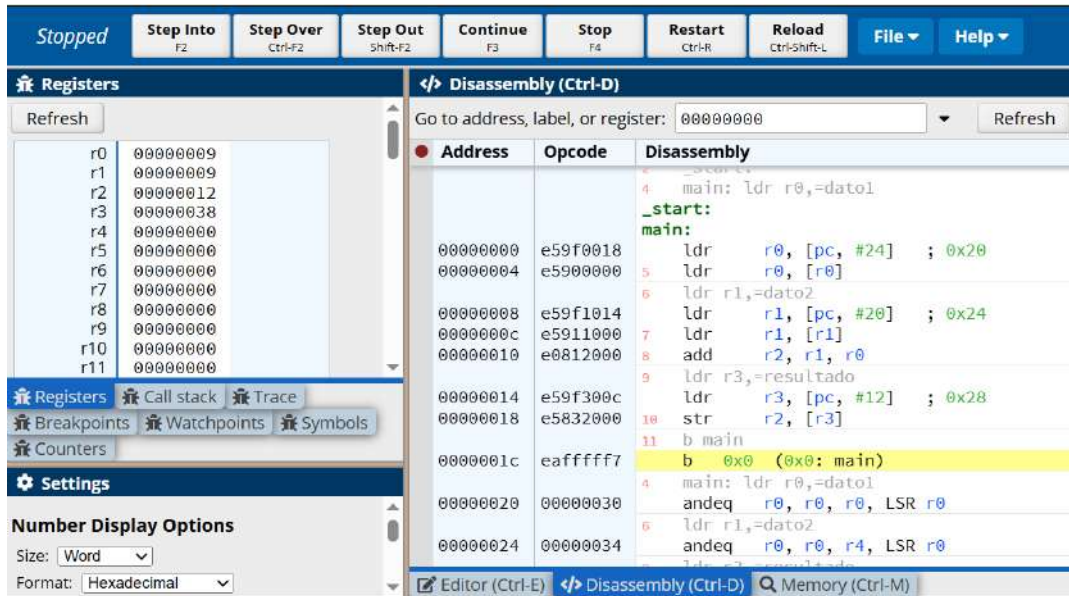


Figura 15. Ejecución del programa; actualización de registros.

D. Revisión de ejecución.

Es recomendable revisar el comportamiento del programa comprobando en memoria los resultados obtenidos.

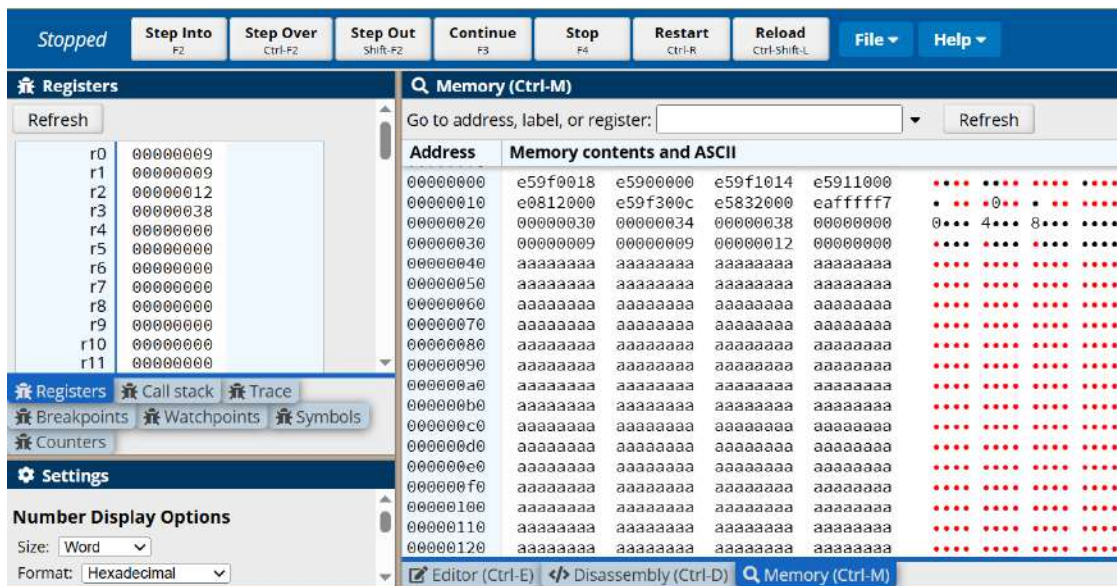


Figura 16. Ejecución de programa; revisión de memoria

7

Raspberry Pi Pico

Thonny - Micropython

Existen varias alternativas de programación de la plataforma Raspberry Pi Pico, entre ellas están C, C++, Arduino y Micropython.

Para la implementación del proyecto, se ha enfocado su desarrollo y propuesta empleando el IDE Thonny, programando en la versión de Python, pensada en el uso de sistemas diseñadas con microcontroladores; conocida como Micropython.

Este trabajo indica el procedimiento para:

-
- ❖ Descargar el software Thonny.
 - ❖ Instalación del IDE Thonny.
 - ❖ Configuración del entorno de Thonny.
 - ❖ Instalación del interprete para Micropython en la Raspberry Pi Pico.
 - ❖ Editar, guardar y ejecutar programas.
-

Descarga de Thonny.

Para descargar el software Thonny, acceder al sitio oficial:

<https://thonny.org/>

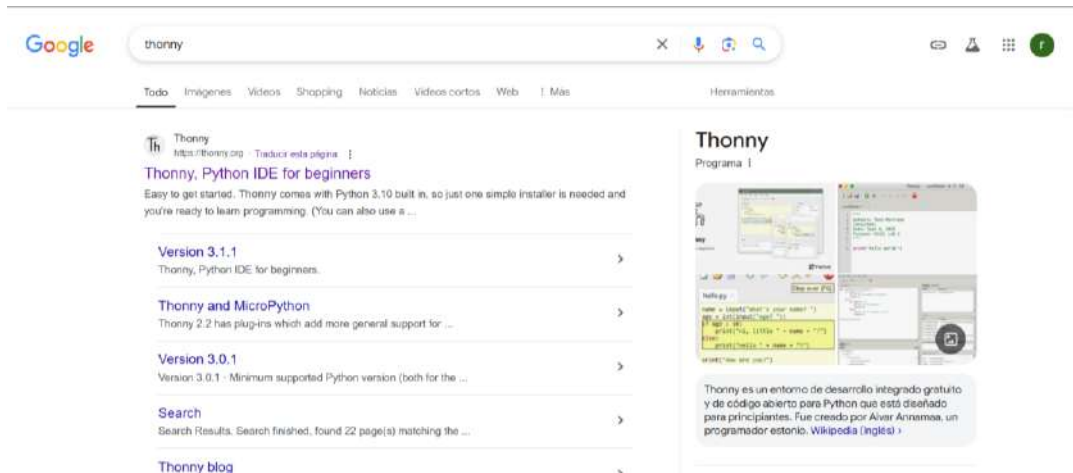


Figura 1. Ubicar el sitio de descarga de Thonny

Requiere especificar la versión del sistema operativo donde será instalado.

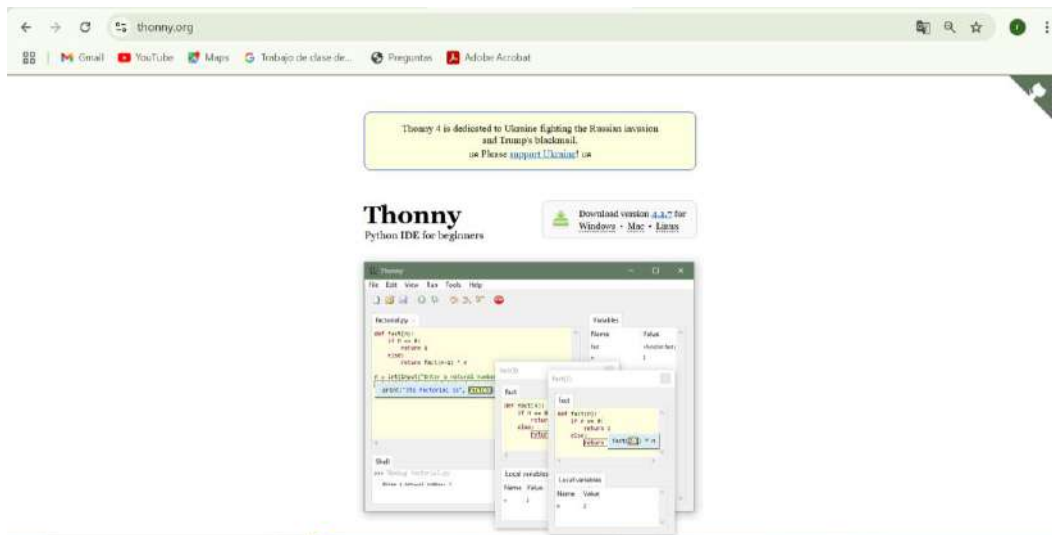


Figura 2. Selección del Sistema Operativo

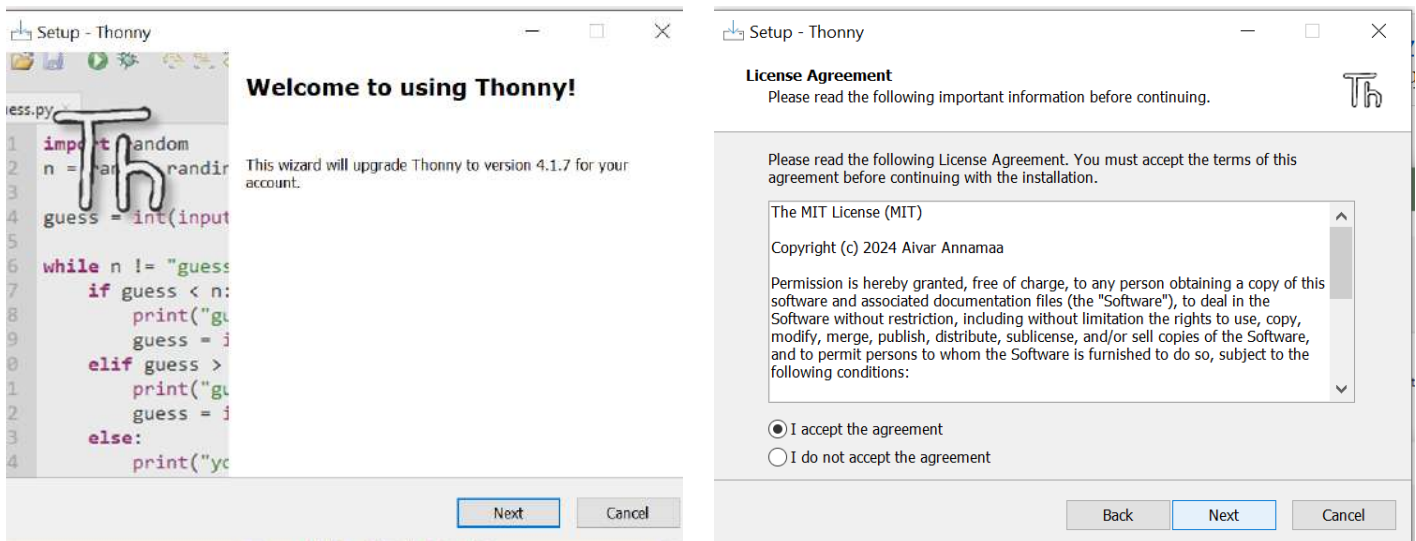
Para cada sistema operativo existen diversas versiones, por lo que deberá elegir la requerida por su equipo de cómputo.



Figura 3. Selección de la versión de Thonny

Instalación de Thonny.

Una vez descargado, proceder a instalar (thonny-4.1.7.exe o la más actual disponible); seguir el proceso de instalación.



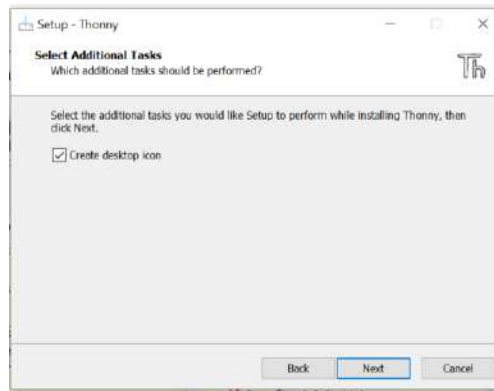


Figura 4. Proceso de instalación

Uso y configuración de Thonny.

