Práctica final 2º SMR

La práctica consiste en crear un compilador y/o un intérprete de un nuevo lenguaje de programación inventado para un nuevo microcontrolador que no existe en el mercado.

Table of Contents

1. Primer parte: Compilador

Un compilador es un programa que se encarga de *traducir* el código escrito en un lenguaje de programación a la ristra de 0 y 1 que es capaz de entender un microcontrolador. A esa ristra de números binarios se le llama código máquina y cada procesador tiene su propio juego de instrucciones y su propio código máquina.

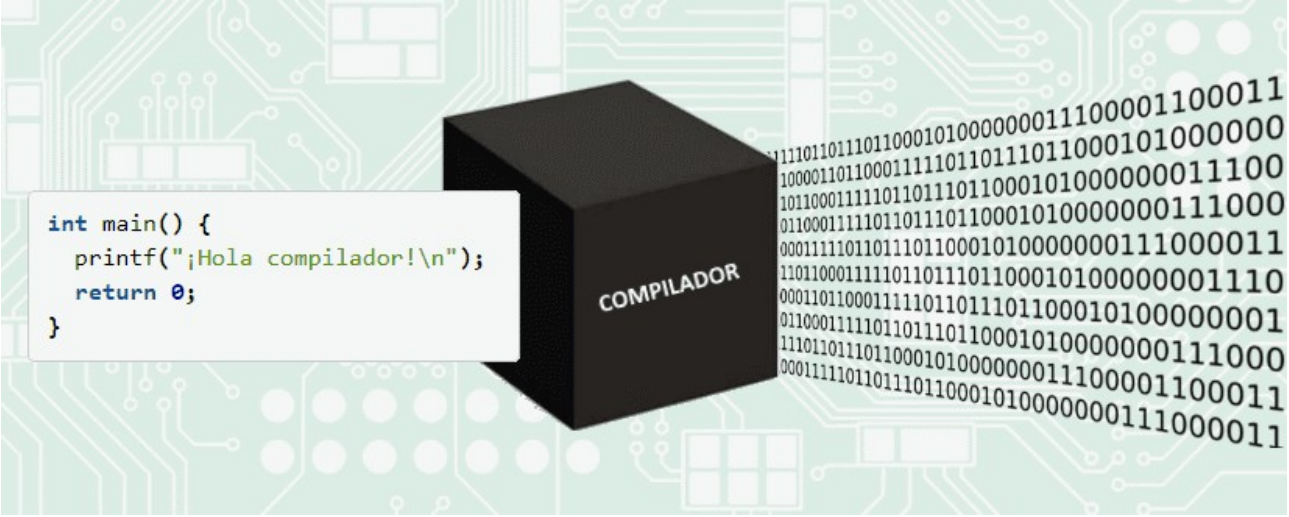

Figure 1: Representación de la función de un compilador

Los lenguajes de programación se dividen en lenguajes de bajo nivel (ensamblador) y lenguajes de alto nivel (C, C++, Javascript, PHP,...). Los lenguajes de bajo nivel son los que están más cercanos al hardware del microprocesador y son por tanto los más sencillos de *compilar* (en realidad, los lenguajes de bajo nivel se *ensamblan* con una utilidad llamada *ensamblador*).

Para simplificar al máximo la tarea, se ha imaginado un lenguaje de bajo nivel y un microcontrolador de 8 bits muy sencillo al que llamaremos procSMR2.

1.a Tarea

La tarea consiste en la creación de una web que permita la traducción de un lenguaje tipo ensamblador al código máquina del microcontrolador ficcticio procSMR2.

1.b El microcontrolador procSMR2

Este micro tiene 8 registros de propósito general a los que llamaremos R0, R1, R2... R7 que pueden contener cualquier dato de 8 bits.

Las instrucciones del procesador ocupan siempre 2 bytes (aunque se desperdicie memoria, esto simplifica mucho la tarea posterior).

Además, consta de una memoria para código de 512 bytes, por lo que los programas creados para este microcontrolador pueden contener como máximo de 256 instrucciones. Se considerará que un programa ha terminado de ejecutarse si se llega a ejecutar la instrucción que ocupa la posición 255 (o posterior).

