Desensamblador AVR

Autor: **Nahuel Agustin Tapia**
Contacto: **nahuelat.8@gmail.com**
GitHub: **github.com/Vortex842**

Versión del proyecto: **v.1.5.7**
Tecnología usada: 

Este proyecto fue desarrollado con el objetivo de ofrecer una herramienta intuitiva que facilite al usuario la visualización de archivos en formato Intel HEX, convirtiéndolos a un formato más legible y conocido como el Assembler de AVR.
Su uso está orientado tanto a estudiantes como a desarrolladores de firmware que deseen analizar, depurar o estudiar el contenido real de un archivo HEX generado por un compilador AVR.

Caso de uso
Analizar qué instrucciones está generando el compilador a partir de un código en lenguaje C.
Esto permite comprender cómo se traduce el código de alto nivel a instrucciones ensamblador, facilitando el aprendizaje, la optimización o la depuración del programa.

Formato **Intel HEX**

El **Hexadecimal Object File Format** es un formato de archivo para la programación de microcontroladores, EPROMs y otros circuitos integrados. Es uno de los formatos más antiguos con esta finalidad.

Consiste en un archivo de texto cuyas líneas contienen valores hexadecimales en ASCII que codifican los datos y su offset o dirección de memoria.

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

- Inicio
- Cantidad de datos
- Dirección de inicio de datos
- Tipo de datos
- Datos
- Checksum

Se calcula sumando todos los bytes de la línea, excluyendo los dos puntos iniciales (:) y la propia suma de verificación, luego tomando el complemento a dos del resultado.

¿Cómo funciona?

ADD – Add without Carry

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

Instrucción (bin): 0000 11rd dddd rrrr

Mascara: 0000 1100 0000 0000 ⟶ FC00

Patrón: 0000 1100 0000 0000 ⟶ 0C00

d : [4, 5, 6, 7, 8] r : [0, 1, 2, 3, 9]

Instrucción (asm): ADD R{d}, R{r}

Ejemplo 1: 0x0C8A ⟶ ADD R8, R10

0000 1100 1000 1010

ANDI – Logical AND with Immediate

(i)	ANDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0111	KKKK	dddd	KKKK

Mascara: 0111 0000 0000 0000 ⟶ 7000

Patrón: 0111 0000 0000 0000 ⟶ 7000

Ejemplo 2: 0x7120 ⟶ ANDI R18, 0x10

0111 0001 0010 0000

Ejemplo:
Sucesión de Fibonacci

fibonacci_p2 (Microchip Studio)

```
// Posicion de memoria inicial
// Y = 0x200
ldi YL, 0x00
ldi YH, 0x02


// Inicializo los primeros 2 numeros "previo" y "siguiente"
ldi prevL, 1
ldi nextL, 0
ldi i, 1

fibo:   st Y, nextH         // m[Y] = nextH
        std Y+1, nextL      // m[Y] = nextL

        add nextL, prevL    // nextL += prevL
        adc nextH, prevH    // nextH += prevH + C

        ld prevH, Y+        // prevH = m[Y], Y++
        ld prevL, Y+        // prevL = m[Y], Y++

        // Verificar que sean los 20 elementos
        inc i
        cpi i, 21
        brlo fibo       // i <= 20 ---> i < 20 + 1 ---> i < 21
```

fibonacci_p2.hex

```
:020000020000FC
:10000000C0E0D2E001E020E041E038832983200F06
:0C001000311F1991099143954531B8F357
:00000001FF
```

fibonacci_p2 (Desensamblado)

```
Instrucciones desensambladas:
0x0000: LDI R28, 0x00    // 0xE0C0
0x0002: LDI R29, 0x02    // 0xE0D2
0x0004: LDI R16, 0x01    // 0xE001
0x0006: LDI R18, 0x00    // 0xE020
0x0008: LDI R20, 0x01    // 0xE041
0x000A: ST Y, R19        // 0x8338
0x000C: STD Y+q, R18     // 0x8329
0x000E: ADD R18, R16     // 0x0F20
0x0010: ADC R19, R17     // 0x1F31
0x0012: LD R17, Y+       // 0x9119
0x0014: LD R16, Y+       // 0x9109
0x0016: INC R20 // 0x9543
0x0018: CPI R20, 21      // 0x3145
0x001A: BRLO 0x77        // 0xF3B8
```