

Programación en C para sistemas embebidos

Arquitectura 2023

Uso de la proposición asm()

Es posible ejecutar código ensamblador dentro de un programa en C utilizando la proposición `asm()`, la cual puede recibir hasta 4 argumentos con la siguiente sintaxis:

```
asm(código: operandosdestino: operandosfuente[:restricciones]);
```

```
asm("NOP"); // No operación, tarda 1 ciclo de reloj
```

```
asm("SBI 0x18, 0"); // Pone en alto el bit 0 de PORTB
```

```
asm("CBI 0x18, 0"); // Pone en bajo el bit 0 de PORTB
```

```
asm("SEI \n" // Habilita las interrupciones y
```

```
"CLC"); // limpia la bandera de acarreo
```

Mezclar código C y Assembler

- Una función en lenguaje C debe declararse como externa en el código ensamblador

.extern mi_función_C

- Una rutina en lenguaje ensamblador debe declararse como global en el código ensamblador

.global mi_fct_assembler

- Un archivo C que pretenda llamar a la rutina en ensamblador deberá tener un prototipo de función que declare la rutina como externa

extern unsigned char mi_fct_assembler (unsigned char, unsigned int);

Variables globales

- Es posible que tanto el código ensamblador como el C accedan a la misma variable global.
- Tal variable tendría que ser una variable global en el código C y declarada como externa en el código ensamblador.

En el archivo .c:

- *unsigned char mi_variable;*

En assembler debería ser codificada:

- *.extern mi_variable*

Para tener en cuenta en el código assembler

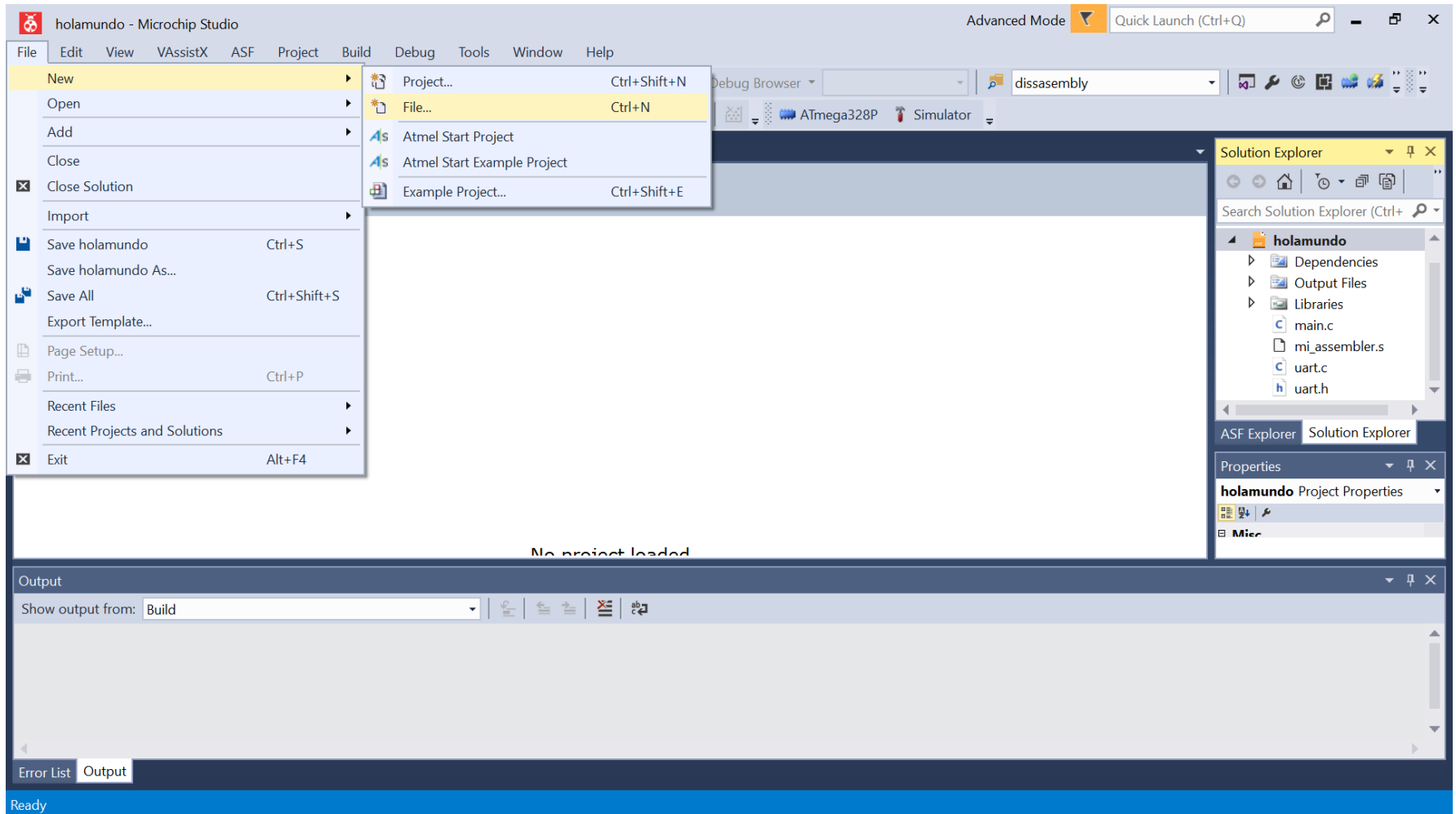
- **r0 es un registro temporal y puede ser utilizado por el código generado por el compilador. Si usa este registro, antes de llamar a una función C debe resguardar el contenido.**
- **El compilador supone que r1 siempre contiene cero. El código de ensamblador que use este registro debe borrar el registro antes de retornar de la rutina**
- **r2-r17, r28, r29 son registros de "resguardo por llamada", lo que significa que las funciones C que son llamadas (ejecutadas) deben dejar estos registros inalterados. La rutina en lenguaje ensamblador llamada desde C deberá resguardar el contenido de todos los que necesite utilizar**
- **r18-r27, r30, r31 son registros de "uso por llamada", lo que significa que los registros están disponibles para cualquier código. El código ensamblador deberá resguardar el contenido de cualquiera de estos registros que hayan sido utilizados previamente.**