1.c El lenguaje SMR2

Los programas en este lenguaje al que llamaremos SMR2 consta de unas pocas instrucciones. Cada instrucción tiene que estar en una línea distinta.

Supondremos por simplificar la tarea que el programador siempre escribe programas correctos (sin errores de sintaxis), aunque podríamos hacer las comprobaciones con facilidad. Es cierto que esto no será lo normal, pero siempre habrá posibilidad de mejorar nuestro compilador más adelante y tratar los posibles errores en versiones posteriores.

Lo que tendremos que hacer es leer cada línea del programa y *traducirla* a su equivalente en código máquina siguiendo las instrucciones de abajo.

Nota: se separa cada byte de las instrucciones en código máquina para que sea fácil verlas, pero en realidad estarían juntas en una larga ristra de bytes.

1.d Instrucciones

imprime RZ

Esta instrucción imprime el contenido del registro Z como número. Para que la instrucción sea correcta Z debe ser un número entre 0 y 7.

Para compilarla correctamente la Z debe convertirse a binario (0 = 000, 1 = 001, 2 = 010 ...). La conversión en binario ocupará 3 bits XXX.

Se traduce como: 00000XXX --------

Recuerda: las instrucciones siempre ocupan 2 bytes, el segundo byte se ignorará al ejecutarlo, por lo que puede contener cualquier cosa. Nosotros siempre cambiaremos los - por 0. XXX es el número Z en binario.

Por ejemplo:

imprime R3 → 00000XXX -------- → 00000011 00000000
imprime R7 → 00000111 00000000

También hay varias opciones que serían consideradas válidas en otros compiladores (recuerda que nosotros siempre usaremos 0 para rellenar):
imprime R7 → 00000111 00000000
imprime R7 → 00000111 11111111
imprime R7 → 00000111 10101010

imprimec RZ

Esta instrucción imprime el contenido del registro Z como un caracter. Para que la instrucción sea correcta Z debe ser un número entre 0 y 7.

Para compilarla correctamente la Z debe convertirse a binario (0 = 000, 1 = 001, 2 = 010 ...). La conversión en binario ocupará 3 bits XXX.

Se traduce como: 00001XXX --------

Recuerda: las instrucciones siempre ocupan 2 bytes, el segundo byte se ignorará al ejecutarlo, por lo que puede contener cualquier cosa. XXX es el número Z en binario. Nosotros siempre cambiaremos los - por 0.

Por ejemplo:

imprimec R3 → 00001XXX -------- → 00001011 00000000
imprimec R7 → 00001111 00000000
imprimec R5 → 00001101 00000000
imprimec R1 → 00001001 00000000

valor RZ VALOR

Esta instrucción introduce el valor VALOR dentro del registro Z. Para que la instrucción sea correcta Z debe ser un número entre 0 y 7 y VALOR un número entre 0 y 255.

Para compilarla correctamente la Z debe convertirse a binario (0 = 000, 1 = 001, 2 = 010 ...) y también debe convertirse a binario VALOR. La conversión en binario de Z ocupará 3 bits XXX y la conversión de VALOR ocupará 8 bits YYYYYYYY.

Se traduce como: 00010XXX YYYYYYYY

Recuerda: las instrucciones siempre ocupan 2 bytes. XXX es el número Z en binario y VALOR en binario será YYYYYYYY.

Por ejemplo:

valor R3 5 → 00010XXX YYYYYYYY → 00010011 00000101
valor R7 255 → 00010111 11111111
valor R0 255 → 00010000 11111111

borra RZ

Esta instrucción borra el valor del registro Z. Lo que hará en realidad será introducir 0 en el registro Z, será por lo tanto equivalente a la instrucción: valor RZ 0
Para que la instrucción sea correcta Z debe ser un número entre 0 y 7.

Se traduce como: 00011XXX --------
Borra el contenido del registro XXX (lo pone a 0). El segundo byte se ignora.
Nosotros siempre cambiaremos los - por 0.

Por ejemplo:

borra R3 → 00011XXX -------- → 00011011 00000000
borra R7 → 00011111 00000000
borra R0 → 00011000 00000000

suma RZ VALOR

Esta instrucción suma el valor VALOR al contenido del registro Z. Para que la instrucción sea correcta Z debe ser un número entre 0 y 7 y VALOR un número entre 0 y 255.

