Instrucciones

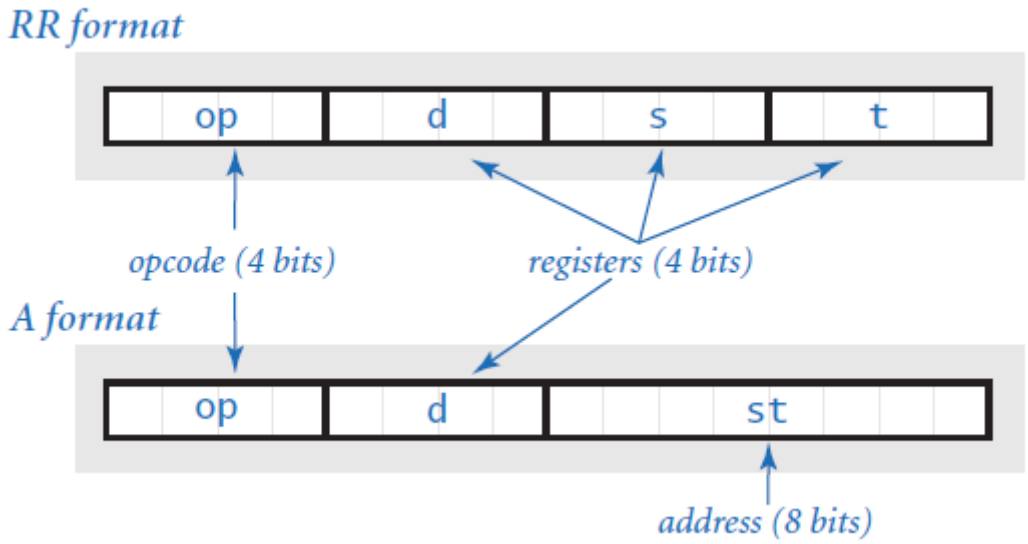
TOY-16

Este lenguaje máquina ficticio es una extensión del lenguaje de la actividad anterior. Ahora tenemos una computadora con una palabra de 16 bits, 16 registros de propósito general y una memoria principal de 256 palabras de 16 bits o 512 bytes.

Las instrucciones de TOY-16 son las siguientes:

opcode	**hex**	**nombre**	**formato**	**pseudocódigo**
0000		`halt`	-	
0001		`add`		
0010		`sub`	RR	`R[d] = R[s] - R[t]`
0011		`and`		
0100		`xor`		
0101		`lshift`	RR	
0110		`rshift`		`R[d] = R[s] >> R[t]`
0111		`la`	A	`R[d] = addr`
1000		`lw`		
1001		`sw`	A	
1010	A	`li`	RR	`R[d] = M[R[t]]`
1011	B	`si`	RR	`M[R[t]] = R[d]`
1100		`bz`	A	
1101		`bp`		`if (R[d] > 0 PC) = addr`
1110		`jr`	-	`PC = R[d]`
1111		`jal`	A	`R[d] = PC; PC = addr`

Tener en cuenta que el registro cero siempre vale cero y que la dirección de memoria FF la usamos como E/S estándar. Para indicar un registro usamos notación de *array* como hacemos con la memoria. Hay 16 registros, desde `R[0]` a `R[F]`. Para las direcciones de memoria hay que usar un byte entero como dirección, por ejemplo `M[1B]`. Las instrucciones de esta máquina vienen en dos formatos distintos: A y RR. Siempre los primeros 4 bits desde la izquierda son el *opcode*. Los 3 dígitos hexadecimales que quedan corresponden a tres registros o a un registro y una dirección de memoria. El formato RR (*register - register*) usa tres registros: *destination*, *source* y *temp*. El formato A usa un registro para el destino y un byte entero para la dirección de memoria (*address*).



Ejercicios

1. Completar la tabla de instrucciones.
2. Reescribir en este lenguaje el programa de la guía anterior que sumaba dos números ya almacenados en memoria.
3. Dar una instrucción que ponga el PC en 15.
4. Dar 7 instrucciones diferentes, usando distintos *opcodes*, que pongan en cero el registro A.
5. Dar 3 instrucciones diferentes que pongan en cero el PC sin cambiar los contenidos de los registros o de la memoria.
6. Dar 5 maneras diferentes de hacer una **NOP** (*no operation*) sin poner 0 en el segundo *nibble* desde la izquierda.
7. ¿Cómo se puede hacer un salto a la dirección 15 si `R[a] >= 0`?
8. Llenar la siguiente tabla.

binario	hex	instrucción
0001001000110100	1234	`R[2] = R[3] + R[4]`
1111111111111111		
1111101011001110		
0101011001000100		
1000000000000001		
0101000001000011		
0001110010101011		
		`R[F] = R[F] & R[F]`
		`R[8] = M[88]`
	7777	

binario	hex	instrucción
		`if (R[C] == 0) PC = CC`

9. Dar una secuencia de instrucciones para calcular el valor absoluto de `R[s]` y ponerlo en `R[d]`
10. ¿Cómo realizar un OR bitwise?
11. ¿Cómo realizar un NAND bitwise?
12. ¿Cómo hacer un NOT bitwise?
13. Escribir un programa que implemente el algoritmo de Euclides para calcular el MCD de dos números ya almacenados en memoria. Sugerencia, escribirlo en C primero sin usar la división y usando un `while` y un `if else`
14. Una consecuencia del modelo de computadora de Von Neumann es que un programa puede modificarse a sí mismo. Escribir un programa que se modifique a sí mismo para sumar una lista de números ubicada en memoria de manera similar a un *array*.
15. Implementar un programa que decida si un número en memoria es primo o no. Usar el programa del ejercicio 13 como una función y llamarla desde este programa.