Al ejecutar Thonny, aparecerá el entorno de trabajo; en donde, se podrá identificar:

- ❖ *El área de comandos, para manejo de archivos, configuraciones, botones de ejecución y paro del programa en curso.*
- ❖ *El área de edición, para escritura del código.*
- ❖ *Consola de Thonny, para ejecución en línea de comandos o despliegue de resultados e información.*

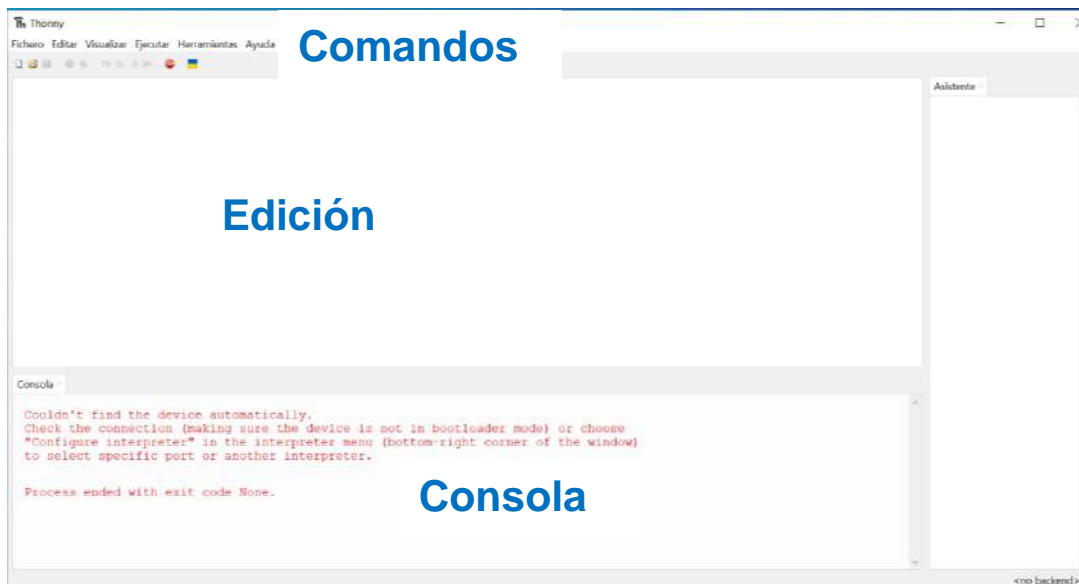


Figura 5. Entorno Thonny

Configuración del interprete.

Seleccionar *MicroPython (Raspberry Pi Pico)*, en el *Interprete* dentro de las opciones de *Thonny* (figuras 6 y 7); en esta etapa no es requerida conectar la tarjeta a la computadora.

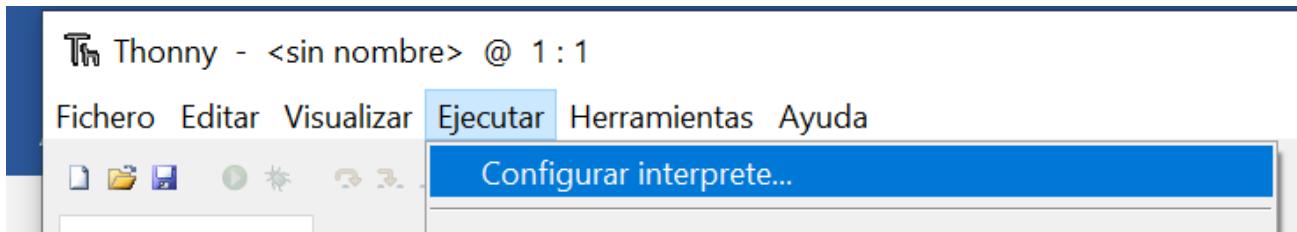


Figura 6. Configuración del interprete

Mantener la detección automática del puerto o REPL.

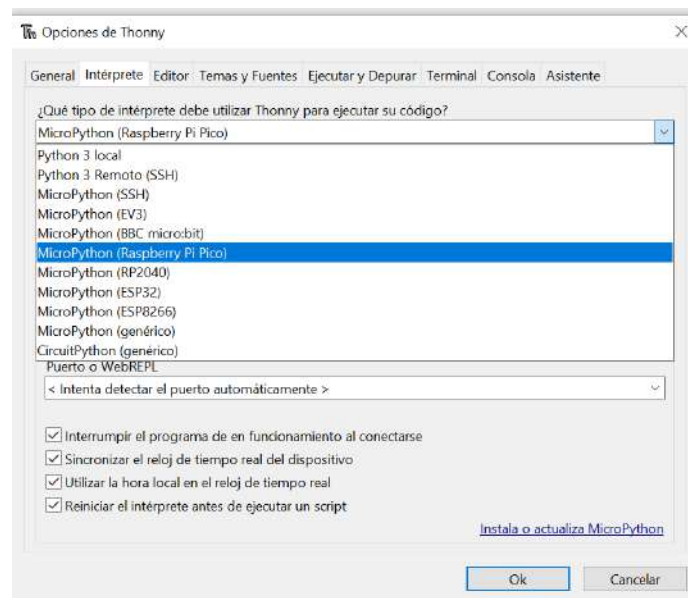


Figura 7. Selección Micropython

Instalación del interprete Micropython.

Otro paso requerido es la instalación del interprete de Micropython en la tarjeta Raspberry Pi Pico; esto se puede realizar mediante dos opciones.

-
- ❖ *Dentro del IDE Thonny*
 - ❖ *Copiando el interprete directamente a la Raspberry Pi Pico.*
-

Desde Thonny.

Conectar la tarjeta Raspberry Pi Pico a la computadora a través del cable USB-Micro_USB-B; en la ventana de configuración de Thonny y del puerto COM, del lado inferior – derecho, encontrará [Instala o actualiza MicroPython](#).

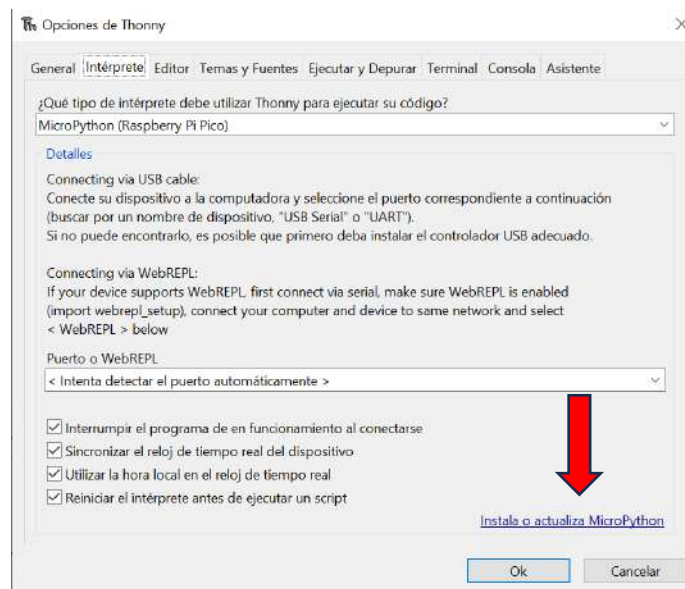


Figura 8. Instalar o actualizar Micropython

Seleccionar la versión de la tarjeta Raspberry Pico y las configuraciones requeridas.

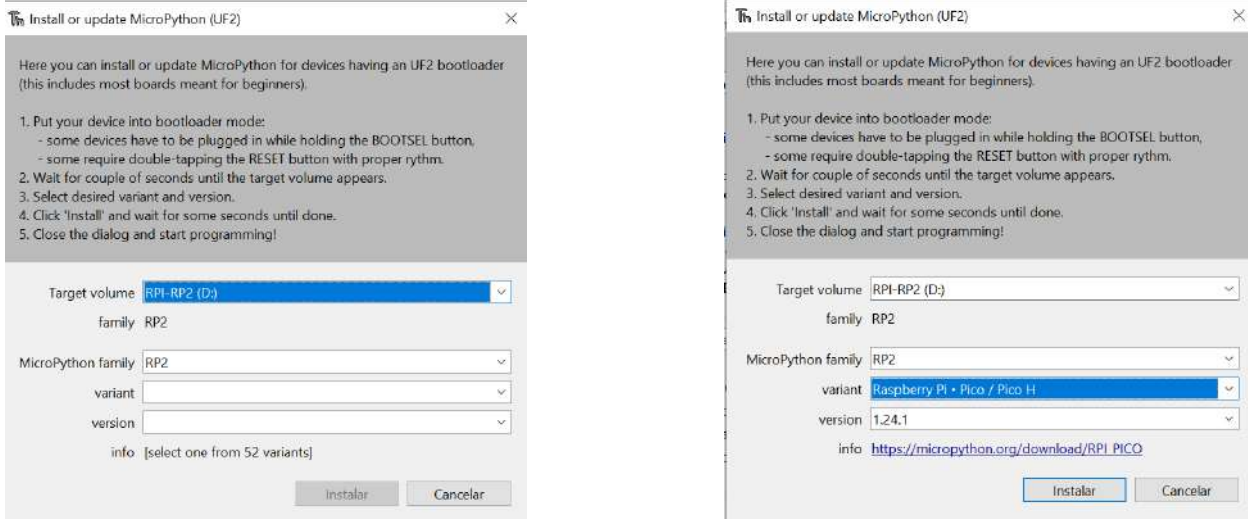


Figura 9. Instalación de Micropython

En caso de no existir algún problema, indicará la conexión correcta, mediante el despliegue de la figura 10; la elección del puerto COM se ha realizado de manera automática.

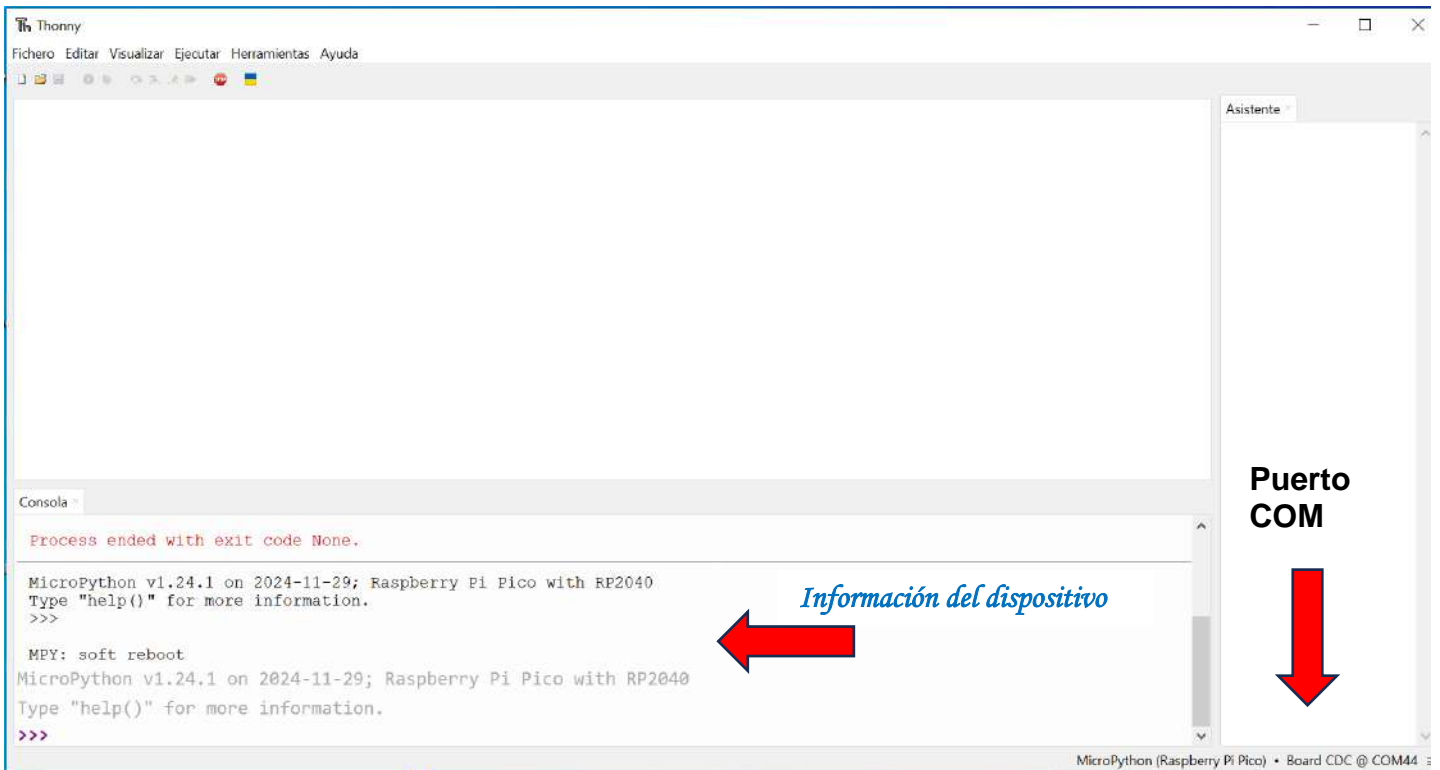


Figura 10. Conexión completa

Transfiriendo RPI_PICO_20241129_v1.24.1.uf2 o la versión más reciente.

Descargar el programa correspondiente a la versión de su plataforma (Pico, Pico W, Pico 2 o Pico 2W); para la primera opción: RPI_PICO_20241129_v1.24.1.uf2, del enlace:

<https://www.raspberrypi.com/documentation/microcontrollers/micropython.html>

También puede descargar de:

https://micropython.org/download/RPI_PICO/

Para este procedimiento requiere de tres pasos.

-
- ❖ Presionar el botón **BOOTSEL** de Raspberry P Pico.

 - ❖ Conectar el cable USB–Micro_USB-B entre la computadora y Raspberry Pi Pico (sin dejar de presionar el botón **BOOTSEL**), la tarjeta es reconocida como una unidad de memoria, en este momento podrá liberar el botón **BOOTSEL** (figura 11).