Pasaje de argumentos

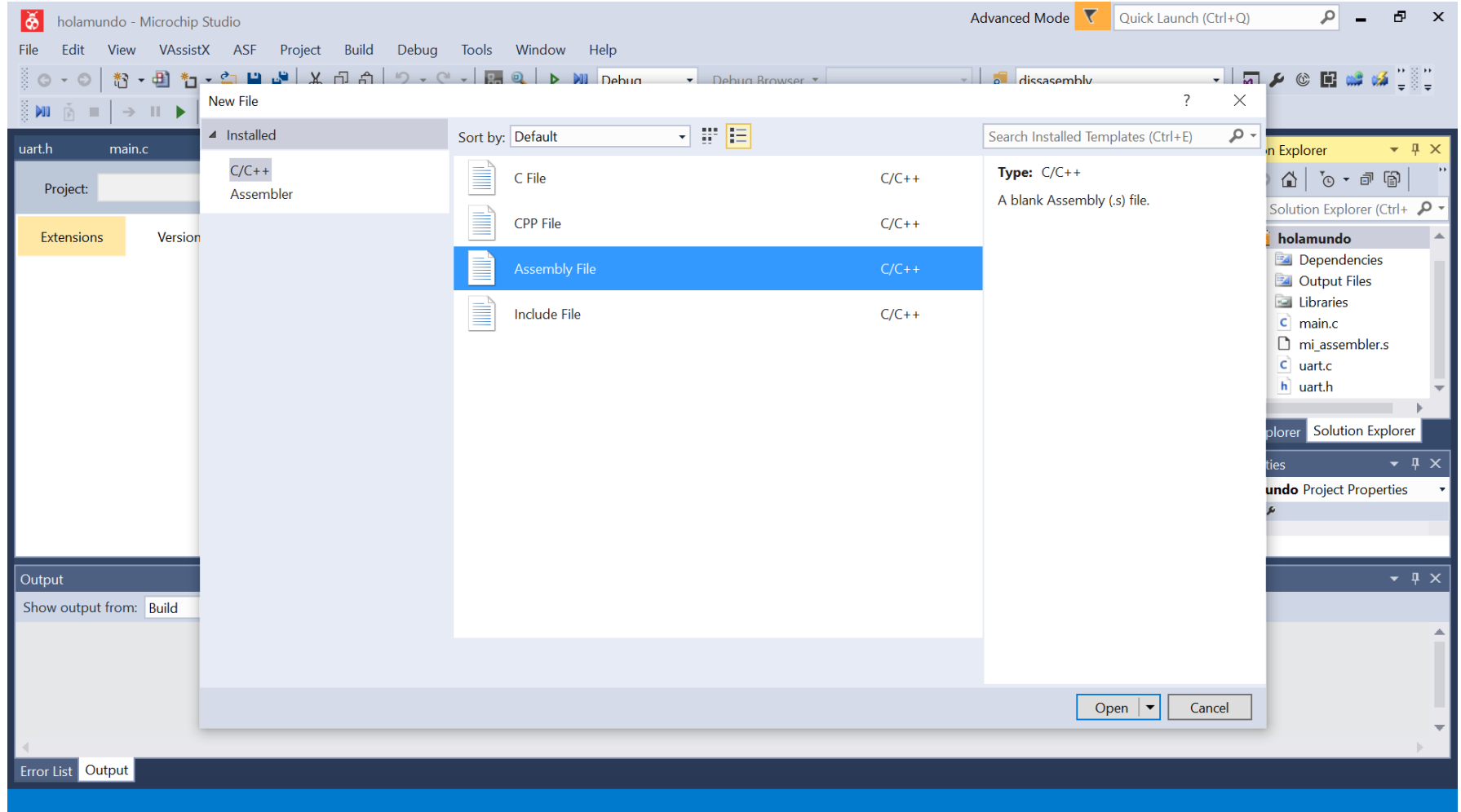
- Los argumentos en una lista de argumentos fijos se asignan, de izquierda a derecha, a los registros r25 a r18. Todos los argumentos usan un número par de registros
- Los valores devueltos por una función se guardan como se muestra en la tabla:

registro	r19	r18	r21	r20	r23	r22	r25	r24
byte	B7	B6	B5	B4	B3	B2	B1	B0

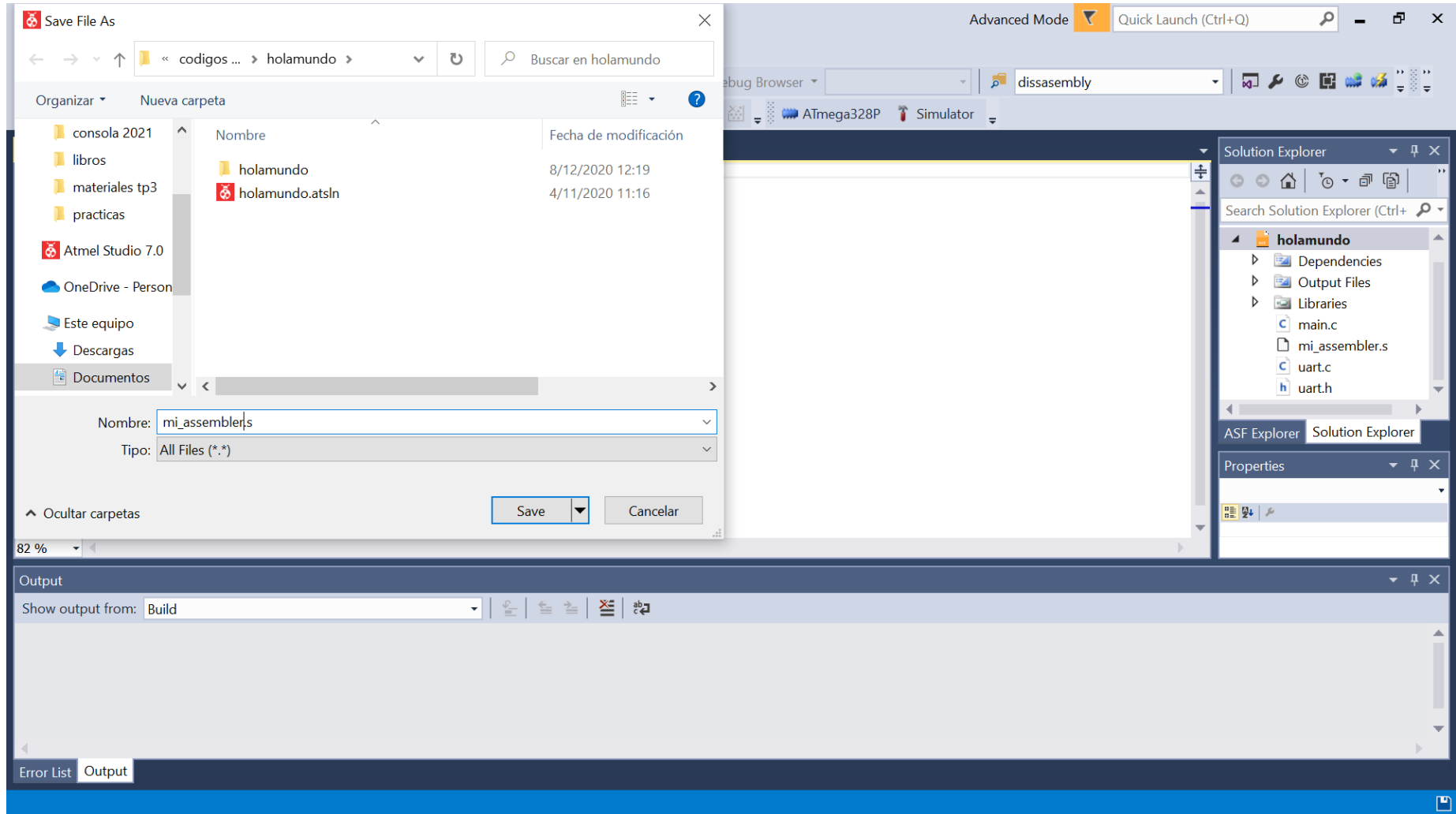
Ejemplo de c y assembler sobre el proyecto “hola mundo”



