Trabajo de Laboratorio Número 1

Arquitectura de Computadoras 11/04/2023

Instrucciones

- El laboratorio consta de 7 ejercicios, los cuales no poseen una única solución sino múltiples soluciones posibles.
- Los códigos de los programas deberán respetar la convención de registros y poseer una tabulación adecuada y comentarios. Se permite el uso de pseudoinstrucciones.
- Posteriormente, se tomará una evaluación con problemas similares, la cual deberá ser resuelta en forma individual. La fecha de evaluación se encuentra en el sitio web de la materia.
- Se recomienda disponer siempre en las evaluaciones del libro *Guía Práctica de RISC-V: El Atlas de una Arquitectura Abierta* en formato digital para consulta de los detalles técnicos de las instrucciones en el Apéndice A del mismo.
- Es obligatorio disponer en las evaluaciones del resumen de las instrucciones impresa. El mismo está disponible como tarjeta doble faz en las primeras páginas del mismo libro.

Objetivo

El objetivo del laboratorio es familiarizar a los estudiantes con el set de instrucciones RV32IM de los procesadores RISC-V y con la escritura de programas en el lenguaje ensamblador del mismo, e introducir el programa de simulación *Venus*.

Problemas propuestos

1) Indique qué instrucción RISC-V se representa en el siguiente cuadro:

funct7	rs2	rs1	funct3	rd	opcode
32	9	10	000	11	51

Elija de las siguientes opciones cuál es la correcta.
a) sub x9, x10, x11
b) add x11, x9, x10
c) sub x11, x10, x9
d) sub x11, x9, x10

2) El siguiente bloque de código en C busca el valor mínimo de un vector y lo almacena en una variable.

```
char minimo;
char vector[15] = {54,5,23,65,2,84,1,78,37,97,56,26,48,13,103,18};

void comparar(char* minimo, int indice) {
    if (*minimo > vector[indice]) {
        *minimo = vector[indice];
    }
}

void main(void) {
    minimo = vector[0];
    for(int indice = 1; indice < 16; indice++) {
        comparar(&minimo, indice);
    }
}
```

El mismo se presenta traducido a lenguaje ensamblador del RISC-V

```
            .data
minimo:     .byte 0,0
vector:     .byte 54,5,23,65,2,84,16,78,37,97,56,26,48,13,103,18

            .text
            .globl main
```

```
comparar:   add t0, a1, a2
            lbu t0, 0(t0)
            bgeu t0, a0, retorno
            mv a0, t0
retorno:    jr ra



main:       la a2, vector
            lbu a0, 0(a2)
            li a1, 1
lazo:       sltiu t0, a1, 16
            beq t0, zero, final
            jal comparar
            addi a1, a1, 1
            j lazo
final:      la t0, minimo
            sb a0, 0(t0)
            mv a1, a0
            li a0,1
            ecall

            li a0,10
            ecall
```

Responda las siguientes preguntas:

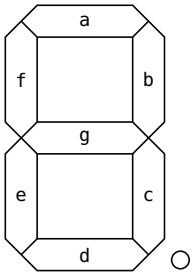
a) ¿Para qué sirve la pseudoinstrucción **`la`**? ¿En qué instrucciones se convierte?
b) ¿Por qué se asigna a1 y no t5 o s0 a la variable utilizada como índice?
c) ¿Por qué el **`addi`** de a1 es de 1 y no de 2 o 4?
d) ¿Para qué sirve la instrucción **`sltiu`**? ¿Qué diferencia tiene con **`bltu`**?
e) ¿Para qué sirven las instrucciones **`jal`** y **`jr`**? ¿Qué diferencias tienen con la instrucción **`j`**? ¿Son instrucciones o pseudo-instrucciones?

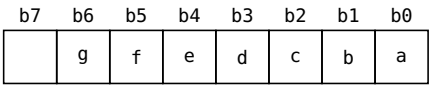
3) Escriba un programa en lenguaje ensamblador utilizando el ISA RV32IM que reciba la dirección de un vector de números enteros en el registro **`a0`**, lea todos los elementos del vector hasta encontrar el valor cero, y en ese momento finalice mostrando por consola la suma y el promedio (sólo parte entera) de los números leídos.

