Instrucciones

TOY-8

Este lenguaje máquina ficticio consta de ocho instrucciones que se resumen en la siguiente tabla.

opcode	**hexadecimal**	**nombre**	**pseudocódigo**
0000	0	`halt`	
0010		`add`	
0100		`and`	`R = R & M[addr]`
0110		`xor`	
1000		`load address`	`R = addr`
1010		`load`	
1100		`store`	`M[addr] = R`
1110		`branch if zero`	`if (R == 0) PC = addr`

El formato de instrucción es el siguiente. Contando desde la derecha, los primeros 3 bits son el *opcode* (código de operación). El cuarto bit no se usa y se deja en cero siempre. Los últimos 4 bits especifican una dirección en una memoria de 16 bytes.

Algunas convenciones que el programador de esta máquina conoce.

- El registro `R` es el único registro de propósito general y está implicito en todas las instrucciones, salvo `halt` que para la computadora.
- La dirección de memoria 0x0 siempre vale cero.
- La dirección de memoria 0xF está conectada a `stdin` y `stdout`. Esto es una forma de *memory mapped I/O* (entrada y salida mapeada a memoria).
- Hay un PC por supuesto, de 4 bits de ancho.

Ejercicios

1. Completar lo que falta de la tabla de instrucciones.
2. ¿Esta computadora es direccionable por byte o por palabra? ¿Cuál es el tamaño de palabra?
3. Los nombres de las instrucciones en la tabla se pueden usar para traducir el lenguaje máquina a un lenguaje ensamblador. Traduzca el siguiente programa escrito en lenguaje máquina. La primer columna indica la dirección de memoria de cada instrucción.

```
0x1  A0
0x2  CE
0x3  AF
0x4  E9
0x5  2E
0x6  CE
0x7  A0
0x8  E3
0x9  AE
0xA  CF
0xB  00
```

4. ¿Qué es lo que hace el programa anterior? Intente escribir el mismo programa en C. ¿Es más corto o más largo el programa en un lenguaje de alto nivel?
5. ¿Cómo podría modificar esta computadora para tener 32 bytes de memoria en vez de 16?
6. Esto es un *memory dump* (volcado de memoria) de la computadora. ¿Cuántos programas distintos hay en la memoria?

```
      00  01  10  11

00    00  A5  26  C7
01    00  08  05  00
10    A7  6D  2E  C7
11    00  FF  01  00
```

7. Escribir un programa que sume dos números ya almacenados en memoria.
8. Escribir un programa que reste dos números ya almacenados en memoria.
9. Escribir un programa que imprima “Hola” en una pantalla. Imagine que `stdout` de esta máquina está conectado a una pantalla que entiende código ASCII y que cada byte que se envía no sobreescribe el anterior.
10. Escribir un programa que duplique un número que ingrese el usuario.