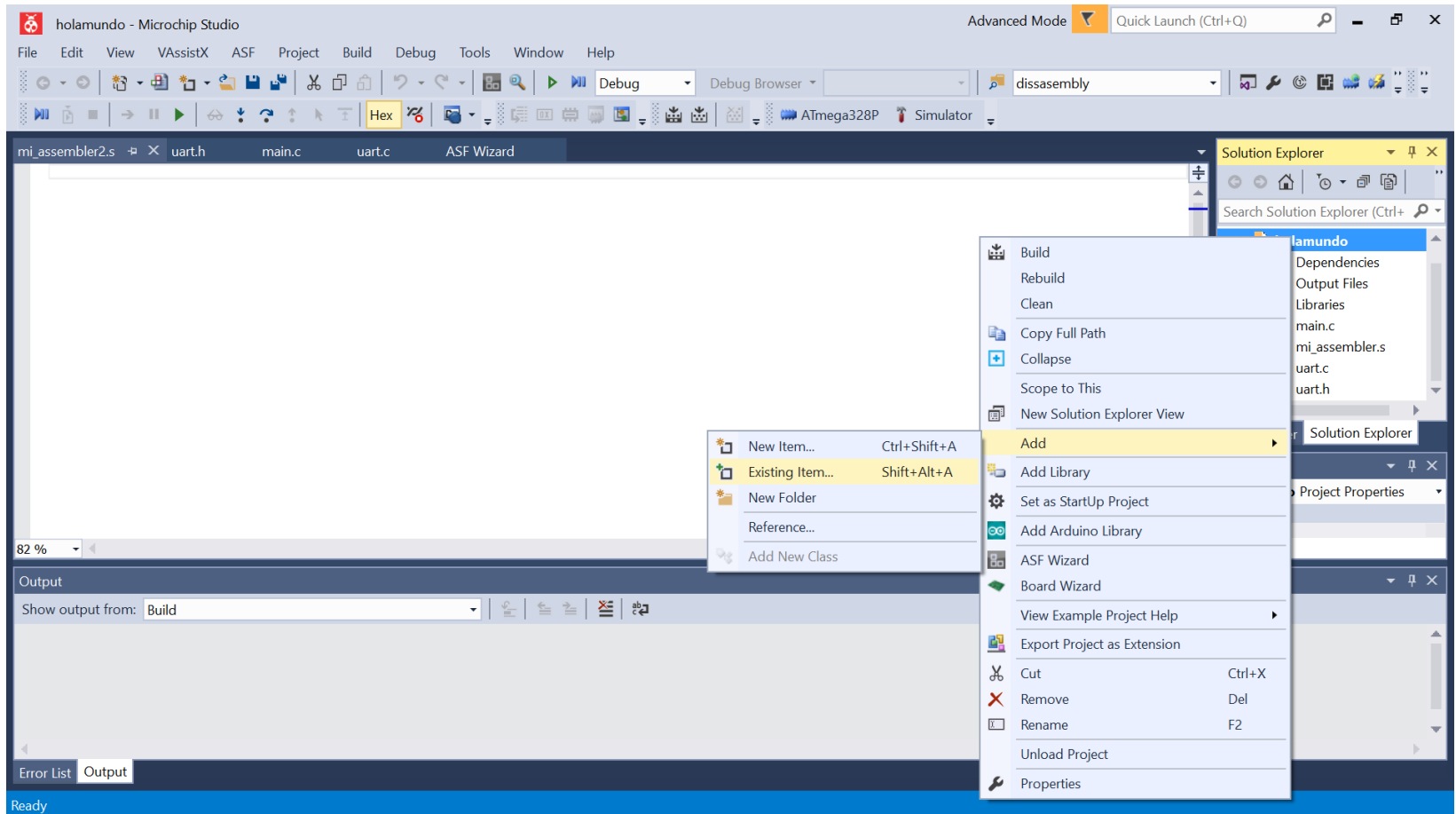
Crear nuevo archivo assembler



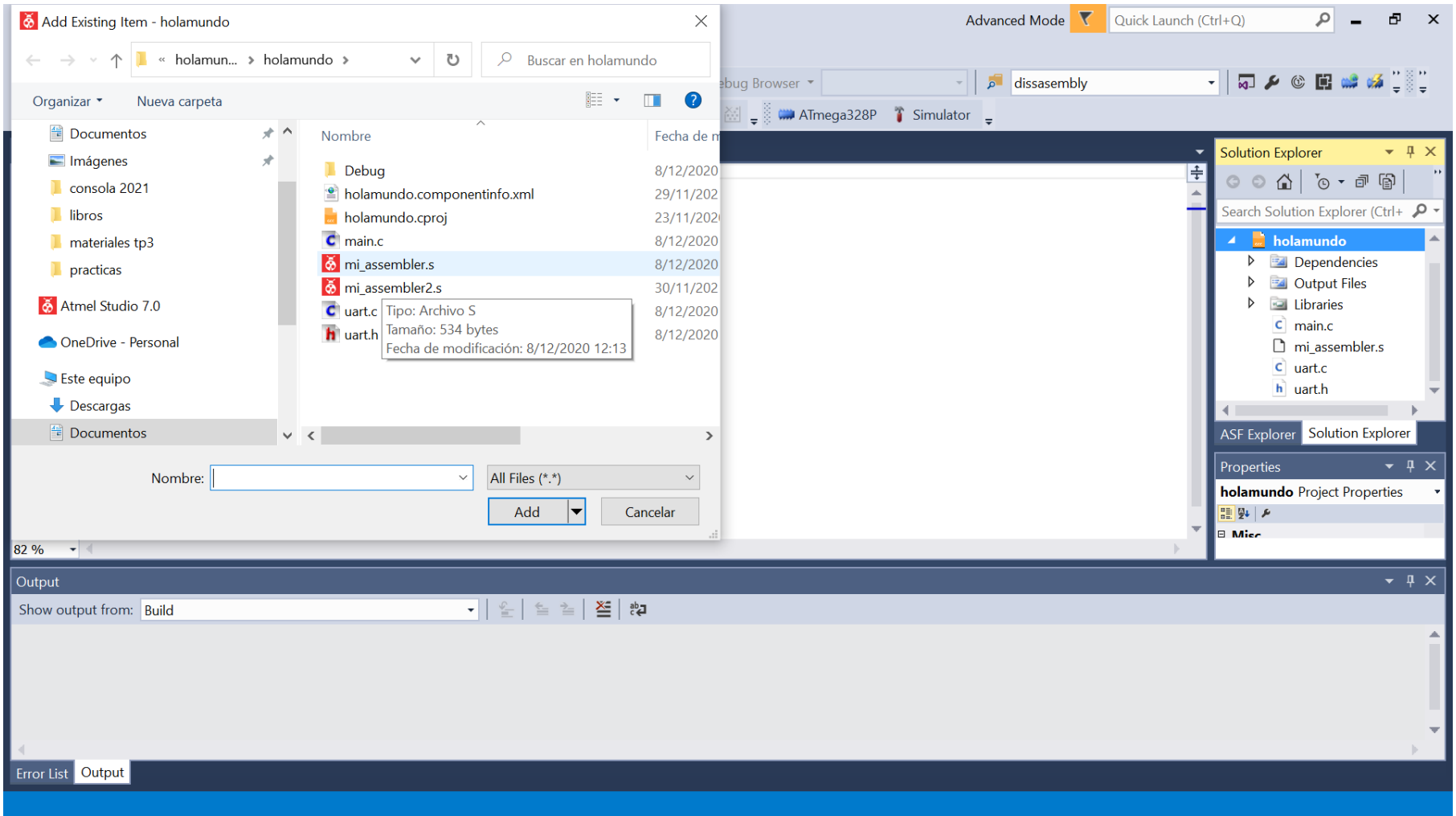
Guardar como mi_assembler.s



Agregar el archivo al proyecto



Agregar el archivo al proyecto



Código del archivo mi_assembler.s

```
#include <avr/io.h>
// #include <avr/interrupt.h>
.extern valor

.global suma
suma:
    add r24,r22
    sts valor,r24
    clr r25
    ret

.global suma_dos
suma_dos:
    ADD R24,R22;val_1 = val_1 + val_2
    CLRR25;return val_1
    ret;
```

En el archivo del programa principal main.c se debe agregar

Fuera de la función main()

```
extern unsigned char suma (unsigned int, unsigned int);  
extern char suma_dos(char, char);  
unsigned char valor=0;  
char String[]="contenido de sum:!! \r\n";  
char String1[]="contenido de valor:!! \r\n";
```

Dentro de la función main()

```
volatile unsigned char val_1, val_2, val_3;  
    volatile unsigned char sum;  
        sum= suma(10,22);  
    val_1 = 0x34;  
    val_2 = 0x56;  
    val_3 = suma_dos(val_1, val_2); // llama a la función en assembler
```

En el bucle infinito while(1)

```
USART_putstring(String);    //Pasa el string a la USART_putstring y
lo envía por el puerto serie

USART_send(sum/10+0x30); //se obtienen el dígito de las decenas y se
pasa a ascii para luego enviar por puerto serie

USART_send(sum%10+0x30); //se obtienen el dígito de las unidades y se
pasa a ascii

USART_putstring("\r\n");

USART_putstring(String1);    //Pasa el string a la USART_putstring y
lo envía por el puerto serie

USART_send(valor/10+0x30);

USART_send(valor%10+0x30);

_delay_ms(5000);           //retardo de 5 segundos para poder reenviarlo
cada 5 segundos
```

Compilar y depurar

- si presionamos alt+F5 vamos al debugging
- Ejecutamos paso a paso y cuando llegamos a la función suma, presionamos step into (F11)
- Luego ejecutamos paso a paso el código en assembler y vemos como cambian los registros

Simular paso a paso la rutina

holamundo (Debugging) - Microchip Studio

Advanced Mode Quick Launch (Ctrl+Q)