Para compilarla correctamente la Z debe convertirse a binario (0 = 000, 1 = 001, 2 = 010 ...) y también debe convertirse a binario VALOR. La conversión en binario de Z ocupará 3 bits XXX y la conversión de VALOR ocupará 8 bits YYYYYYYY.

Se traduce como: 00100XXX YYYYYYYY
Suma el valor YYYYYYYY al registro XXX.

Recuerda: las instrucciones siempre ocupan 2 bytes. XXX es el número Z en binario y VALOR en binario será YYYYYYYY.

Por ejemplo:

suma R3 1 → 00100XXX YYYYYYYY → 00100011 00000001
suma R7 15 → 00100111 00001111
suma R0 255 → 00100000 11111111

resta RZ VALOR

Esta instrucción resta el valor VALOR al contenido del registro Z. Para que la instrucción sea correcta Z debe ser un número entre 0 y 7 y VALOR un número entre 0 y 255.

Para compilarla correctamente la Z debe convertirse a binario (0 = 000, 1 = 001, 2 = 010 ...) y también debe convertirse a binario VALOR. La conversión en binario de Z ocupará 3 bits XXX y la conversión de VALOR ocupará 8 bits YYYYYYYY.

Se traduce como: 00101XXX YYYYYYYY
Resta el valor YYYYYYYY al registro XXX.

Recuerda: las instrucciones siempre ocupan 2 bytes. XXX es el número Z en binario y VALOR en binario será YYYYYYYY.

Por ejemplo:

resta R3 1 → 00101XXX YYYYYYYY → 00101011 00000001
resta R7 15 → 00101111 00001111
resta R0 255 → 00101000 11111111

salta POSICIÓN

Esta instrucción "salta" a la instrucción que está en la posición POSICIÓN (es decir, esa será la siguiente instrucción a ejecutar). Para que la instrucción sea correcta POSICIÓN debe ser un número entre 0 y 255. La primera instrucción ocupará la posición 0, la segunda instrucción la posición 1 y así sucesivamente.

Para compilarla correctamente la POSICIÓN debe convertirse a binario. La conversión en binario de POSICIÓN ocupará 8 bits YYYYYYYY.

Se traduce como: 00110--- YYYYYYYY
Salta a la instrucción YYYYYYYY

Nosotros siempre cambiaremos los - por 0.

Por ejemplo:

salta 0 → 00110--- YYYYYYYY → 00110000 00000000
salta 15 → 00110000 00001111
salta 255 → 00110000 11111111

saltasi0 RZ POSICIÓN

Esta instrucción "salta" a la instrucción que está en la posición POSICIÓN (es decir, esa será la siguiente instrucción a ejecutar) si el contenido del registro Z es 0 . Para que la instrucción sea correcta POSICIÓN debe ser un número entre 0 y 255 y Z debe ser un número entre 0 y 7. La primera instrucción ocupará la posición 0, la segunda instrucción la posición 1 y así sucesivamente.

Para compilarla correctamente la POSICIÓN y Z deben convertirse a binario. La conversión en binario de POSICIÓN ocupará 8 bits YYYYYYYY y la de Z ocupará bits XXX.

Se traduce como: 00111XXX YYYYYYYY
Salta a la instrucción YYYYYYYY si el contenido del registro XXX es 0.

Por ejemplo:

saltasi0 R0 0 → 00111XXX YYYYYYYY → 00111000 00000000
saltasi0 R7 15 → 00111111 00001111
saltasi0 R2 255 → 00111010 11111111

1.e Ejemplos de programas

Imprime los números del 0 al 4

Imprimirá todos los números en una única línea:

borra R1
imprime R1
suma R1 1
imprime R1
suma R1 1
imprime R1
suma R1 1
imprime R1
suma R1 1
imprime R1
salta 255

Resultado

0001100100000000000000010000000000100001000000010000000100000000001000010000000100000001000000000010000100000001000000010000000000100001000000010000000011000011111111

Separando por instrucción y bytes:

```
borra R1:   00011001 00000000
imprime R1: 00000001 00000000
suma R1 1:  00100001 00000001
imprime R1: 00000001 00000000
suma R1 1:  00100001 00000001
imprime R1: 00000001 00000000
suma R1 1:  00100001 00000001
imprime R1: 00000001 00000000
suma R1 1:  00100001 00000001
imprime R1: 00000001 00000000
salta 255:  00110000 11111111
```