 - ❖ Copiar el archivo RPI_PICO_20241129_v1.24.1.uf2 a Raspberry Pi Pico, la tarjeta queda lista para programar con Micropython; dejará de mostrarse como unidad de almacenamiento (figura 12).
-



Figura 11. Detección de Raspberry Pi Pico

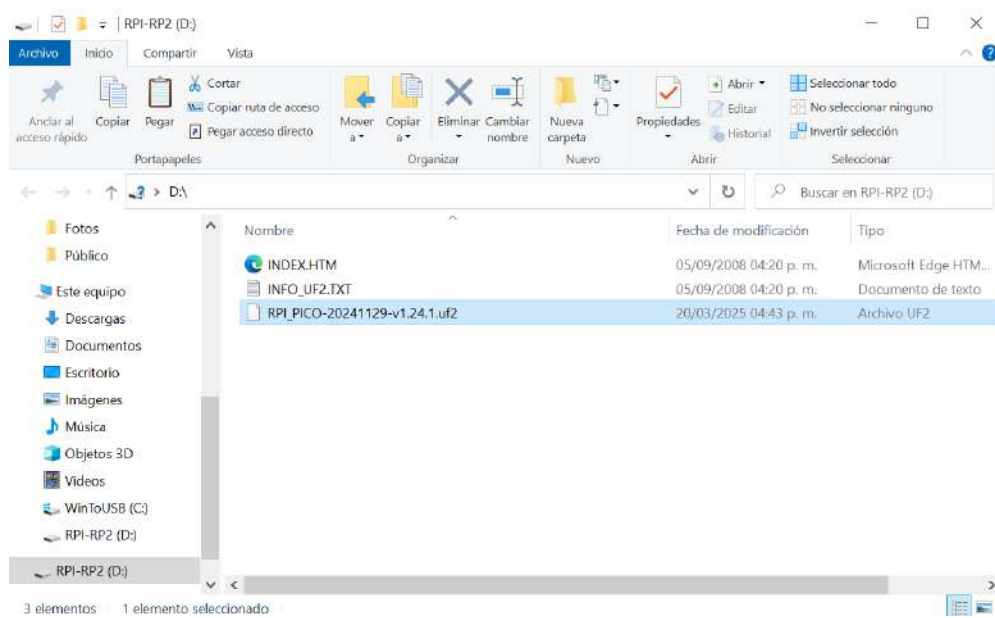


Figura 12. Interprete de Micropython instalado

Editar, guardar y ejecutar un programa.

Editar código.

Considerando la estructura de programación en Python, escribir el programa en la zona designada.

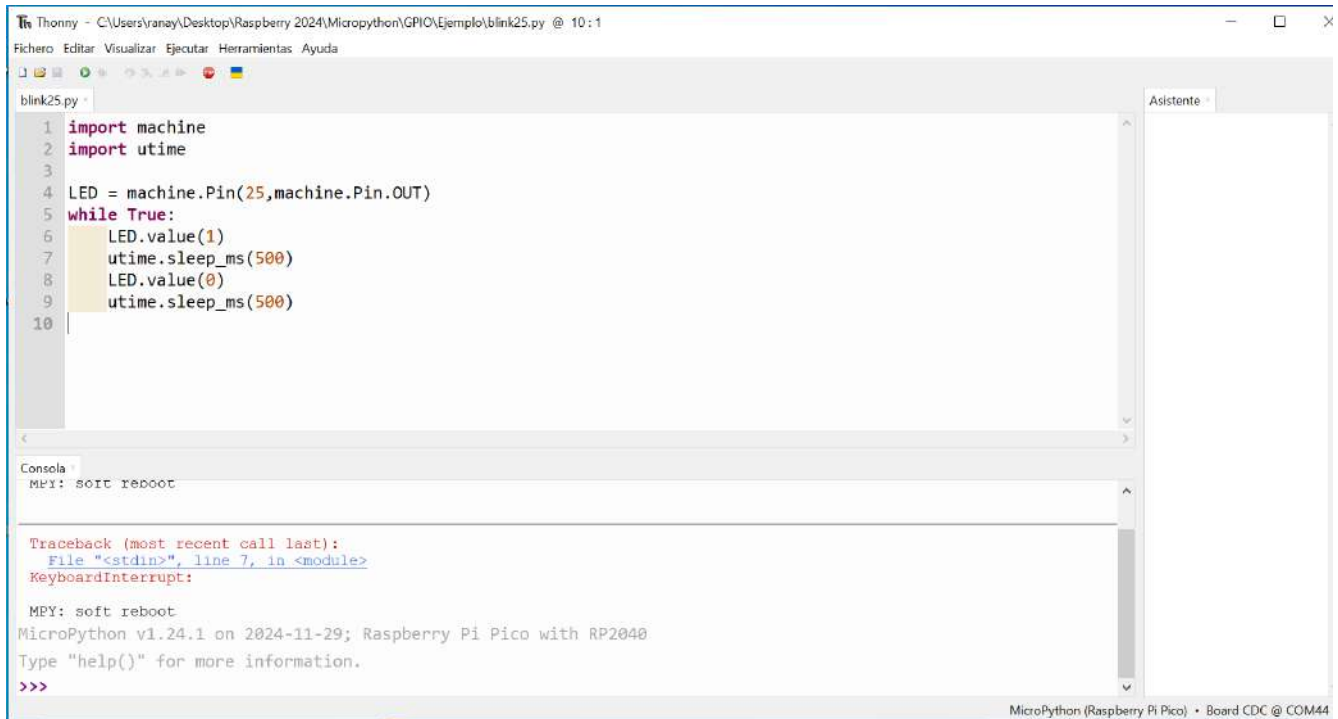


Figura 13. Edición de código en Micropython

Guardar programa.

Tendrá dos opciones a elegir:

-
- 1. En la computadora.**
 - 2. En la Raspberry Pi Pico.**
-

Seleccionar: **Guardar como.**

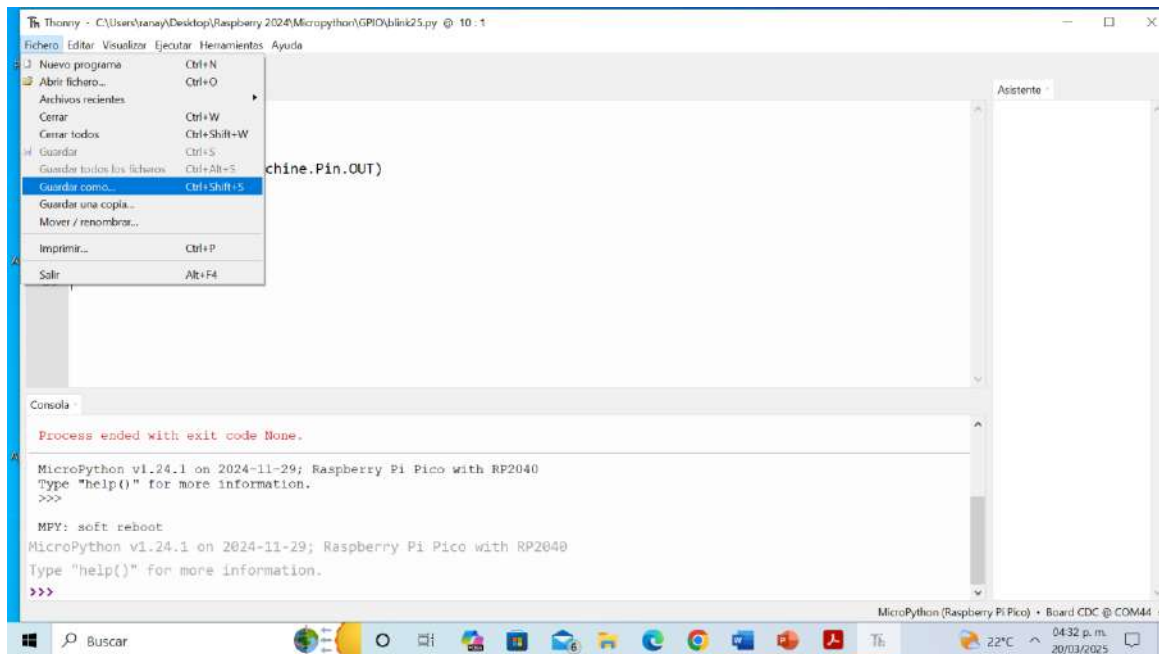


Figura 14. Salvar programa actual

- I. **Este computador;** lo podrá guardar en alguna ubicación dentro de su computadora con el nombre de su elección **NombrePrograma.py**; este dejará de correr en cuanto lo detenga o se salga de Thonny (figura 15).
- II. **Salvar en Raspberry Pi Pico.** Permite dejar permanente el programa, debido a que se almacena en memoria de programa ROM (flash), el nombre que debe tener es **main.py**; de manera que es lo primero que ejecutará la Raspberry Pi Pico (figuras 16 y 17).