File Edit View VAssistX ASF Project Build Debug Tools Window Help

Debug Browser dissassembly

Hex ATmega328P Simulator

Disassembly main.c mi_assembler.s mi_assembler2.s uart.h uart.c

```
#include <avr/io.h>
// #include <avr/interrupt.h>
.extern valor

.global suma
suma:
add r24,r22
sts valor,r24
clr r25
ret

.global change_clock
change_clock:
LDI R25,0x80 ;
STS CLKPR,R25 ;enable updating of CLKPR
STS CLKPR,R24 ;write the new value
ret
```

Processor Status

Name	Value
R17	0x01
R18	0x01
R19	0x00
R20	0x00
R21	0x00
R22	0x16
R23	0x00
R24	0x0A
R25	0x00
R26	0x11
R27	0x01

82 %

I/O

Filter:

Name	Value
Analog Comparator (AC)	
Analog-to-Digital Convert...	
CPU Registers (CPU)	
EEPROM (EEPROM)	

Name Address Value Bits

Memory 4

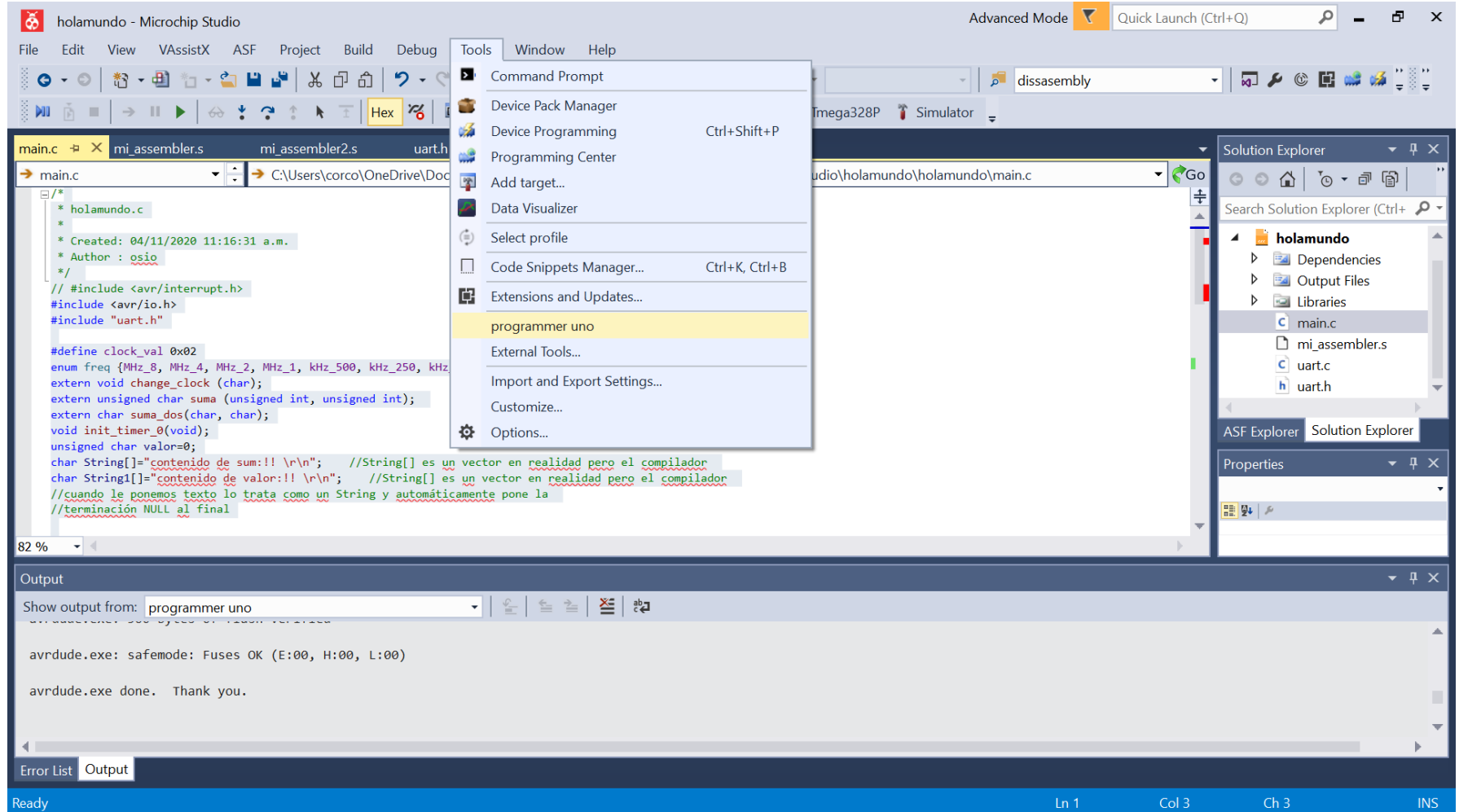
Memory: prog FLASH Address: 0x0000,prog

prog 0x0000	0c 94 34 00 0c 94 51 00 0c 94 51 00 0c 94 51 00	."4.."Q.."Q.."Q.
prog 0x0010	0c 94 51 00 0c 94 51 00 0c 94 51 00 0c 94 51 00	.."Q.."Q.."Q.."Q.
prog 0x0020	0c 94 51 00 0c 94 51 00 0c 94 51 00 0c 94 51 00	.."Q.."Q.."Q.."Q.
prog 0x0030	0c 94 51 00 0c 94 51 00 0c 94 51 00 0c 94 51 00	.."Q.."Q.."Q.."Q.
prog 0x0040	0c 94 61 00 0c 94 51 00 0c 94 51 00 0c 94 51 00	.."a.."Q.."Q.."Q.
prog 0x0050	0c 94 51 00 0c 94 51 00 0c 94 51 00 0c 94 51 00	.."Q.."Q.."Q.."Q.
prog 0x0060	0c 94 51 00 0c 94 51 00 11 24 1f be cf ef d8 e0	.."Q.."Q.."\$.ii0à
prog 0x0070	de bf cd bf 11 e0 a0 e0 b1 e0 ec ee f1 e0 02 c0	p;Í;.à àtàîñà.À
prog 0x0080	05 90 0d 92 a0 31 b1 07 d9 f7 21 e0 a0 e1 b1 e0	...' 1±.Ü÷là áàà
prog 0x0090	01 c0 1d 92 a1 31 b2 07 e1 f7 0e 94 6f 00 0c 94	.À.'j1..á±.."o.."
