Práctico 6

Formato de instrucción. Manejo de instrucciones

Objetivo: comprender cómo se define un set de instrucciones y los formatos de instrucción. Implementar pequeñas rutinas utilizando las instrucciones definidas.

Notas

- Se realizará en clase el ejercicio 1 y se publicará la solución del ejercicio 3.

Preguntas teóricas

1. Explique el propósito de los registros IP e IR.
2. ¿Qué implica decir que una arquitectura es de n bits? ¿Los tamaños de cuáles recursos de hardware quedan determinados?
3. Indique al menos tres aspectos fundamentales a tener en cuenta en el diseño de máquinas RISC.

Ejercicio 1 (Difícil)

Considere un microprocesador RISC de 16 bits con 16 registros y las siguientes instrucciones:

Instrucción	Descripción
`NOP`	No realiza ninguna operación
`MOVI Inm, Reg1`	Carga el inmediato Inm en los bits menos significativos de Reg1, dejando los bits más significativos en 0
`MOVR Reg1, Reg2`	Carga en Reg2 el contenido de Reg1
`LOAD Reg1, Reg2`	Carga el contenido de memoria apuntado por Reg1 en Reg2
`SAVE Reg1, Reg2`	Guarda el registro Reg1 en la dirección de memoria apuntada por Reg2
`ADD Reg1, Reg2, Reg3`	Suma Reg1 y Reg2 y guarda el resultado en Reg3
`SUB Reg1, Reg2, Reg3`	Realiza la operación Reg1-Reg2 y guarda el resultado en Reg3
`SHL Reg1, Reg2, Reg3`	Desplaza Reg1 a la izquierda Reg2 lugares y guarda el resultado en Reg3
`SHR Reg1, Reg2, Reg3`	Desplaza Reg1 a la derecha Reg2 lugares y guarda el resultado en Reg3
`AND Reg1, Reg2, Reg3`	Realiza un AND bit a bit entre Reg1 y Reg2 y guarda el resultado en Reg3
`OR Reg1, Reg2, Reg3`	Realiza un OR bit a bit entre Reg1 y Reg2 y guarda el resultado en Reg3
`XOR Reg1, Reg2, Reg3`	Realiza un XOR bit a bit entre Reg1 y Reg2 y guarda el resultado en Reg3
`NOT Reg1, Reg2`	Realiza un NOT bit a bit de Reg1 y guarda el resultado en Reg2
`JMP Reg1`	Hace un salto incondicional a la dirección apuntada por Reg1
`JC Reg1`	Hace un salto a la dirección apuntada por Reg1 si la bandera de C vale 1
`JZ Reg1`	Hace un salto a la dirección apuntada por Reg1 si la bandera

	de Z vale 1

Aclaraciones:

- Las flags Z (cero) y C (acarreo) son afectadas por las operaciones aritméticas y lógicas.
- Para el caso de los desplazamientos, la flag C corresponde al último bit que fue desplazado hacia fuera del registro.
- Los 16 registros llevan el nombre R00 hasta R15.

A. Defina un formato de instrucción para la CPU e indique el largo de instrucción.
B. Implemente en alto nivel un programa que sume el contenido de memoria de las direcciones comprendidas en el rango 0xABCD–0xEF00 y almacene en la dirección 0x00FF el triple de dicha suma.
C. Implemente en lenguaje ensamblador el programa de la parte B.
D. Implemente en lenguaje máquina las primeras 7 instrucciones del programa de la parte C.

Notas:

- El programa se debe implementar con los nombres de operación dados. Las direcciones y constantes deben representarse en hexadecimal.
- El programa estará cargado en memoria a partir de la dirección 0x0000.

Ejercicio 2 (Básico)

Sea una CPU de 16 bits que tiene 8 registros de uso general de 16 bits y en la que las instrucciones se codifican en 16 bits. La CPU posee las siguientes instrucciones:

Instrucción	**Descripción**
`LOAD REG1, REG2`	Guarda en REG2 el contenido de la dirección de memoria almacenada en REG1
`STORE REG1, REG2`	Escribe el contenido de REG1 en la dirección de memoria almacenada en REG2.
`NOP`	No realiza ninguna operación.
`SETLO INM, REG`	Carga el valor constante INM de 8 bits en la parte baja de REG
`SETHI INM, REG`	Carga el valor constante INM de 8 bits en la parte alta de REG.
`ADD REG1, REG2, REG3`	Suma REG1 y REG2, guarda el resultado en REG3
`SUB REG1, REG2, REG3`	Resta REG2 a REG1, guarda el resultado en REG3
`AND REG1, REG2, REG3`	And bit a bit entre REG1 y REG2, guarda el resultado en REG3
`OR REG1, REG2, REG3`	Or bit a bit entre REG1 y REG2, guarda el resultado en REG3

`NOT REG1, REG2`	Guarda en REG2 el resultado de hacerle un NOT al REG1
`SL REG1, INM, REG2`	Shift a la izquierda de tantos bits de REG1 como indique el inmediato INM (de 4 bits) y guarda el resultado en REG2
`SR REG1, INM, REG2`	Shift a la derecha de tantos bits de REG1 como indique el inmediato INM (de 4 bits) y guarda el resultado en REG2. *Expande el signo*.
`CMP REG1, REG2`	Instrucción de comparación que realiza la resta REG1-REG2 sin guardar el resultado y actualiza las banderas de condición.
`JZ INM`	Si la bandera Z está en 1 salta INM (INM es un número con signo de 12 bits) instrucciones desde la posición actual
`JN INM`	Si la bandera N está en 1 salta INM (INM es un número con signo de 12 bits) instrucciones desde la posición actual
`JMP INM`	Salto incondicional de INM instrucciones (INM es un número con signo de 12 bits) desde la posición actual.