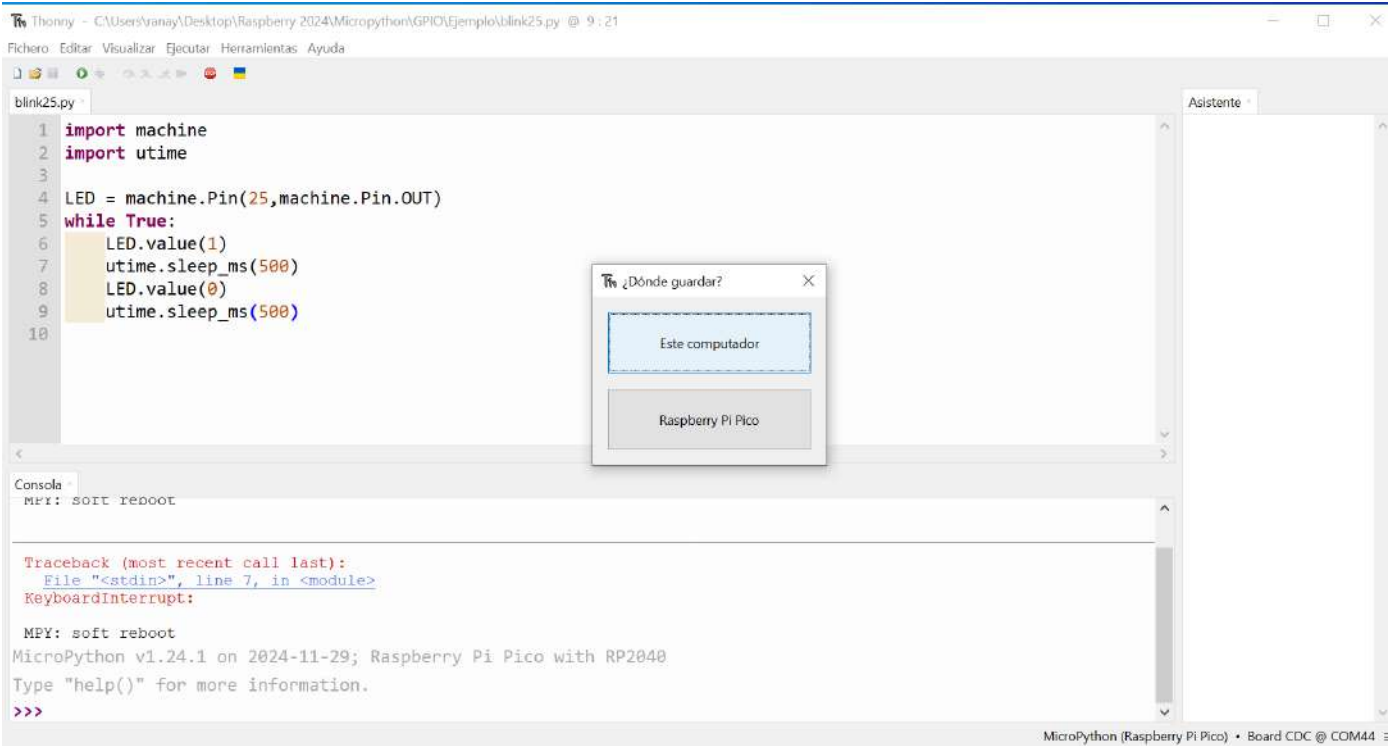


Figura 15. Salvar en computadora personal

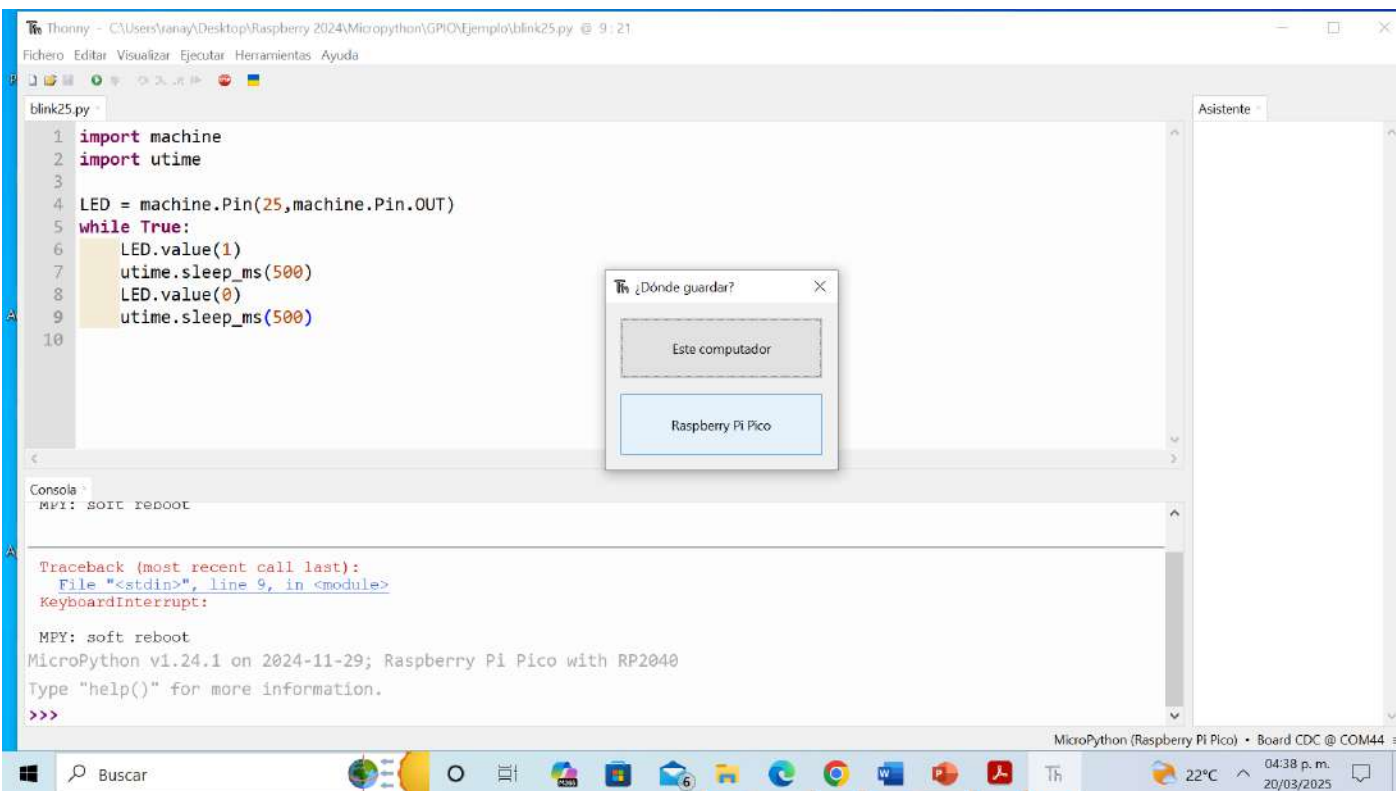
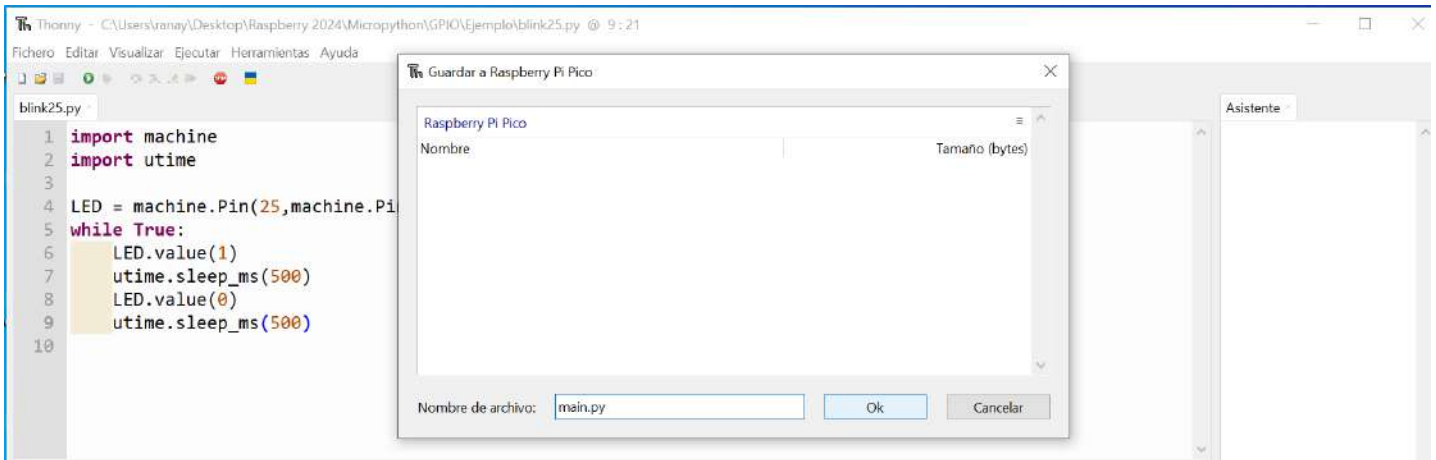


Figura 16. Salvar en la Raspberry Pi Pico

Figura 17. Nombrar el código *main.py*

Ejecutar un programa.



Presionar el icono  (Ejecutar el script actual F5) iniciar o  (Detener/Reiniciar back-end Ctrl-F2) para suspender la ejecución o reconectar; en caso de haber guardado en la Raspberry Pi Pico, será requerido ejecutar desde Thonny la primera vez que lo descargue, posteriormente al salir del IDE o al conectar a alimentación, el programa se ejecutará de manera automática.



Figura 18. Ejecutar o detener un programa


En caso de reconectar el cable al entrar a Thonny, e iniciar la ejecución en la Raspberry Pi Pico, o al haber conectado la plataforma con el IDE Thonny ya abierto, sin tener programa en la plataforma; bastará con presionar el botón  para restablecer la conexión.



Figura 19. Restablecer conexión con Raspberry Pi Pico

Una vez reconectando, la plataforma quedará lista para continuar ejecutando programas nuevos.



Figura 20. Resultado de reconexión

8

Programación de App Inventor

Comunicación Bluetooth

Transmisión 'A', 'D', 'T', 'T', 'S'

APLICACIÓN PARA CONTROLAR DOS MOTORES DE CORRIENTE DIRECTA POR MEDIO DE BLUETOOTH.

OBJETIVO GENERAL: Realizar el software y hardware para comunicar y controlar dos motores de corriente directa, que semejarán la estructura de un robot móvil de tipo diferencial de manera inalámbrica, utilizando Bluetooth.

OBJETIVOS PARTICULARES: Se debe realizar la aplicación APK que correrá en el teléfono celular y transmitirá comandos a través del Bluetooth, estos serán recibidos por otra interfaz Bluetooth, que enviará a Raspberry Pi Pico; recibirá los comandos y generará las acciones pertinentes para generar los movimientos deseados en los motores de cd. Las tablas 1 y dos describen la asignación de señales de control que el microcontrolador generará de acuerdo al comando recibido y las acciones a ejecutar respectivamente.

I. Hardware

El circuito electrónico es el siguiente (figura 1).

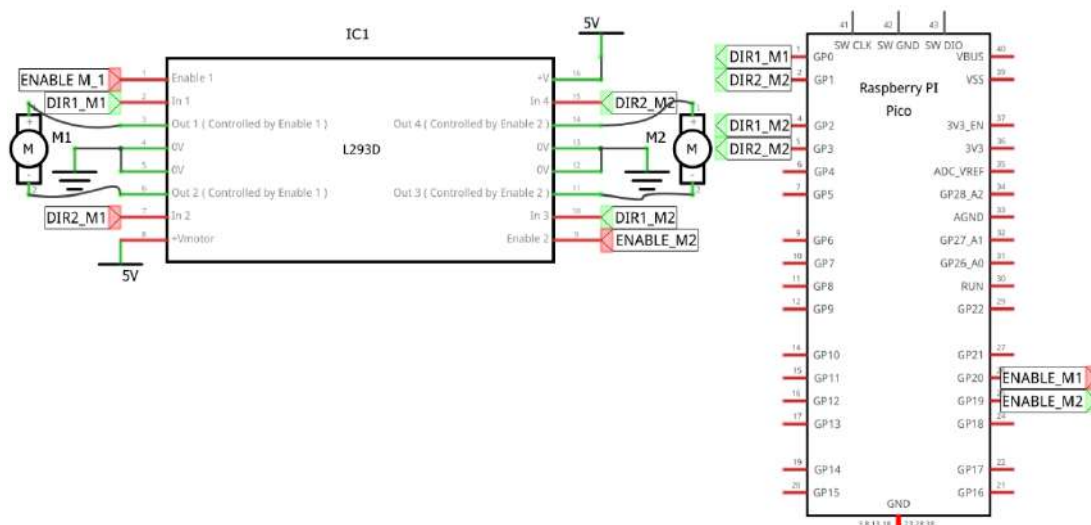


Figura 1. Circuito de control de dos motores de CD

Motor 1			Motor 2			
GPIO0	GPIO1	GPIO20	GPIO2	GPIO3	GPIO19	
DIR1_M1	DIR2_M1	EN_M1	DIR1_M2	DIR2_M2	EN_M2	

Tabla 1. Asignación de señales de control de los motores de CD

Comando Puerto serie	ACCION	
	MOTOR M1	MOTOR M2
'S'	PARO	PARO
'A'	DERECHA	DERECHA
'T'	IZQUIERDA	IZQUIERDA
'D'	DERECHA	IZQUIERDA
'I'	IZQUIERDA	DERECHA

Tabla 2. Control de motores, comunicación serie

II. Aplicación APK

Se ha utilizado el software App Inventor del MIT para construir y programar la aplicación, el diseño de la aplicación se muestra en la figura 2.

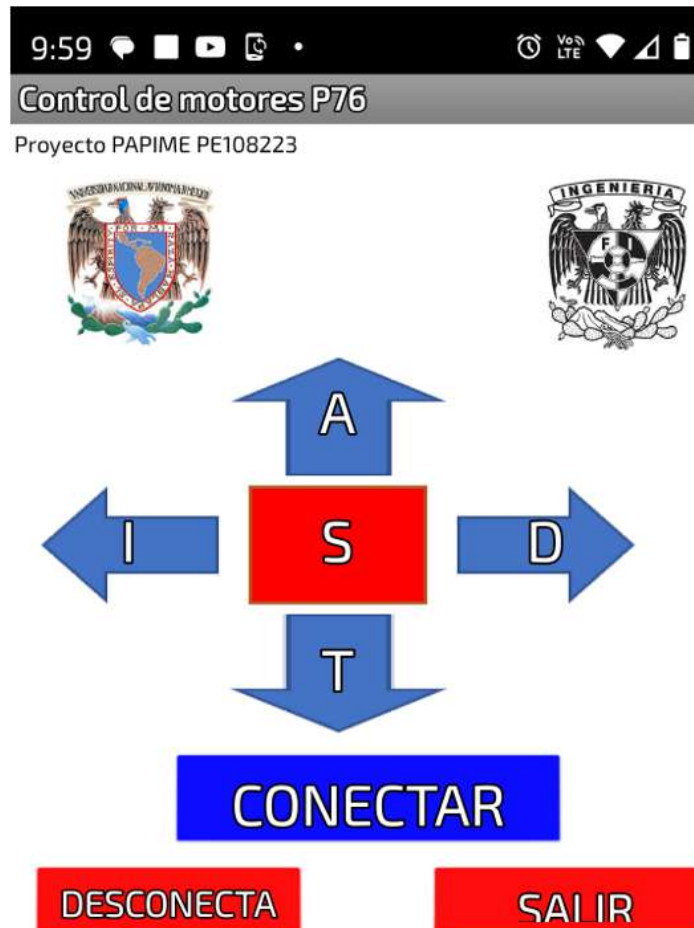


Figura 2. Interfaz de la aplicación

Una vez hecha la interfaz, se inicia la programación de los diferentes elementos que la integran, como son los botones, buscar los dispositivos Bluetooth cercanos al teléfono celular, conectarse al Bluetooth deseado, desconectarse y salir de la aplicación.

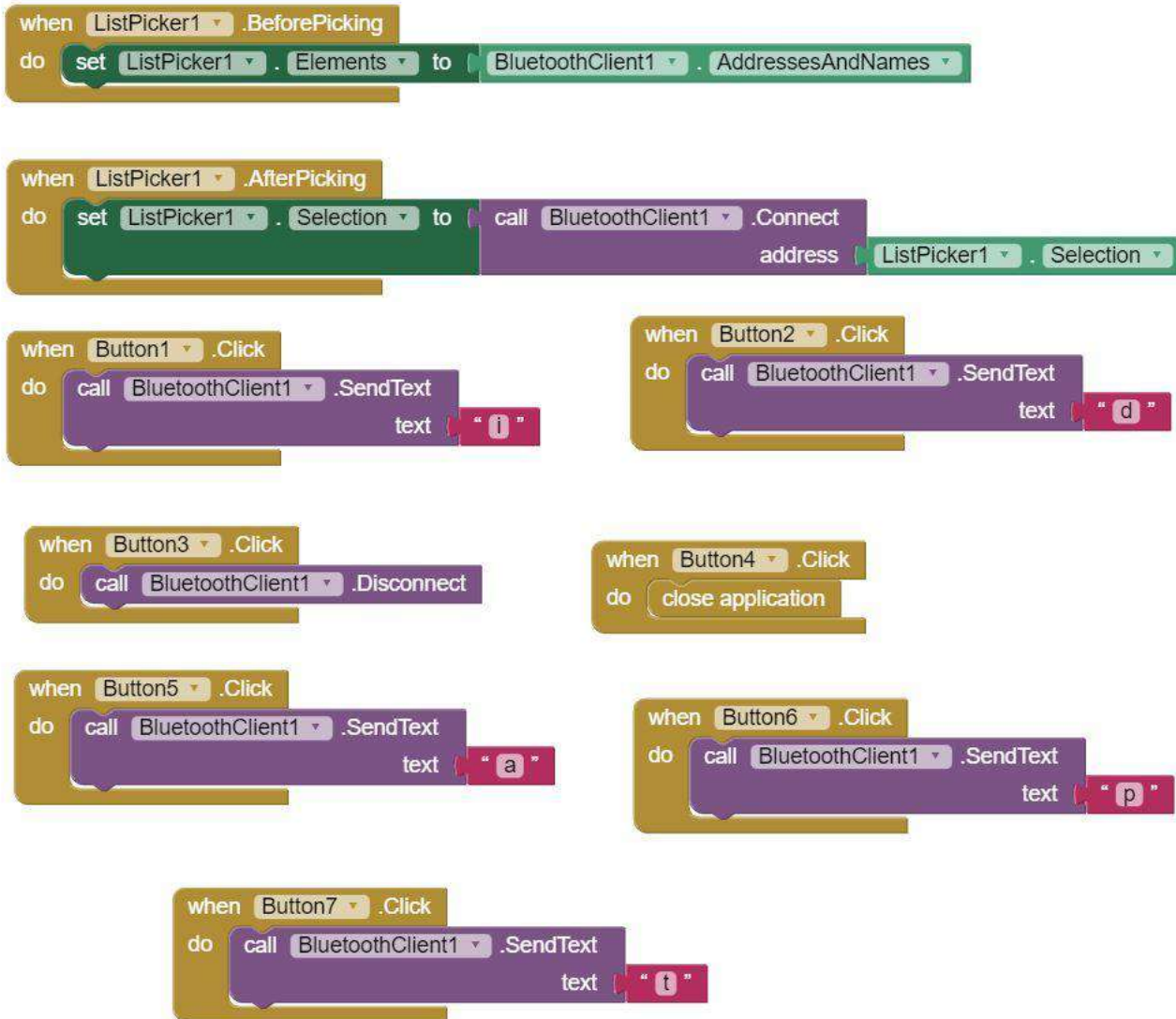


Figura 3. Programación de la aplicación

Se proporcionará a los alumnos la liga para descargar la aplicación APK para proceder a instalar y ejecutar para iniciar con el control vía Bluetooth, una vez instalada despliega la pantalla de la figura 4.