prog 0x00A0	f4 00 0c 94 00 00 86 0f 80 93 10 01 99 27 08 95	ô.."....€"..™'..

Call Stack Breakpoints Command Window Immediate Window Output Error List Memory 4

Stopped

Por último programamos el kit



Visualizamos en la consola serie

The screenshot displays the Microchip Studio IDE interface. The top menu bar includes File, Edit, View, VAssistX, ASF, Project, Build, Debug, Tools, Window, and Help. The toolbar shows various icons for file operations, debugging, and simulation. The main workspace displays the project files: main.c, mi_assembler.s, mi_assembler2.s, uart.h, and uart.c. The right sidebar contains the Solution Explorer, showing the project structure, and the Properties window, displaying the file properties for main.c.

The **DGI Control Panel** is visible, showing the **Serial Port Control Panel** configuration for **USB-SERIAL CH340 (COM7)**. The configuration includes:

- Baud rate: 9600
- Parity: None
- Stop bits: 1 bit
- Buttons: Disconnect, DTR, RTS, Open Terminal, Autodetect protocols

The **Terminal 1** window shows the following output:

```
contenido de sum:!!  
32contenido de valor:!!  
32contenido de sum:!!  
32contenido de valor:!!  
32contenido de sum:!!  
32contenido de valor:!!  
32contenido de sum:!!  
32contenido de valor:!!  
32contenido de sum:!!  
32contenido de valor:!!
```

The **Output** window shows the following output:

```
avrdude.exe: safemode: Fuses OK (E:00, H:00, L:00)  
avrdude.exe done. Thank you.
```

The status bar at the bottom indicates the IDE is in **Ready** state.