Organización del Procesador

año 2019

Guía de ejercicios prácticos 2

Assembler

Requerimientos: Esta práctica está basada en la utilización de lenguaje Assembly, en particular NASM. Para obtenerlo puede descargarlo de su página oficial para diferentes arquitecturas: *http://www.nasm.us*. También utilizaremos un entorno de desarrollo SASM, el mismo pude obtenerse de su página *https://dman95.github.io/SASM/english.html*. Además, para simplificar la generación de archivos ejecutables y la Entrada/Salida, deberá descargar los archivos *driver.c, asm_io.asm y asm_io.inc* de la página de su autor *http://pacman128.github.io/pcasm/* o del aula virtual.

1. Escribir las instrucciones que representa cada inciso en el lenguaje NASM e indicar el modo de direccionamiento de cada una de ellas:

   a Incrementar el valor del registro EAX.

   b Cargar el registro EBX con el decimal 18 utilizando una etiqueta *label* binaria.

   c Sumar al registro EAX el valor 200 (expresado en decimal).

   d Cargar el registro AX (parte baja del registro EAX) con el contenido de la celda de memoria cuya dirección está almacenada en el registro EBX.

   e Multiplicar el valor 52 almacenado en una etiqueta en formato hexadecimal, con el valor cuya dirección de memoria está almacenada en el registro EBX.

   f Sumar al registro EAX el contenido de la celda de memoria cuya dirección es calculada incrementando el registro EBX en 4 unidades.

2. Indicar con qué valores queda representado el registro extendido EAX de la arquitectura 80386, según las operaciones aplicadas:

(a)

```
1 L1 db 0
2 mov eax, L1
```

(b)

```
1 L6 db 18h
2 mov eax, [L6]
3 add eax, [L6]
```

(c)

```
1 L1 dd 1Ah
2 mov AL, [L1]
```

(d)

```
1 mov ax, 5h
2 shl ax, 1
3 shr ax, 1
```

(e)

```
1 mov ax, 0110b
2 rol ax, 1
3 ror ax, 2
```

3. ★ Determine cómo simularía los esquemas de las estructuras de control (comparaciones, saltos condicionales y ciclos) en assembler.

4. ¿Qué diferencia existe entre el segmento de programa *.data* y el *.bss*? Dé un ejemplo que muestre la utilidad de cada uno.

5. Escriba un programa en assembler que utilice tres arreglos definidos en el segmento de datos, pero cada uno con un tamaño diferente de elementos (byte, word, dword). Determine cuántos bytes se necesitan para obtener cada elemento de estos arreglos. Realice un programa en assembler que muestre los elementos de los tres arreglos.

6. Defina cómo se puede determinar si un número es par utilizando operadores a nivel de bits. Luego, construya un programa assembler que dado un arreglo de 10 enteros de 16 bits (words), definido en el segmento *.data*, sume los números pares y muestre el resultado por pantalla. **Nota:** revise el capítulo 5 del libro de Paul A. Carter para conocer en detalle el tratamiento de arreglos en NASM.

7. Construya un programa assembler que calcule el factorial de un número dado. (resuelva de manera iterativa).

8. Construya un programa assembler que dado un número entero positivo N, calcule la serie de fibonacci hasta N y la muestre por pantalla (resuelva de manera iterativa). Ejemplo: La serie de fibonacci para $n = 6$ es 11235.

9. Retome el ejercicio Nro 6. Utilizando la solución propuesta, construya una librería (subrutina global) en assembler para calcular a nivel de bits si un número es par. Luego utilice esta librería desde un programa C para determinar si un número es par.

10. Revise el archivo `asm_io.asm`:

    - Comprenda el código de las subrutinas `print_int` y `read_int` que esta librería ofrece.
    - Agregue una subrutina `print_hex` para imprimir en formato hexadecimal siguiendo el estilo de la librería.
    - No olvide de declararla `global` para poder utilizarla desde otro código.
    - Recompile la librería `asm_io.asm`.
    - Retome algún ejercicio anterior o cree uno simple para utilizar la subrutina creada y poder imprimir en formato hexadecimal.

11. Realice un subrutina en assembler que calcule el mayor de dos números enteros. Utilice la misma para calcular el mayor valor de un arreglo de 10 elementos declarado en el segmento `.data`.

12. Construya una subrutina en assembler que dado un número (representado en un byte) determine si su representación binaria es palíndromo o no. Utilice la subrutina para determinar la cantidad de palíndromos (representación) de un arreglo de números.

13. Codifique en assembler (NASM) el siguiente programa C: (Aclaración, recuerde que las variables locales deben alojarse en memoria)

```
#include <stdio.h>
void main(){
        int n;
        printf("Sumar enteros hasta:");
        scanf("%d",&n);
        int sum = calc_sum(n);
        printf("Sum is %d\n",sum);
}

int calc_sum(int n){
        int acum =0;
        int i;
        for(i=1;i<=n;i++){
                acum = acum + elevar_cubo(i)
        }
        return acum;
}
```

```
int elevar_cubo(int x){
        return x*x*x;
}
```

14. ★ Implemente en assembler mediante una subrutina el algoritmo de Euclides (Máximo Común Divisor). Utilice la misma para calcular el MCD entre dos número enteros.