Imprime los números del 20 al 50

Imprimirá cada número en una línea diferente desde el 20 al 50 sin incluir éste último. Se usa R7 para imprimir los retornos de carro. Se crea un bucle usando los saltos y R1 como índice.

valor R7 10
valor R0 30
valor R1 20
imprime R1
suma R1 1
imprimec R7
resta R0 1
saltasi0 R0 255
salta 3

Resultado

0001011100001010000100000001111000010001000101000000000100000000001000010000000100001111000000000010100000000000100111000111111110011000000000011

Separando por instrucción y bytes:

```
valor R7 10:     00010111 00001010
valor R0 30:     00010000 00011110
valor R1 20:     00010001 00010100
imprime R1:      00000001 00000000
suma R1 1:       00100001 00000001
imprimec R7:     00001111 00000000
resta R0 1:      00101000 00000001
saltasi0 R0 255: 00111000 11111111
salta 3:         00110000 00000011
```

Imprime Hola

```
valor R0 72
imprimec R0
valor R0 111
imprimec R0
valor R0 108
imprimec R0
valor R0 97
imprimec R0
salta 255
```

Resultado

```
00010000010010000000100000000000000100000110111100001000000000000010000
01101100000010000000000000010000011000010000100000000000011000011111111
```

Imprime las letras de la A a la Z

```
valor R0 26
valor R1 65
imprimec R1
suma R1 1
imprimec R7
resta R0 1
saltasi0 R0 255
salta 2
```

Resultado

```
00010000000110100001000101000001000010010000000000100001000000010000111 1
00000000001010000000000100111000111111110011000000000010
```

Comparador

Compara los valores contenidos en R0 y R1 e indica si R0 es mayor (>), menor (<) o igual (=) que R1.

```
valor R0 50
valor R1 25
valor R6 63
imprime R0
imprimec R6
imprime R1
valor R6 10
imprimec R6
saltasi0 R0 13
saltasi0 R1 17
resta R0 1
resta R1 1
salta 8
saltasi0 R1 20
valor R6 60
imprimec R6
salta 255
valor R6 62
imprimec R6
salta 255
valor R6 61
imprimec R6
salta 255
```

Resultado

```
00010000001100100001000100011001000101100011111100000000000000000000011100000000000000000100000000000010110000010100000111000000000000111000000011010011100100010001001010000000000100101001000000010011000000001000001110010001010000010110001111000000111000000000001100001111111100010110001111101000011100000000000110000111111111
```

2. Parte 2: Ejecutando programas con un intérprete

Vamos a ejecutar código máquina del microcontrolador ficticio procSMR2, un procesador de 8 bits con 8 registros de propósito general:
R0, R1, R2, ...R7 (en binario el registro 000, el 001, el 010, ... hasta el 111).

Supongamos también que para simplificar, todas las instrucciones del procesador ocupan 2 bytes y que el programa más grande que cabe en memoria es de 512 bytes (256 instrucciones). Un programa empezará a ejecutarse en la instrucción 0 y terminará cuando se llegue a la instrucción 255 que se considerará la instrucción de fin.

Nota: En principio los registros del microcontrolador se considerará que no contienen números negativos, en revisiones futuras se podría plantear representar los números negativos con exceso a 128 (es decir, podrímos representar números entre -128 y 127). Se hace así en esta versión por simplificar.

2.a Instrucciones

Esta es la lista de instrucciones que puede ejecutar el procesador:

00000XXX --------
Imprime como número decimal el contenido del registro XXX. El segundo byte se ignora

00001XXX --------
Imprime como caracter el contenido del registro XXX. El segundo byte se ignora

00010XXX YYYYYYYY
Introduce el valor YYYYYYYY en el registro XXX

00011XXX --------
Borra el contenido del registro XXX (lo pone a 0). El segundo byte se ignora.

00100XXX YYYYYYYY
Suma el valor YYYYYYYY al contenido del registro XXX.
Si el resultado del contenido del registro más el valor es mayor que 255, el contenido del registro será 255.

