Respuestas a los ejercicios de ensamblador

1. ¿Qué hace el siguiente código ensamblador?

```
OFFSET 0
BYTE ARRAY arreglo 5

OFFSET 10
        MOV R0, 0
        MOV R1, 1
        MOV R5, 5
inicia: GRE R0, R5
        JIF salida
        MOV BYTE [R0 + arreglo], R0
        ADD R0, R0, R1
        JMP inicia
salida: NOP
```

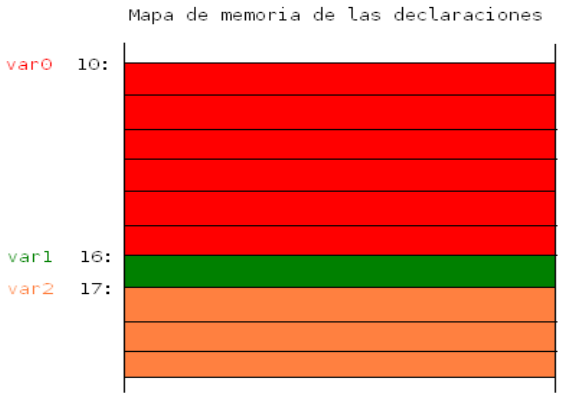
Visto en clase, en un ciclo inicializa el arreglo con los valores { 0, 1, 2, 3, 4 }

2. Traduzca el siguiente código ensamblador a código máquina y muestre la tabla de etiquetas

```
OFFSET 10
WORD ARRAY var0 2
BYTE var1
WORD var2

OFFSET 100
        MOV R0, 0
        NOP
        MOV R0, 128
final:  JMP final          # codificar usando direccionamiento relativo a RP
```

La etiqueta var0 representa un arreglo de 2 palabras con dirección inicial 10.
La etiqueta var1 representa un byte con dirección inicial 16.
La etiqueta var2 representa una palabra con dirección inicial 17.



MOV R0, 0 — *se puede codificar usando la instrucción "MOV ra, entero" donde el entero puede representarse con 8 bits, por lo que se ensambla como 1001000000000000 (y ocupa dos bytes)*

NOP — *se ensambla directamente como 00000000 (y ocupa un byte)*

MOV R0, 128 — *usando codificación en complemento a dos se necesitan más de 8 bits para representar 128, por lo que se debe usar la instrucción "MOV ra, entero" donde el entero se codifica en 24 bits. El entero debe codificarse en little endian:*

00000000 00000000 10000000 = 128 en binario
10000000 00000000 00000000 = 128 en binario, little endian

La instrucción ocupa 4 bytes y quedaría ensamblada como 10011000100000000000000000000000

¿Qué número es 10000000 (usando 8 bits) en complemento a dos? Para averiguarlo pruebe correr el siguiente programa en C:

```
signed char c = 128;
signed int n = c;

printf("%d", n);
```

final: JMP final — *La etiqueta "final" tiene la dirección 107 (100 + 2 + 1 + 4). Sin embargo por un error mío la indicación no es suficientemente clara. La idea era que, codificada como direccionamiento absoluto, "final : JMP final" es un ciclo infinito. ¿Cómo se puede codificar un ciclo infinito usando direccionamiento relativo a RP?*

```
JMP -2          # ¿por qué? pista: JMP con dir. relativo ocupa 2 bytes
```

3. Compare las siguientes instrucciones. ¿Qué hacen y cuál es mejor?

```
NOP
MOV RP, RP
JMP 0                   # codificado usando direccionamiento relativo a RP
```

Visto en clase, ninguna de las tres realiza trabajo útil. NOP ocupa menos espacio al traducirlo a código máquina.