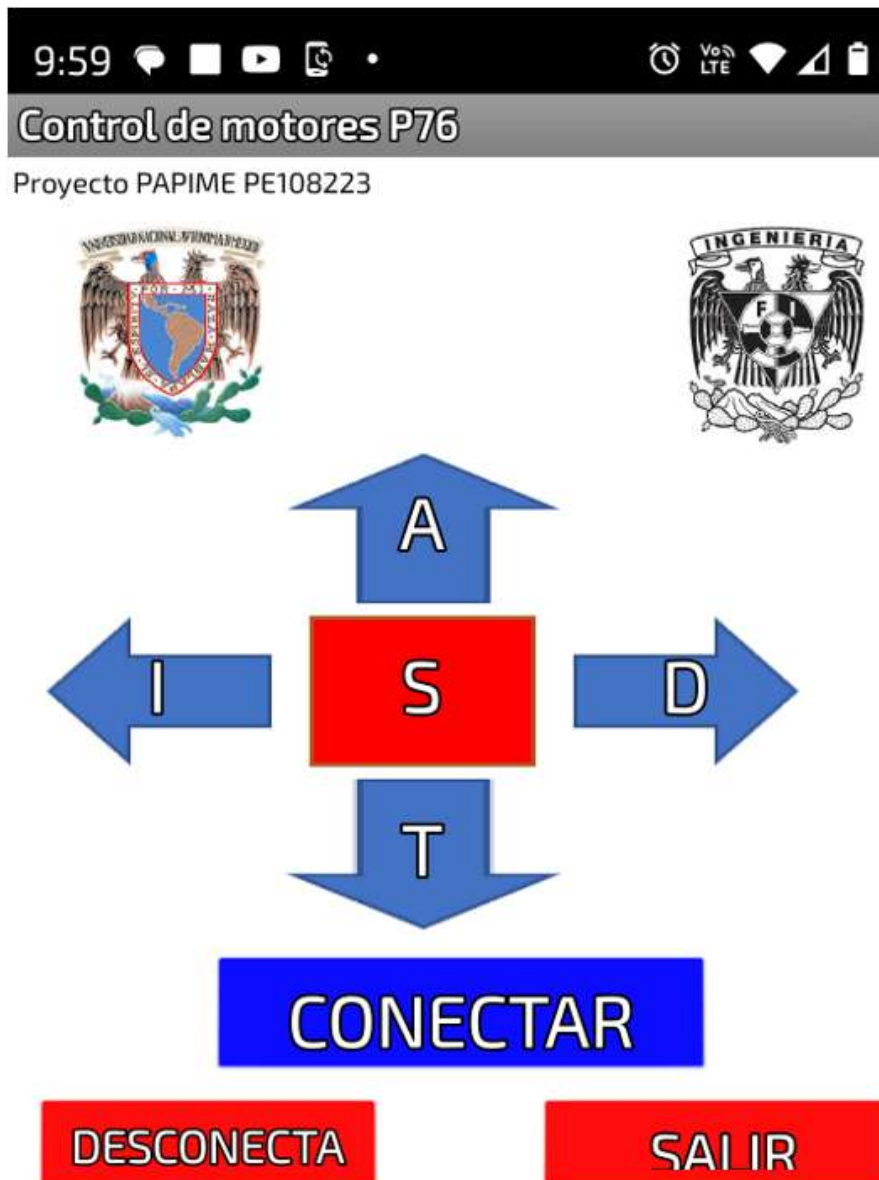


Figura 4 Aplicación APK

III. Software

El programa que correrá en el microcontrolador será resuelto por el alumno(a), deberá tener como requisito configurar la comunicación a 9600 BAUDS y recibir los valores ASCII de los caracteres A, D, I, T, S en mayúsculas para generar el control.

9

Raspberry Pi Pico

Control remoto de giro de motor a pasos

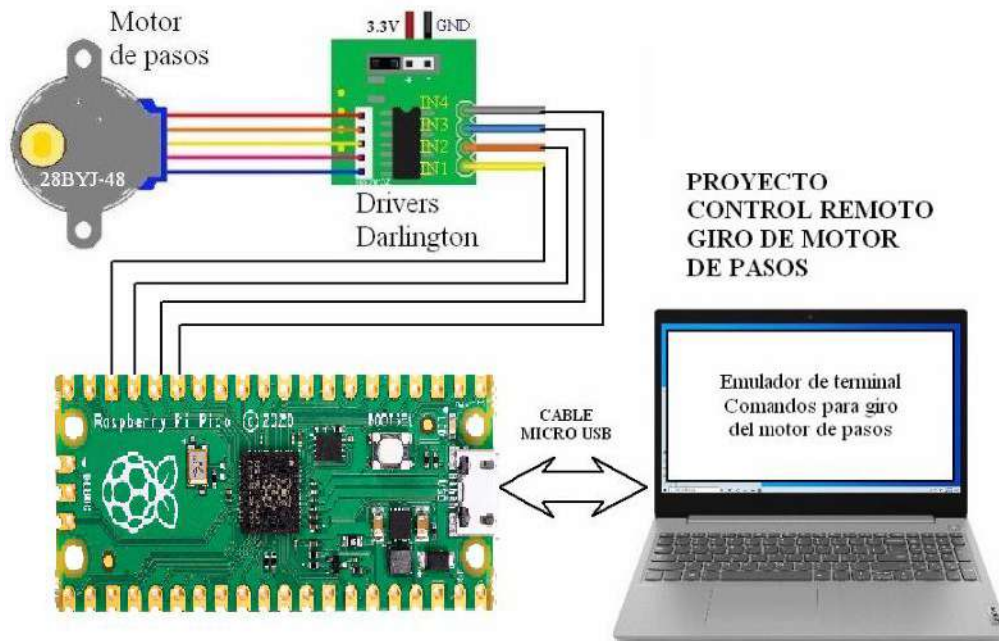


Figura 1. dispositivos para implementación del proyecto

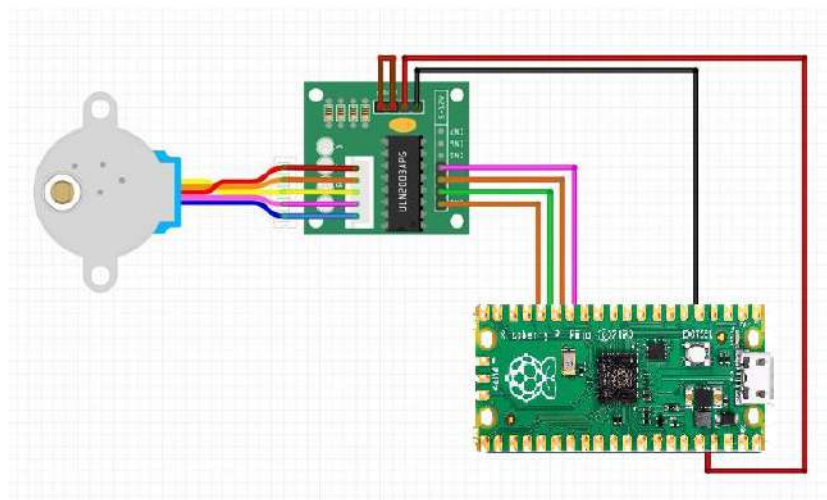


Figura 2. Diagrama de bloques del proyecto

		STEPS							
METHODS	PHASES	1	2	3	4	5	6	7	8
WAVE DRIVE One phase at a time Simplest, but least used	BLUE	1				1			
	PINK		1				1		
	YELLOW			1				1	
	ORANGE				1				1
FULL STEP Two phases at a time Strongest Torque	BLUE	1			1	1			1
	PINK	1	1			1	1		
	YELLOW		1	1			1	1	
	ORANGE			1	1			1	1
HALF STEP One, or two phases at a time Smallest step angle Medium torque	BLUE	1	1						1
	PINK		1	1	1				
	YELLOW			1	1	1	1		
	ORANGE						1	1	1

Figura 3. el motor de pasos tiene varios métodos para operarlo. En la tabla se detalla cómo deben activarse las fases para obtener resultados distintos en cuanto al torque y el ángulo de giro por paso, dependiendo del método utilizado

Introducción

Raspberry pi pico w integra el protocolo TCP-IP y tiene conectividad Wifi, lo cual permite su conexión a internet a través de un router cercano al módulo, con la posibilidad de control remoto del tipo "Anywhere", desde cualquier parte del mundo. Esto significa que, ya sea a través de un teléfono celular ó una computadora conectados a internet, se pueden enviar comandos al microcontrolador para activar dispositivos y recibir información de sensores ó alarmas instaladas en casas y oficinas.

Objetivos

El proyecto muestra la manera de realizar el control remoto de un motor de pasos 28BYJ-48, desde una PC y con un programa emulador de terminal, con comandos enviados al microcontrolador Raspberry pi pico.

A través de un programa desarrollado en lenguaje Micropython, recibe los comandos y opera un motor de pasos, el cual gira el número de grados indicados por el usuario de la terminal.

Se utilizan herramientas de software gratuitas, como son el firmware Micropython, el software IDE llamado uPyCraft, y el emulador de terminal Termite v3.4. Igualmente los componentes del proyecto.

Descripción

- Con base en las figuras 1 y 2, se muestran los diagramas de conexión del proyecto que utiliza componentes de bajo costo.
- Para desarrollar el proyecto, además del hardware indicado, se requiere primeramente cargar en la PC el driver para la interfaz CH340G. Al conectar la PC con el módulo, por cable MicroUSB, en el administrador de dispositivos deberá darse de alta un nuevo puerto serial COM, que será usado por las herramientas de software posteriormente utilizadas.
- Debe recordarse que el lenguaje Micropython funciona a través de un firmware intérprete que debe ser cargado previamente al microcontrolador. Posteriormente, por medio de un emulador de terminal llamado "Termite v3.4", se envían los comandos al microcontrolador de tal manera de indicar el número de grados de giro del motor, en ambos sentidos (clockwise, counterclockwise).
- Siguiendo el diagrama de conexiones mostrado en las figuras 1 y 2, el microcontrolador se conecta con un cable Micro USB a la computadora PC y al motor de pasos por medio de cables dupont. Se observa como el motor se alimenta desde el pin de 3.3v ó de 5v del microcontrolador.
- Para la comunicación inicial con el NodeMCU y el almacenamiento del programa de aplicación, se utiliza la herramienta ya mencionada uPyCraft; main.py, es un programa de prueba;

Hace giros del motor en un sentido y el contrario. El programa se carga desde uPyCraft y el funcionamiento es automático ó después del reset. No requiere el uso del emulador Termite.

motorGradosSerial-2 éste es el programa de aplicación. Debe primeramente cargarse en el microcontrolador desde uPyCraft. Una vez cargado, se cierra uPyCraft y abrimos el emulador de terminal "Termite v3.4" . Para escribir el comando para ejecutar el programa, debe copiar y pegar en la línea de comandos de Termite lo siguiente: `exec(open('motorGradosSerial-2.py').read(),globals())`

Código fuente:

```

from machine import Pin
from time import sleep

pin1 = Pin(16, Pin.OUT)    #PINES DE CONEXION DEL MODULO ESP8266 CON EL DRIVER ULN2003
pin2 = Pin(5, Pin.OUT)
pin3 = Pin(4, Pin.OUT)
pin4 = Pin(0, Pin.OUT)

tiempo = 0.005
#tiempo = 0.5

WaveDrive=[
    [1,0,0,0],
    [0,1,0,0],
    [0,0,1,0],
    [0,0,0,1]]           #SECUENCIA DE ONDA

FullStepDrive=[
    [1,0,0,1],
    [1,1,0,0],
    [0,1,1,0],
    [0,0,1,1]]           #SECUENCIA DE PASO COMPLETO