Nota: en versiones posteriores del lenguaje este comportamiento de la instrucción será revisado.

Ejemplos:
RX + Y = 20 + 1 = 21
RX + Y = 20 + 50 = 70
RX + Y = 127 +128 = 255
RX + Y = 255 +103 = 358 → 255
RX + Y = 128 + 128 = 256 → 255

00101XXX YYYYYYYY
Resta el valor YYYYYYYY al registro XXX.
Si el resultado del contenido del registro menos el valor es menor que 0, el contenido del registro será 0.

Nota: en versiones posteriores del lenguaje este comportamiento de la instrucción será revisado.

Ejemplos:
RX - Y = 20 - 1 = 19
RX - Y = 50 - 50 = 0
RX - Y = 127 - 128 = -1 → 0
RX - Y = 250 -100 = 150

00110XXX YYYYYYYY
Salta a la instrucción YYYYYYYY

00111XXX YYYYYYYY
Salta a la instrucción YYYYYYYY si el contenido del registro XXX es 0.

2.b Ejemplos de programas

Si ejecutamos los programas de ejemplo obtendremos esto.

Imprime los números del 0 al 4

```
borra R1
imprime R1
suma R1 1
imprime R1
suma R1 1
imprime R1
suma R1 1
imprime R1
suma R1 1
imprime R1
salta 255
```

Se compiló en:

```
0001100100000000000000010000000001000010000000100000001000000000010000100000001
0000000100000000001000010000000100000001000000000010000100000001000000010000000
0011000011111111
```

Y el resultado de la ejecución fue:

```
01234
```

Imprime los números del 20 al 50

```
valor R7 10
valor R0 30
valor R1 20
imprime R1
suma R1 1
imprimec R7
resta R0 1
saltasi0 R0 255
salta 3
```

Se compiló en:

```
0001011100001010000100000001111000010001000101000000000100000000000100001000000001
0000111100000000001010000000000100111000111111110011000000000011
```

Y el resultado de la ejecución fue:

```
20
21
```

```
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

Imprime Hola

```
valor R0 72
imprimec R0
valor R0 111
imprimec R0
valor R0 108
imprimec R0
valor R0 97
imprimec R0
salta 255
```

Se compiló en:

```
00010000010010000000100000000000000100000110111100001000000000000001000001101100
00001000000000000000100000110000100001000000000000011000011111111
```

Y el resultado de la ejecución fue:

```
Hola
```

Imprime las letras de la A a la Z

```
valor R0 26
valor R1 65
imprimec R1
suma R1 1
imprimec R7
resta R0 1
saltasi0 R0 255
salta 2
```

Se compiló en:

```
00010000000110100001000101000001000010010000000000100001000000010000111100000000
00101000000000010011100011111111001100000000010
```

Y el resultado de la ejecución es:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Comparador

Compara los valores contenidos en R0 y R1 e indica si R0 es mayor (>), menor (<) o igual (=) que R1.

```
valor R0 50
valor R1 25
valor R6 63
imprime R0
imprimec R6
imprime R1
valor R6 10
imprimec R6
saltasi0 R0 13
saltasi0 R1 17
resta R0 1
resta R1 1
salta 8
saltasi0 R1 20
valor R6 60
imprimec R6
salta 255
valor R6 62
imprimec R6
salta 255
valor R6 61
imprimec R6
salta 255
```

Se compiló en:

```
00010000001100100001000100011001000101100011111100000000000000000000011100000000
00000001000000000001011000001010000011100000000000111000000011010011100100010001
00101000000000010010100100000001001100000000100000111001000101000001011000111100
00001110000000000011000011111111000101100011111000001110000000000011000011111111
00010110001111010000111000000000011000011111111
```

Y el resultado de la ejecución es:

```
50?25
>
```

3. Ejemplo de web

En la imagen siguiente se puede ver un ejemplo de cómo se podría crear la web para compilar e interpretar el código que se ha escrito.

Como puede verse, hay dos textarea en la web en la que se puede escribir. En la primera se puede escribir el código en lenguaje SMR2 que será compilado a código máquina al pulsar el botón Generar código. El código generado aparecerá en el segundo textarea.

En la segunda textarea podemos introducir código máquina del microcontrolador procSMR2 (o dejar el generado al compilar). Cuando se pulse el botón de Ejecutar, se recogerá el código máquina en la segunda textarea y se ejecutará, mostrando el resultado de la ejecución en una división que está abajo.

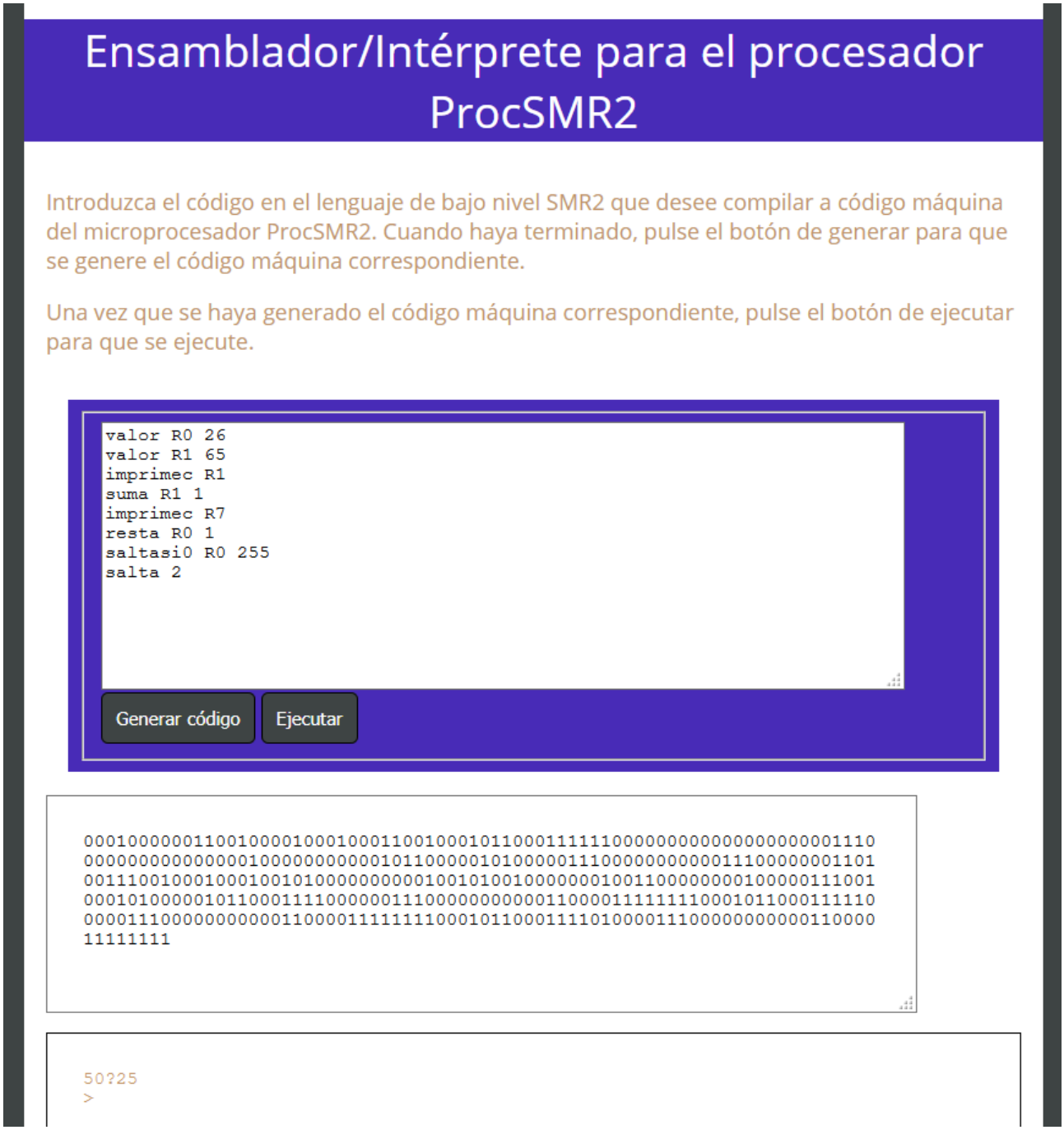

Figure 2: Ejemplo de compilador/intérprete web del lenguaje SMR2 en el microcontrolador SMR2