Laboratorio de Estructura de Computadores
Curso 2020-2021

Práctica 3:
Programación en Ensamblador

Ejercicio 1: Desensamblar

Se proporciona un programa "Desensamblar.txt" compatible con la arquitectura MIPS descrita en las clases de teoría. En este ejercicio se debe desensamblar este programa, es decir, convertir las instrucciones en notación hexadecimal a notación ensamblador, para poder comprender la funcionalidad del programa.

El programa suministrado contiene dos segmentos, código y datos. Algunas instrucciones hacen referencia a posiciones del segmento de datos.

Objetivo

Desensamblar el programa propuesto y comprender su funcionamiento, para ello, se sugiere que con cada instrucción ensamblador, se incluya un comentario que explique la funcionalidad que genera.

Comprobar con el simulador Mars de MIPS, si el programa desensamblado coincide con el programa en código máquina. Con tal fin, se debe configurar la memoria como **<<Compacta con dirección de inicio del segmento text en x0000>>**. Para ello vaya a "**Settings** -> **Memory Configuration**…" y seleccione "**Compact, Text at Address 0**".

En todo este curso, debe verificar esta configuración siempre que use el simulador MARS, en concreto para todos los ejercicios y exámenes de esta práctica.

Ejercicio 2. Llamada a función y paso de parámetros

En este ejercicio se debe escribir en ensamblador un programa que tenga funcionalidad equivalente al programa en C propuesto y verificar su correcto funcionamiento. Para ello, debe entender cómo codificar una llamada a función en ensamblador y su paso de parámetros, a través de los registros indicados para ello, $a0, $a1 y el retorno del resultado a través de $v0.

Tenga en cuenta que la instrucción principal del programa C incluye múltiples operaciones:

- Lectura de la variable X de memoria
- Lectura de la variable Y de memoria
- Carga de los parámetros X e Y en los registros indicados para el paso de parámetros a funciones***, $a0*** y ***$a1***, respectivamente
- Llamada a función, mediante el uso de la instrucción ***jal <Etiqueta>***. Donde <etiqueta> identifica a la primera línea del código de la función en su programa en ensamblador. Se recomienda utiliza el nombre de la función ***calculaSumaMult*** como etiqueta.
- Recuperación del retorno de la función del registro ***$v0***
- Escritura del retorno en la variable R de memoria

A su vez, la instrucción principal de la función ***calculaSumaMult*** incluye las siguientes operaciones:

- Suma de los parámetros
- Multiplicación por 2 de la suma anterior
- Retorno del resultado a *main*, a través del registro indicado para ello, ***$v0.*** Para retornar a main, se debe hacer uso de la instrucción ***jr***, y como argumento el registro donde se guarda la dirección de retorno, es decir, ***$ra.***

Se adjunta un código equivalente en C del ejercicio:

```
int X = 10;
int Y = 4;
int R;

int calculaSumaMult (int a, int b)
 {
    return (a+b)*2;
 }

 int main()
 {
        R = calculaSumaMult(X,Y);

        while(1);
 }
```

Objetivo

Aprender a realizar llamadas a funciones y gestionar el paso de parámetros y retornos a través de los registros específicos.

Para la realización de este ejercicio se debe **configurar la memoria** como **<<Compacta con dirección de inicio del segmento text en x0>>.** Para ello vaya a **"Settings -> Memory Configuration…" y seleccione "Compact, Text at Address 0"**.

En todo este curso, debe verificar esta configuración siempre que use el simulador MARS, en concreto para todos los ejercicios y exámenes de esta práctica.

Ejercicio 3. Compilación de código C

En este ejercicio se debe escribir en ensamblador un programa que tenga funcionalidad equivalente al programa en C propuesto y verificar su correcto funcionamiento.

Para ello, debe entender cómo codificar y acceder a las distintas posiciones de un vector en ensamblador.

Utilice las etiquetas A, B, C y N para estos cuatro elementos en memoria de datos. Para gestionar el offset en las direcciones de acceso a memoria, debe utilizarse un registro, que habrá que incrementar en función del tamaño del dato.

Para guardar el espacio en memoria necesario para la variable C podemos usar la directiva **.space**

```
int N=2;
int A[N]={2,2};
int B[N]={-1,-5};
int C[N];

int main()
{
     C[0]=A[0]+B[0]*4;

     C[1]=A[1]+B[1]*4;

     while(1);
}
```

Objetivo

Comprender el problema propuesto, plantear e implementar una solución para dicho problema y verificar su correcto funcionamiento.

Para la realización de este ejercicio se debe **configurar la memoria** como **<<Compacta con dirección de inicio del segmento text en x0>>.** Para ello vaya a "**Settings -> Memory Configuration…" y seleccione "Compact, Text at Address 0".**

Otros ejercicios

- Puede seguir el tutorial de MARS disponible en Moodle donde va a trabajar con distintos tipos de salto y bifurcaciones.

- Repetir el ejercicio 3, considerando que ahora dispone de vectores de 10 posiciones y utilizando un ***bucle for*** para programar la funcionalidad descrita. Nota: Como ayuda podéis acudir a la pag. 91 de las transparencias de la U3

```
int N=10;
int A[N]={2,2,4,6,5,6,7,8,9,10};
int B[N]={-1,-5,4,10,1,-2,5,10,-10,0};
int C[N];

int main()
{
     int i;
     for (i=0; i<N; i++)
             C[i]=A[i]+B[i]*4;
     while(1);
}
```