Diseñar el formato de instrucción presentado e indicar la codificación de cada instrucción.

Ejercicio 3 (Difícil)

La rutina `BBinaria` implementa el algoritmo de búsqueda binaria. Dada una palabra en la dirección de memoria 100h y una secuencia de 100 palabras ordenadas en memoria a partir de la dirección 101h hasta la dirección 165h, determina si la palabra pertenece a la secuencia, y en tal caso escribe en la posición 166h el valor 1, o en caso contrario escribe el valor 0.

```
void BBinaria () {
      int inf, sup, medio, dato;
      bool esta;

      esta = false;
      inf = 0x101;
      sup = 0x165;
      dato = memoria[0x100];

      do {
            medio = (inf + sup) / 2;
            if (dato == (memoria[medio])) {
                  esta = true;
            } else if (dato < memoria[medio]) {
                  sup = medio - 1;
            } else {
                  inf = medio + 1;
            }
      } while (! (esta || (inf > sup)));
      if (esta) {
            memoria[0x166] = 1;
      } else {
```

```
            memoria[0x166] = 0;
      }
}
```

Compilar la rutina `BBinaria` en el lenguaje ensamblador del Ejercicio 2.

Ejercicio 4 (Medio)

Considere un microprocesador de 32 bits con 16 registros y el siguiente conjunto de instrucciones:

`NOP`	`RESTA Reg1, Reg2`	`NOT Reg`
`CARGARM Reg1, Reg2`	`SHIFTL Reg1, Reg2`	`JUMP Reg`
`CARGARI Reg`	`SHIFTR Reg1, Reg2`	`JE Reg`
`SALVAR Reg1, Reg2`	`AND Reg1, Reg2`	`JG Reg`
`SUMA Reg1, Reg2`	`OR Reg1, Reg2`	

Aclaraciones:

- En las operaciones que involucran memoria, el segundo registro debe contener una dirección de memoria.
- En las operaciones aritméticas o lógicas, el resultado se almacena en el primer registro.
- En las operaciones de desplazamiento, el segundo registro representa el desplazamiento.
- JE salta si Z = 0 (jump equal) y JG salta si Z ≠ 0 y N > 0 (jump greater)

A. Implemente en alto nivel el algoritmo de ordenación por selección (selection sort), que dadas 10 palabras de memoria a partir de la dirección 100h las escriba en forma ordenada a partir de la dirección 10Ah. El algoritmo debe buscar en cada iteración la palabra más grande del conjunto y copiarla a la última posición del nuevo conjunto. Mediante este procedimiento la última palabra ya queda ordenada, por lo cual en la siguiente iteración se debe considerar el conjunto de la iteración anterior sin considerar la última palabra.
B. Implemente en ensamblador el programa de la parte A, utilizando los nombres de operación dados, y representando las direcciones y constantes en hexadecimal.

Ejercicio 5 (Medio)

Se dispone de un microprocesador con el siguiente conjunto de instrucciones de 32 bits:

`LOAD Reg1,Reg2`	`SHL Reg`	`ADD Reg1,Reg2`
`LOAD #Cte,Reg`	`SHR Reg`	`CMP Reg1,Reg2`
`OR Reg1,Reg2`	`OUT Reg1,Reg2`	`JMPZ #DirRelativa`
`AND Reg1,Reg2`	`IN Reg1,Reg2`	`JMPC #DirRelativa`
`XOR Reg1,Reg2`	`SUB Reg1,Reg2`	`JMP #DirRelativa`

Aclaraciones:

- Se dispone de un conjunto de siete registros de 16 bits (R1 a R7).
- Las instrucciones que operan sobre dos registros almacenan el resultado en el segundo registro.
- Las instrucciones de salto condicional (JMPZ y JMPC) Z consideran las flags Z (cero) y C (acarreo).

- Para leer un byte de memoria se debe realizar OUT a la dirección LEER_MEM1 con el valor de la dirección a leer y luego realizar un IN en la dirección LEER_MEM2.
- Para almacenar un byte de memoria se debe realizar OUT a la dirección ESCRIBIR_MEM1 con el valor de la dirección a escribir y luego realizar OUT en la dirección ESCRIBIR_MEM2 con el byte a escribir.

A. Escribir un programa en alto nivel que almacene en la memoria, desde la dirección 0 en adelante, el código de Gray de 8 bits. La Figura 1 presenta ejemplos de código de Gray para 1, 2 y 3 bits, ejemplificando el procedimiento utilizado para su definición.
B. Implementar el programa de la parte A en lenguaje ensamblador.

1 bit	*2 bits*	*3 bits*
0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100

Figura 1: Ejemplos de código de Gray para 1, 2 y 3 bits.