```

```

HalfStepDrive=[
  [1,0,0,1],
  [1,0,0,0],
  [1,1,0,0],
  [0,1,0,0],
  [0,1,1,0],
  [0,0,1,0],
  [0,0,1,1],
  [0,0,0,1]]          #SECUENCIA DE MEDIO PASO

#Secuencias sentido de giro HORARIO
#=====

def secuenciaWaveDrive_horario(tiempo):
    for indice in range(2048):
        pin1.value(WaveDrive[indice%4][3])
        pin2.value(WaveDrive[indice%4][2])
        pin3.value(WaveDrive[indice%4][1])
        pin4.value(WaveDrive[indice%4][0])
        sleep(tiempo)

def secuenciaFullStepDrive_horario(tiempo):
    for indice in range(2048):
        pin1.value(FullStepDrive[indice%4][3])
        pin2.value(FullStepDrive[indice%4][2])
        pin3.value(FullStepDrive[indice%4][1])
        pin4.value(FullStepDrive[indice%4][0])
        sleep(tiempo)

def secuenciaHalfStepDrive_horario(tiempo):
    for indice in range(4096):
        pin1.value(HalfStepDrive[indice%8][3])
        pin2.value(HalfStepDrive[indice%8][2])
        pin3.value(HalfStepDrive[indice%8][1])
        pin4.value(HalfStepDrive[indice%8][0])
        sleep(tiempo)

#Secuencias sentido de giro AHORARIO
#=====

def secuenciaWaveDrive_Ahorario(tiempo):
    for indice in range(2048):
        pin1.value(WaveDrive[indice%4][0])
        pin2.value(WaveDrive[indice%4][1])
        pin3.value(WaveDrive[indice%4][2])
        pin4.value(WaveDrive[indice%4][3])
        sleep(tiempo)

def secuenciaFullStepDrive_Ahorario(tiempo):
    for indice in range(2048):
        pin1.value(FullStepDrive[indice%4][0])
        pin2.value(FullStepDrive[indice%4][1])
        pin3.value(FullStepDrive[indice%4][2])
        pin4.value(FullStepDrive[indice%4][3])
        sleep(tiempo)

def secuenciaHalfStepDrive_Ahorario(tiempo):
    for indice in range(4096):
        pin1.value(HalfStepDrive[indice%8][0])

```

```
pin2.value(HalfStepDrive[indice%8][1])
pin3.value(HalfStepDrive[indice%8][2])
pin4.value(HalfStepDrive[indice%8][3])
sleep(tiempo)

#=====
def secuenciaParo():
    pin1.value(0)
    pin2.value(0)
    pin3.value(0)
    pin4.value(0)
    sleep(tiempo)

#=====
def main():
    secuenciaWaveDrive_horario(tiempo)
    secuenciaWaveDrive_Ahorario(tiempo)    #GIRO COMPLETO

    secuenciaFullStepDrive_horario(tiempo)
    secuenciaFullStepDrive_Ahorario(tiempo)    #GIRO COMPLETO

    secuenciaHalfStepDrive_horario(tiempo/2)
    secuenciaHalfStepDrive_Ahorario(tiempo/2)    #GIRO COMPLETO

    secuenciaParo()

#=====
main()

#=====
```

10

Raspberry Pi Pico

Control manual de posición angular en un motor a pasos, mediante un potenciómetro de precisión

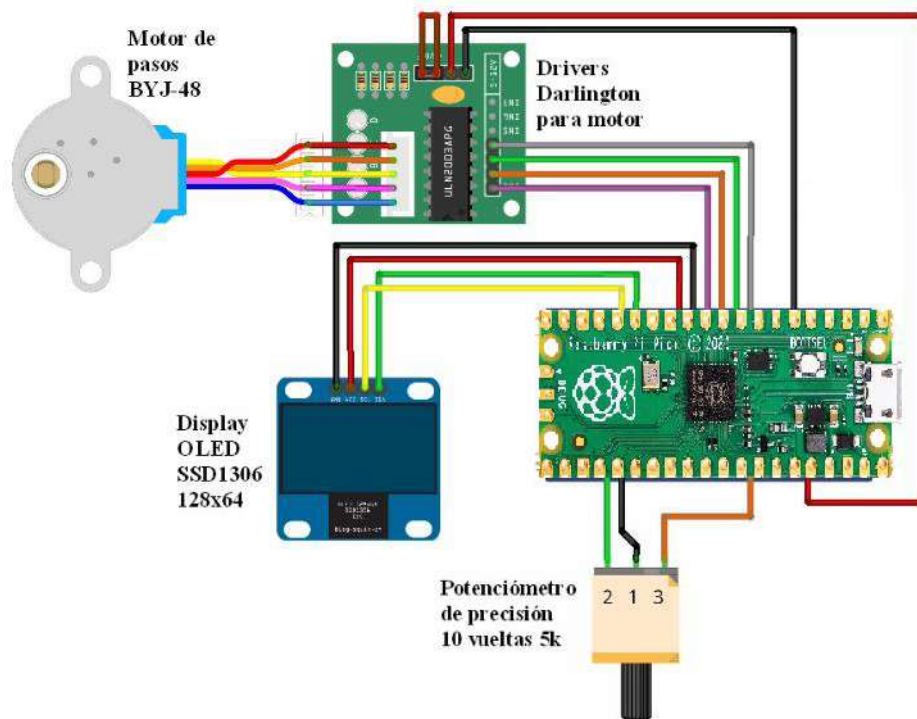


Figura 1. Diagrama de conexiones del proyecto. Incluye un microcontrolador, un motor de pasos BYJ-48, un display OLED SSD1306 y un potenciómetro de precisión para el control manual.

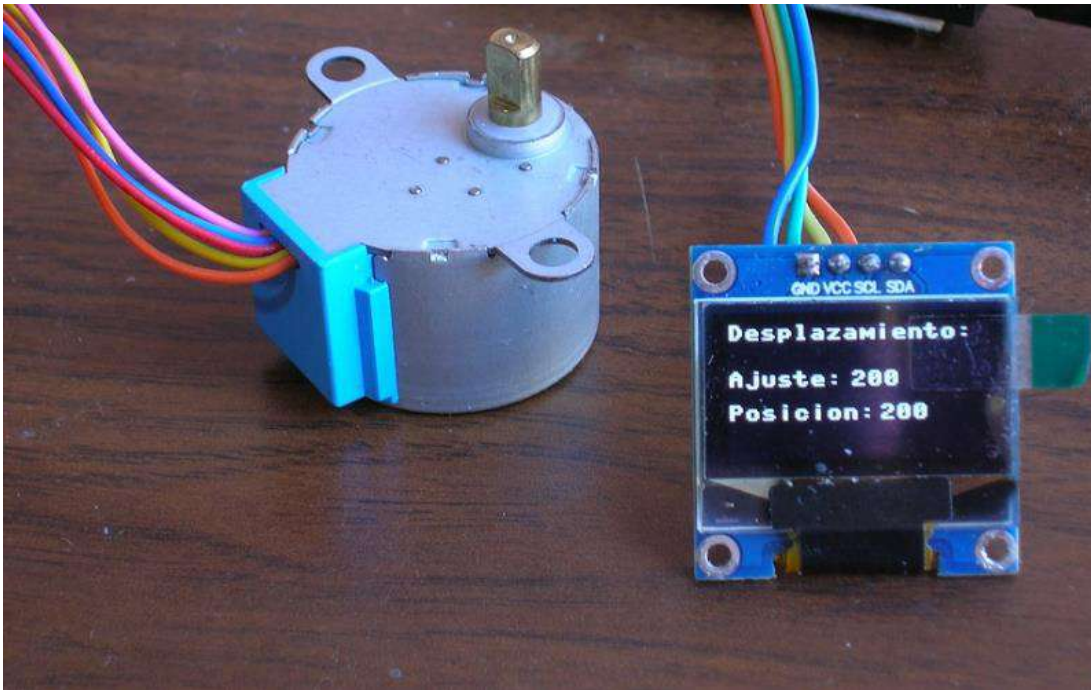


Figura 2. detalle del display OLED SSD1306, de 128 x 64 pixeles, 0.96 pulgadas

Descripción general

En este proyecto se realiza el control en forma manual y por medio de un potenciómetro de precisión. Adicionalmente, se cuenta con un display OLED de alta definición para mostrar los datos del valor deseado y valor final del ángulo de giro del motor, ahorrando la necesidad del uso de un emulador de terminal, ya que todas las componentes del proyecto están conectadas en un espacio reducido.

Se hace uso de la función "Autorun" del lenguaje Micropython.

Con ayuda de las figuras 1 y 2, se hacen las conexiones de los componentes usando cables del tipo dupont, como se muestra.

Para hacer el programa Autorun, se carga el archivo llamado main.py. Una vez cargado, puede prescindir de la computadora y conectar directamente al microcontrolador. El programa se ejecuta automáticamente al encender la fuente.

Código fuente, main.py

```
from machine import Pin, ADC, I2C
import ssd1306
from time import sleep

i2c = I2C(scl=Pin(5), sda=Pin(4))
pot=ADC(0)
```

```

oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)

pin1 = Pin(15, Pin.OUT)
pin2 = Pin(13, Pin.OUT)
pin3 = Pin(12, Pin.OUT)
pin4 = Pin(14, Pin.OUT)

tiempo = 0.005

WaveDrive=[
    [1,0,0,0],
    [0,1,0,0],
    [0,0,1,0],
    [0,0,0,1]]

FullStepDrive=[
    [1,0,0,1],
    [1,1,0,0],
    [0,1,1,0],
    [0,0,1,1]]

HalfStepDrive=[
    [1,0,0,1],
    [1,0,0,0],
    [1,1,0,0],
    [0,1,0,0],
    [0,1,1,0],
    [0,0,1,0],
    [0,0,1,1],
    [0,0,0,1]]

#Secuencias sentido de giro HORARIO wavedrive
#=====
def secuenciaWaveDrive_horario(tiempo):
    for indice in range(2048):
        pin1.value(WaveDrive[indice%4][3])
        pin2.value(WaveDrive[indice%4][2])
        pin3.value(WaveDrive[indice%4][1])
        pin4.value(WaveDrive[indice%4][0])
        sleep(tiempo)

#Secuencias sentido de giro AHORARIO wavedrive
#=====
def secuenciaWaveDrive_Ahorario(tiempo):
    for indice in range(2048):
        pin1.value(WaveDrive[indice%4][0])
        pin2.value(WaveDrive[indice%4][1])
        pin3.value(WaveDrive[indice%4][2])
        pin4.value(WaveDrive[indice%4][3])
        sleep(tiempo)

#Secuencia de paro
#=====
def secuenciaParo():
    pin1.value(0)

```

```

pin2.value(0)
pin3.value(0)
pin4.value(0)
sleep(tiempo)

#Secuencia gira n grados, horario, antihorario
#=====
def giroGrados (grados, tiempo) :
    if (grados >= 0) :
        for indice in range(int((2048/360)*grados)) :
            pin1.value(WaveDrive[indice%4][0])           #horario
            pin2.value(WaveDrive[indice%4][1])
            pin3.value(WaveDrive[indice%4][2])
            pin4.value(WaveDrive[indice%4][3])
            sleep(tiempo)
        else:
            for indice in range(int((2048/360)*abs(grados))) :
                pin1.value(WaveDrive[indice%4][3])       #Ahorario
                pin2.value(WaveDrive[indice%4][2])
                pin3.value(WaveDrive[indice%4][1])
                pin4.value(WaveDrive[indice%4][0])
                sleep(tiempo)

#Funcion principal
#=====

def main() :
    i=0
    pos_actual=0
    #0123456789012345
    oled.text(' Control de giro', 0, 0 )
    oled.text('angular mediante', 0, 15)
    oled.text(' potenciómetro ', 0, 30)
    oled.text(' DESPLAZANDO...', 0, 45)
    oled.show()

    while (True) :
        nueva_posicion=2*pot.read()
        if (nueva_posicion > pos_actual) :
            pos_actual=pos_actual+1
            pin1.value(WaveDrive[pos_actual%4][3])     #Horario
            pin2.value(WaveDrive[pos_actual%4][2])
            pin3.value(WaveDrive[pos_actual%4][1])
            pin4.value(WaveDrive[pos_actual%4][0])
            sleep(tiempo)
        elif (nueva_posicion < pos_actual) :
            pos_actual=pos_actual-1
            pin1.value(WaveDrive[pos_actual%4][3])     #Ahorario
            pin2.value(WaveDrive[pos_actual%4][2])
            pin3.value(WaveDrive[pos_actual%4][1])
            pin4.value(WaveDrive[pos_actual%4][0])
            sleep(tiempo)
        else:
            pin1.value(0)                               #Paro
            pin2.value(0)
            pin3.value(0)
            pin4.value(0)
            sleep(tiempo)