4) Escriba una subrutina que reciba dos elementos A y B almacenados en memoria, los compare y los intercambie para retornar siempre con A menor que B. La subrutina recibe las direcciones de los elementos en los registros **`a0`** y **`a1`** respectivamente. Por simplicidad, asuma que los elementos tienen un tamaño de un byte. Escriba además un programa principal de prueba que defina dos valores y muestre por pantalla cuál es el menor.

5) Usando la subrutina del ejercicio anterior escriba una subrutina que ordene de menor a mayor los elementos de un vector de números de 8 bits sin signo usando el método de ordenamiento que usted considere conveniente. El puntero al primer elemento se recibe en el registro **`a0`** y la cantidad de elementos en **`a1`**. Para las pruebas utilice un vector que tenga como mínimo 10 elementos.

6) En el laboratorio 1 de Sistemas con Microprocesadores y Microcontroladores, como parte del desarrollo del reloj, usted escribió una subrutina para encender los segmentos correspondientes a un dígito BCD. Escriba la misma subrutina con el set de instrucciones RV32IM de los procesadores RISC-V. El valor a mostrar, que deberá estar entre 0 y 9, se encuentra almacenado en el registro **`a0`**. Para la solución deberá utilizar una tabla de conversión de BCD a 7 segmentos la cual deberá estar almacenada en memoria a partir de la etiqueta **`TABLA`**. La asignación de los bits a los correspondientes segmentos del dígito se muestra en la figura que acompaña al enunciado.





7) En aplicaciones como procesamiento gráfico, cálculos científicos o inteligencia artificial es muy común tener que realizar multiplicaciones de matrices, del tipo $C = A * B$.
Es por ello que un algoritmo eficiente para multiplicar matrices es de gran utilidad, y es una de las más importantes funciones que implementan las librerías de este tipo. Es conocida como función GEMM *(General Matrix Multiplication)*, y en realidad hace $C = C + A * B$ para ganar generalidad.
En el código C que se muestra a continuación, se muestra una posible implementación de esta función, en el cual las matrices A, B y C son matrices cuadradas de 32x32 elementos, y cada elemento es de tipo entero de 32 bits.

```
void gemm(int c[][], int a[][], int b[][]) {
    size_t i, j, k;

    for(i = 0; i < 32; i++)
        for(j = 0; j < 32; j++)
            for(k = 0; k < 32; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Se pide que escriba una subrutina en lenguaje ensamblador RV32IM que implemente el código anterior.
Como las matrices son pasadas como parámetros, puede asumir que las direcciones base de cada una están en los registros **a0**, **a1** y **a2** (para C, A y B respectivamente).
Al igual que en muchos otros lenguajes de programación, en C las matrices en memoria se almacenan *ordenadas por filas*. Esto quiere decir que primero se guarda en memoria la primera fila completa, luego la segunda fila completa, y así sucesivamente con el resto de las filas.
Las variables temporales **i**, **j** y **k** se pueden mapear a los registros **t0**, **t1** y **t2**, respectivamente.
Para poder simular el código, le recomendamos que pruebe con matrices más pequeñas, por ejemplo de 4x4 elementos. Y también le recomendamos inicializar la matriz C en cero, para que sea más sencillo verificar los resultados.

Soluciones propuestas

1) La opción correcta es la C: **`sub x11, x10, x9`**

2) ...

3) ...

4) La subrutina puede hacerse con 6 instrucciones: dos de carga, dos de almacenamiento, un salto condicional y el retorno.

5) Como esta subrutina llamará a la anterior, es necesario almacenar al comienzo la dirección de retorno en la pila, y restaurarla al final antes de retornar. Luego, puede utilizarse un método d eordenación simple, como por ejemplo el de la burbuja. El mismo maneja dos punteros a elementos del vector, y tiene dos lazos anidados. En total la subrutina tendrá aproximadamente 13 líneas.

6) La subrutina tiene aproximadamente 5 líneas.

7) ...