```



```

oled.fill(0)
oled.text('Desplazamiento:', 0, 0 )
oled.text('Ajuste: ', 0, 22)
oled.text('Posicion: ', 0, 38)
oled.text(grado_ajuste, 60, 22)
oled.text(grado_actual, 74, 38)
oled.show()

i=i+1
if(i>50):
    grado_actual=str(int((360/2048)*pos_actual))
    grado_ajuste=str(int((360/2048)*nueva_posicion))
    print('Posicion actual: '+grado_actual+' grados, Posicion de ajuste:
'+grado_ajuste+' grados')
    #oled.text(grado_ajuste, 74, 22)
    #oled.text(grado_actual, 74, 38)
    #oled.show()

i=0

#Programa principal
#=====

main()

#=====

from machine import Pin, ADC, I2C
import ssd1306
from time import sleep

i2c = I2C(scl=Pin(5), sda=Pin(4))
pot=ADC(0)

oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)

pin1 = Pin(15, Pin.OUT)
pin2 = Pin(13, Pin.OUT)
pin3 = Pin(12, Pin.OUT)
pin4 = Pin(14, Pin.OUT)

tiempo = 0.005

WaveDrive=[
    [1,0,0,0],
    [0,1,0,0],
    [0,0,1,0],
    [0,0,0,1]]

FullStepDrive=[
    [1,0,0,1],
    [1,1,0,0],
    [0,1,1,0],

```

```

[0,0,1,1]]

HalfStepDrive=[
  [1,0,0,1],
  [1,0,0,0],
  [1,1,0,0],
  [0,1,0,0],
  [0,1,1,0],
  [0,0,1,0],
  [0,0,1,1],
  [0,0,0,1]]

#Secuencias sentido de giro HORARIO wavedrive
#=====
def secuenciaWaveDrive_horario(tiempo):
    for indice in range(2048):
        pin1.value(WaveDrive[indice%4][3])
        pin2.value(WaveDrive[indice%4][2])
        pin3.value(WaveDrive[indice%4][1])
        pin4.value(WaveDrive[indice%4][0])
        sleep(tiempo)

#Secuencias sentido de giro AHORARIO wavedrive
#=====
def secuenciaWaveDrive_Ahorario(tiempo):
    for indice in range(2048):
        pin1.value(WaveDrive[indice%4][0])
        pin2.value(WaveDrive[indice%4][1])
        pin3.value(WaveDrive[indice%4][2])
        pin4.value(WaveDrive[indice%4][3])
        sleep(tiempo)

#Secuencia de paro
#=====
def secuenciaParo():
    pin1.value(0)
    pin2.value(0)
    pin3.value(0)
    pin4.value(0)
    sleep(tiempo)

#Secuencia gira n grados, horario, antihorario
#=====
def giroGrados(grados, tiempo):
    if (grados >= 0):
        for indice in range(int((2048/360)*grados)):
            pin1.value(WaveDrive[indice%4][0]) #horario
            pin2.value(WaveDrive[indice%4][1])
            pin3.value(WaveDrive[indice%4][2])
            pin4.value(WaveDrive[indice%4][3])
            sleep(tiempo)
    else:
        for indice in range(int((2048/360)*abs(grados))):
            pin1.value(WaveDrive[indice%4][3]) #Ahorario
            pin2.value(WaveDrive[indice%4][2])
            pin3.value(WaveDrive[indice%4][1])
            pin4.value(WaveDrive[indice%4][0])
            sleep(tiempo)

```

```

#Funcion principal
#=====

def main():
    i=0
    pos_actual=0
        #0123456789012345
    oled.text(' Control de giro', 0, 0 )
    oled.text('angular mediante', 0, 15)
    oled.text(' potenciómetro ', 0, 30)
    oled.text(' DESPLAZANDO...', 0, 45)
    oled.show()

while(True):
    nueva_posicion=2*pot.read()
    if(nueva_posicion>pos_actual):
        pos_actual=pos_actual+1
        pin1.value(WaveDrive[pos_actual%4][3]) #Horario
        pin2.value(WaveDrive[pos_actual%4][2])
        pin3.value(WaveDrive[pos_actual%4][1])
        pin4.value(WaveDrive[pos_actual%4][0])
        sleep(tiempo)
    elif(nueva_posicion<pos_actual):
        pos_actual=pos_actual-1
        pin1.value(WaveDrive[pos_actual%4][3]) #Ahorario
        pin2.value(WaveDrive[pos_actual%4][2])
        pin3.value(WaveDrive[pos_actual%4][1])
        pin4.value(WaveDrive[pos_actual%4][0])
        sleep(tiempo)
    else:
        pin1.value(0) #Paro
        pin2.value(0)
        pin3.value(0)
        pin4.value(0)
        sleep(tiempo)

    oled.fill(0)
    oled.text('Desplazamiento:', 0, 0 )
    oled.text('Ajuste: ', 0, 22)
    oled.text('Posicion: ', 0, 38)
    oled.text(grado_ajuste, 60, 22)
    oled.text(grado_actual, 74, 38)
    oled.show()

    i=i+1
    if(i>50):
        grado_actual=str(int((360/2048)*pos_actual))
        grado_ajuste=str(int((360/2048)*nueva_posicion))
        print('Posicion actual: '+grado_actual+' grados, Posicion de ajuste:
'+grado_ajuste+' grados')
        #oled.text(grado_ajuste, 74, 22)
        #oled.text(grado_actual, 74, 38)
        #oled.show()

    i=0

```

```
#Programa principal
#=====
main()
#=====
```

11

Raspberry Pi Pico 2 W

Giroscopio y acelerómetro

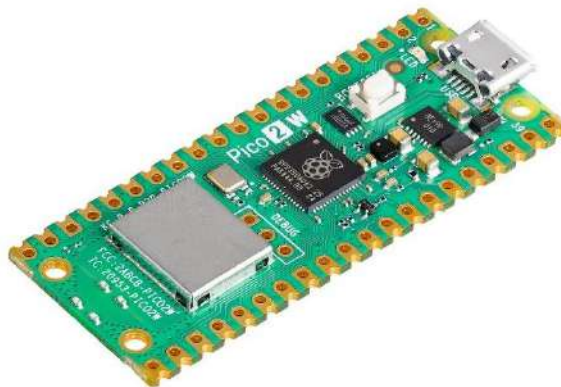
RESUMEN

En este documento se intenta crear una guía para la utilización del módulo MPU-6050, que consiste en un acelerómetro y un giroscopio, de 3 ejes, utilizando una tarjeta raspberry pi pico 2 W.

En esta misma guía se abordarán rápidamente las características del módulo mencionado, el software que se utilizará para configurar la comunicación entre dispositivos y programar alguna función en la Raspberry.

MATERIALES

- Raspberry pi pico 2 W



- Módulo MPU-6050



- Headers para colocar o soldar en la raspberry
- Jumpers o cables de conexión para los headers

DESCRIPCIÓN Y CARACTERÍSTICAS DEL MODULO MPU-6050

Este módulo es un sensor de movimiento, está diseñado para funcionar en tarjetas de desarrollo como Arduino y Raspberry, consta de un acelerómetro y un giroscopio, cada uno de ellos con 3 ejes, económico y de bajo consumo de energía. Debido a su alta sensibilidad, es utilizado para diferentes aplicaciones, por ejemplo, robótica, celulares o dispositivos de videojuegos.

Consta de los siguientes componentes, características y funciones:

- Giroscopio de 3 ejes, con convertidor analógico-digital de 16 bits, programable para los rangos: +/-250, +/-500, +/-1000, +/-2000 dps (grados por segundo)
- Acelerómetro de 3 ejes, con convertidor analógico-digital de 16 bits, programable para los rangos: +/-2g, +/-4g, +/-8g, +/-16g
- Interfaces de comunicación I2C y SPI
- El módulo tiene acceso a 8 pines, de los 24 que tiene en su totalidad el sensor.
- Rango de voltaje de alimentación: 2.375V a 3,46V
- Consumo de corriente: 3.9 mA en modo normal, 5 uA en modo de reposo
- Rango de temperatura del sensor: -40 a 85 grados centígrados.

Pines del módulo:

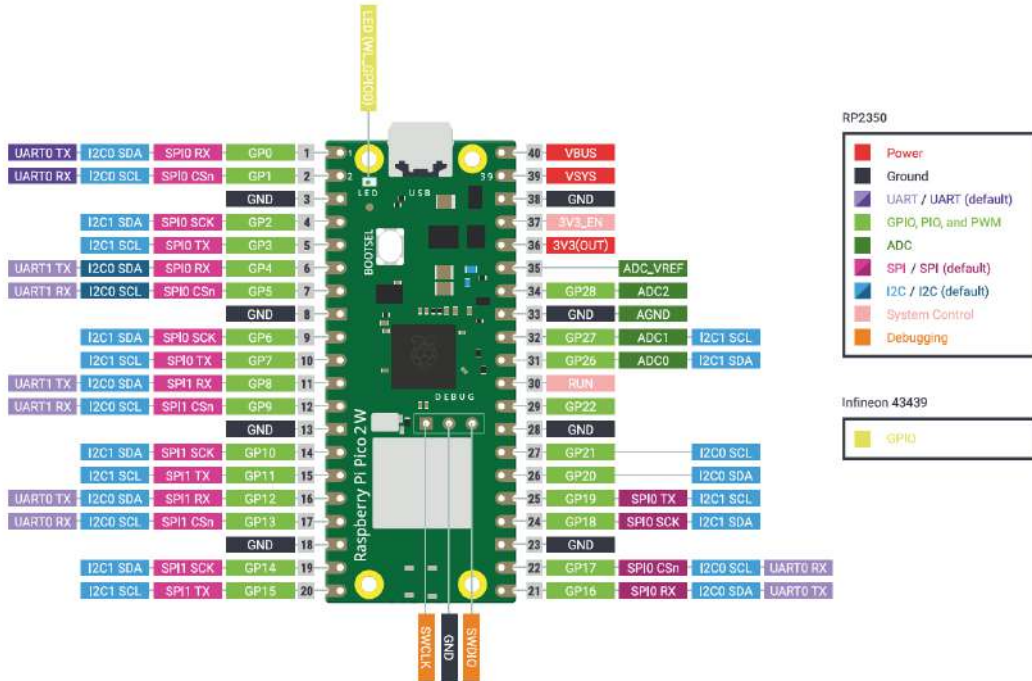
- Vcc: Pin de alimentación del módulo, acepta entre 3.3V y 5V, dependiendo del regulador.
- GND: Pin de tierra.
- SCL: Línea de reloj I2C, usado para la comunicación con el microcontrolador
- SDA: Línea de datos I2C, usado para la comunicación con el microcontrolador
- XDA: Línea de datos I2C auxiliar, para conectar sensores externos I2C
- XCL: Línea de reloj I2C auxiliar, para conectar sensores externos I2C
- AD0: Pin de selección de dirección I2C esclavo. Utilizado cuando se tiene más de un módulo MPU-6050
- INT Pin de interrupción, utilizado para indicar al microcontrolador que un dato está disponible para ser leído desde un módulo MPU-6050

CONEXIÓN CON RASPBERRY PI PICO

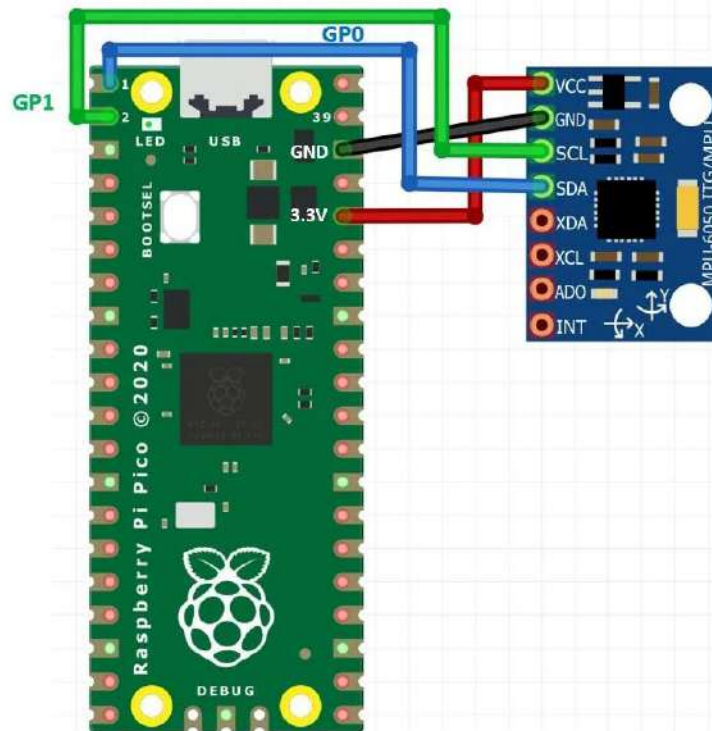
Para este paso sólo se requieren los pines Vcc, GND, SCL y SDA del módulo MPU-6050.

La raspberry pi pico 2 W tiene 2 controladores I2C (I2C0 e I2C1), se puede configurar cada uno de ellos, a través de diferentes pines GPIO (general purpose input output) como se observa en la imagen, pero es importante asignar por código los pines que se utilizarán

Figure 2. The pinout of the Pico 2 W board



En este caso se usará el I2C0 con los pines GP0 y GP1, como se muestra en la siguiente imagen, ya que la funcionalidad de esos pines coincide con los de una Raspberry pi pico.



SOFTWARE, CÓDIGO Y LIBRERÍAS PARA UTILIZAR EL MPU-6050

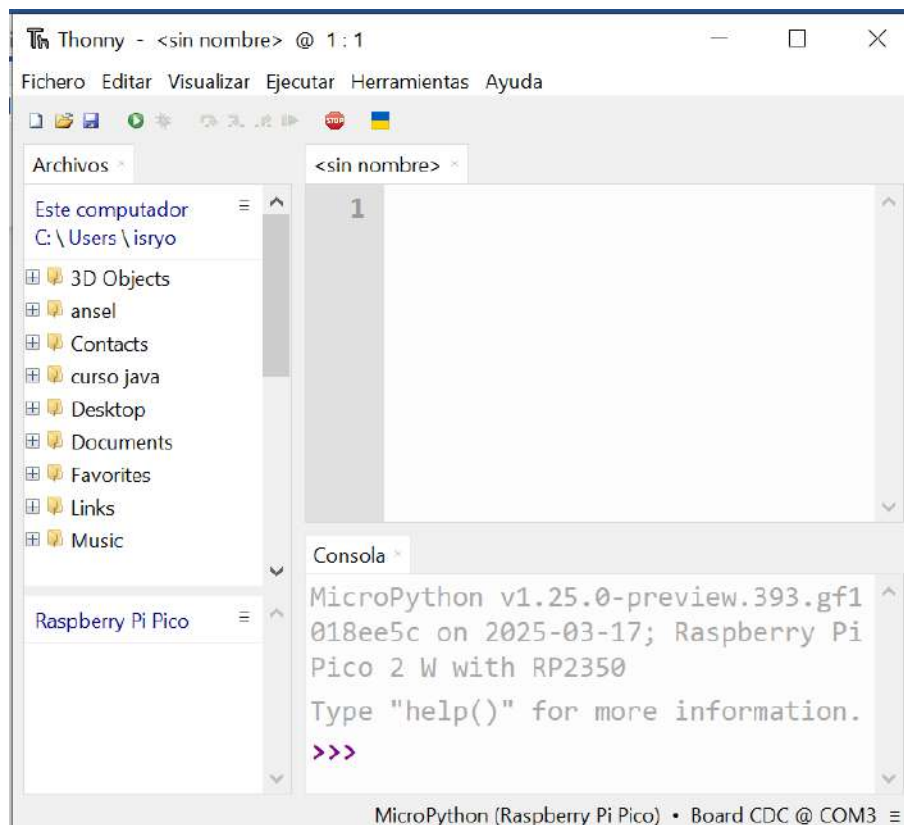
La interfaz de desarrollo propuesta es Thonny, que es una IDE basada en Python para microcontroladores. Se puede descargar en la siguiente página <https://thonny.org/>.

Por otro lado, se cargará el software MicroPython, que es un lenguaje de programación que se puede ejecutar en la Raspberry, se puede descargar desde aquí <https://www.raspberrypi.com/documentation/microcontrollers/micropython.html> en este caso en particular, usamos el apropiado para Raspberry pi pico 2 W.

Para cargar MicroPython en la Raspberry primero conectamos la Raspberry a la computadora y apretar el botón bootsel, se abrirá una carpeta de la memoria de la Raspberry, ahí copiamos el archivo descargado desde la página anterior.

Otra manera de descargar el archivo es ejecutar Thonny y en el menú Ejecutar>Configurar intérprete>Intérprete encontraremos una opción “Instala o actualiza MicroPython”, seleccionamos la familia RP2, variante Raspberry Pi Pico 2 W, nos dará opción de elegir la variante correspondiente y finalmente aceptamos instalar.

Después de esos procesos se reiniciará la Raspberry y es necesario reiniciar Thonny para ejecutar el IDE de manera correcta.



En la sección consola nos indica sobre qué tarjeta se está programando.

Se utilizarán las dos siguientes librerías para la configuración de los componentes del MPU-6050 que ejecutará nuestra raspberry: [imu.py](https://github.com/micropython-imu/micropython-mpu9x50) y [vector3d.py](https://github.com/micropython-imu/micropython-mpu9x50), las cuales pueden ser descargadas desde el siguiente enlace <https://github.com/micropython-imu/micropython-mpu9x50>.

El código que podemos utilizar es el siguiente y lo guardaremos con el nombre main.py:

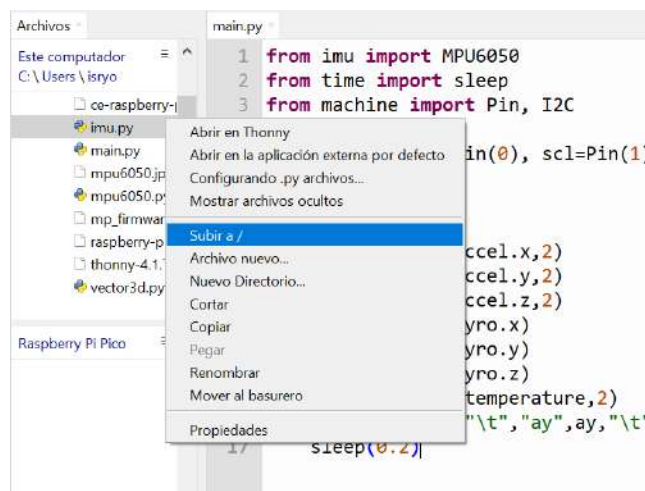
```
from imu import MPU6050
from time import sleep
from machine import Pin, I2C

i2c = I2C(0, sda=Pin(0), scl=Pin(1), freq=400000)
imu = MPU6050(i2c)

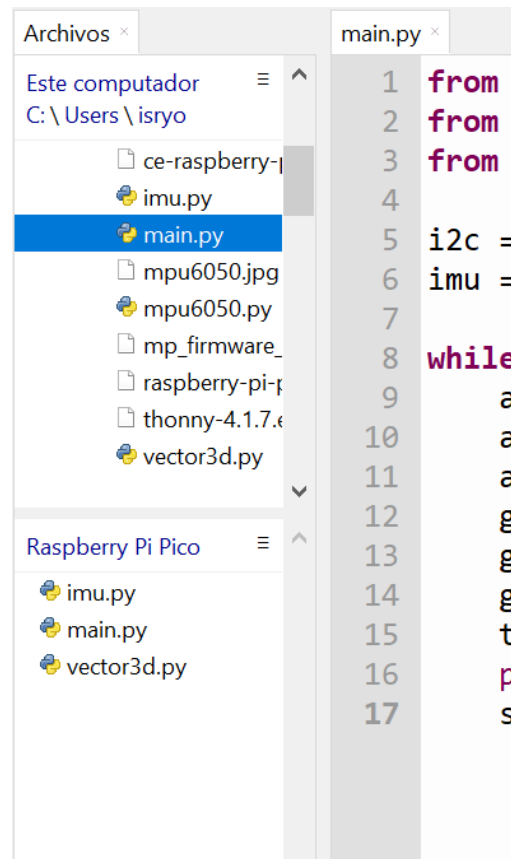
while True:
    ax=round(imu.accel.x,2)
    ay=round(imu.accel.y,2)
    az=round(imu.accel.z,2)
    gx=round(imu.gyro.x)
    gy=round(imu.gyro.y)
    gz=round(imu.gyro.z)
    tem=round(imu.temperature,2)
    print("ax",ax,"\t","ay",ay,"\t","az",az,"\t","gx",gx,"\t","gy",gy,"\t","gz",gz,"\t","Temperature",tem,"
",end="\r")
    sleep(0.2)
```

CARGANDO LOS CÓDIGOS

Utilizaremos Thonny para subir las librerías y el código a la Raspberry. Para ello iremos a menú Visualizar>archivos (debe estar activado) y en el lado izquierdo buscaremos la carpeta donde guardamos cada uno de nuestros códigos y los cargaremos en la Raspberry, dando click derecho sobre el nombre del archivo y enseguida aplicamos subir a/ como se muestra en la siguiente imagen.



Una vez que se subieron correctamente los códigos aparecerán los mismos en la sección de la memoria de la raspberry utilizada.



Una vez terminado todo este proceso se puede ejecutar el proyecto y observar los resultados en la consola.

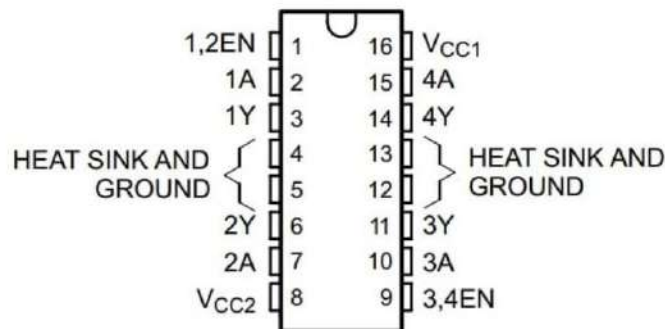
12

Raspberry Pi Pico**Control de motor de Corriente Directa**

Los motores de DC son ampliamente utilizados en diferentes proyectos de robótica y electrónica. Dada la versatilidad de la Raspberry Pi Pico, su tamaño y los diferentes sensores y actuadores que se pueden conectar, además del bajo consumo de energía es una opción muy importante a considerar, utilizar este microcontrolador para el manejo de este tipo de motores.

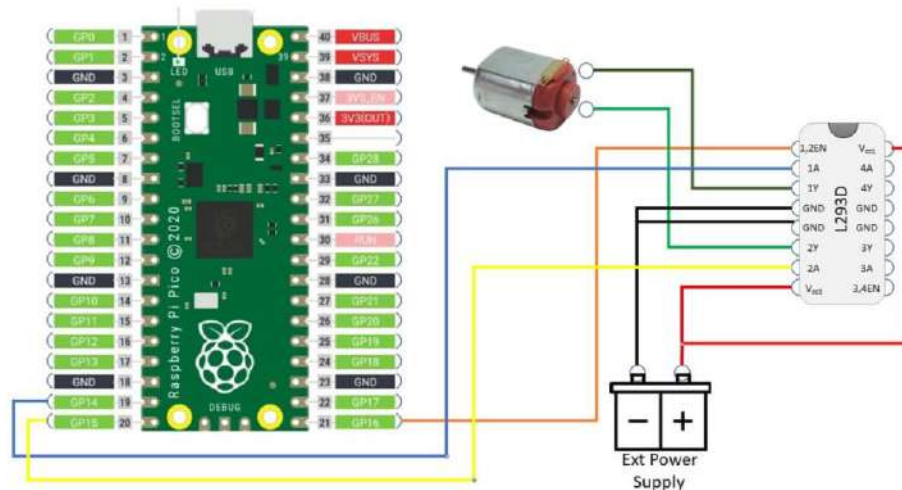
L293D

Este componente es un puente H de 4 canales. La energía que puede proporcionar al motor está en un rango de 4.6-36 V y una corriente de hasta 600 mA, para que la lógica en el componente sea funcional, requiere 5V de alimentación.



En la imagen se observan los habilitadores de los canales (EN), las entradas lógicas (A) y las salidas de potencia que van directamente al motor (Y), los pines de tierra (GROUND) van conectados a una tierra común, V_{CC1} es la alimentación lógica del circuito (5V) y V_{CC2} es la alimentación de voltaje con el que opera el motor de DC.

CONEXIÓN AL MOTOR DE DC



PROCEDIMIENTO PARA EL MANEJO DEL MOTOR

Como se puede ver en el diagrama propuesto, se utilizan los pines de propósito general como salidas. Para crear el código se utiliza la IDE Thonny, que se mencionó en el proyecto propuesto anteriormente. Con el siguiente código aplicado a las conexiones de la imagen anterior, que es compatible con el modelo de Raspberry Pi Pico.

```
from machine import Pin, PWM
from time import sleep
```

```
pwmPIN=16
cwPin=14
acwPin=15
```

```
def motorMove(speed,direction,speedGP,cwGP,acwGP):
    if speed > 100: speed=100
    if speed < 0: speed=0
    Speed = PWM(Pin(speedGP))
    Speed.freq(50)
    cw = Pin(cwGP, Pin.OUT)
    acw = Pin(acwGP, Pin.OUT)
    Speed.duty_u16(int(speed/100*65536))
    if direction < 0:
        cw.value(0)
        acw.value(1)
    if direction == 0:
        cw.value(0)
        acw.value(0)
    if direction > 0:
        cw.value(1)
        acw.value(0)
```

```
# main program
motorMove(100,1,pwmPIN,cwPin,acwPin)
sleep(5)
motorMove(100,0,pwmPIN,cwPin,acwPin)
```

Como se observa en el código, el PIN 16 se utiliza como salida PWM, que va conectada directamente al habilitador de los canales 1 y 2 del circuito L293D.

También se observa que hay 5 parámetros a considerar en el código.

El primero es la velocidad que va entre 0 y 100, el cual indica el ciclo de trabajo en el PWM.

El segundo es la dirección de la rotación del motor, los valores positivos crean un giro en sentido horario, mientras que los valores negativos crean un sentido de giro antihorario, ambos deben ser de tipo entero.

El tercer parámetro envía el valor de PWM, a través del pin utilizado como salida PWM, dicha salida define la velocidad del motor.

El cuarto y quinto parámetro son salidas digitales que se activan o desactivan dependiendo del valor que se haya colocado en el parámetro “dirección”

El código propuesto se guarda y se carga directamente en la Raspberry, de la misma manera que se carga el archivo main.py del proyecto propuesto anteriormente.

Referencias

<https://thonny.org/>

<https://www.raspberrypi.com/>

<https://micropython.org/>

<https://cpulator.01xz.net/>

<https://www.raspberrypi.com/documentation/>

<https://developer.arm.com/documentation>
