|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **UNIVERSIDAD AUTÓNOMA “TOMAS FRÍAS” CARRERA DE INGENIERÍA DE SISTEMAS** | | | |  |
| **Materia:** | Arquitectura de computadoras (SIS-522) | | |  |
| **Docente:**  **Auxiliar:** | Ing. Gustavo A. Puita Choque  Univ. Aldrin Roger Perez Miranda  Edwin Cruz | | | N° Práctica |
| 9 |
| **16/06/2024** | **Fecha publicación** | | |
| **01/07/2024** | **Fecha de entrega** | | |
| **Grupo:** | **1** | **Sede** | **Potosí** | |

**Responda las siguientes preguntas de MANERA CONCISA**

**LAS RESPUESTAS DE MANERA DIGITAL en formato .pdf**

1. ¿Qué es el 'stack' en el contexto del lenguaje ensamblador y cómo se utiliza?

El 'stack' (pila) en el contexto del lenguaje ensamblador es una estructura de datos de tipo LIFO (Last In, First Out), es decir, el último elemento en entrar es el primero en salir. Se utiliza para almacenar datos temporales, como parámetros de funciones, direcciones de retorno y registros de propósito general.

La pila se maneja principalmente mediante dos registros: el puntero de pila (SP) que apunta a la cima de la pila, y el puntero de base (BP) que se utiliza generalmente para acceder a los parámetros y variables locales de una función.

Las operaciones básicas con la pila incluyen:

* **PUSH**: Empuja un valor al tope de la pila, decrementando el SP.
* **POP**: Extrae un valor del tope de la pila, incrementando el SP.
* **CALL**: Guarda la dirección de retorno en la pila y salta a una subrutina.
* **RET**: Recupera la dirección de retorno de la pila y vuelve a la llamada anterior.

Ejemplo de uso en ensamblador:

assembly

Copiar código

PUSH AX ; Guarda el contenido del registro AX en la pila

MOV AX, 5 ; Modifica AX

POP AX ; Recupera el valor original de AX desde la pila

1. Describe un escenario práctico donde el uso de ensamblador sería más ventajoso que el uso de un lenguaje de alto nivel.

Un escenario práctico donde el uso de ensamblador es ventajoso es en el desarrollo de sistemas embebidos con recursos limitados (memoria, velocidad de CPU). En estos sistemas, el control preciso del hardware y la optimización del uso de recursos son cruciales.

Por ejemplo, en el desarrollo de controladores de dispositivos (drivers) o firmware para microcontroladores en aplicaciones críticas como sistemas de navegación de aeronaves, sistemas de control industrial o dispositivos médicos, donde cada ciclo de reloj y byte de memoria cuenta. En estos casos, el ensamblador permite:

* Control fino sobre el hardware.
* Optimización extrema del código para maximizar el rendimiento y minimizar el uso de memoria.
* Respuesta en tiempo real.

3) Explique cada línea del siguiente código del lenguaje ensamblador y diga que es lo que se está haciendo

MOV AX, 5 ; Línea 1

MOV BX, 10 ; Línea 2

ADD AX, BX ; Línea 3

MOV CX, AX ; Línea 4

**Línea 1: MOV AX, 5**

* **Operación**: Mueve el valor inmediato 5 al registro AX.
* **Descripción**: AX ahora contiene el valor 5.

**Línea 2: MOV BX, 10**

* **Operación**: Mueve el valor inmediato 10 al registro BX.
* **Descripción**: BX ahora contiene el valor 10.

**Línea 3: ADD AX, BX**

* **Operación**: Suma el valor del registro BX al registro AX.
* **Descripción**: AX = AX + BX. Después de esta instrucción, AX contiene 5 + 10 = 15.

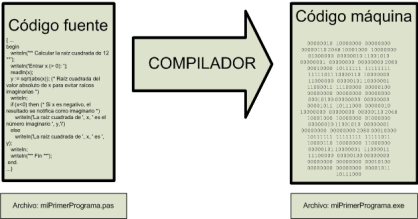
**Línea 4: MOV CX, AX**

* **Operación**: Mueve el valor del registro AX al registro CX.
* **Descripción**: CX ahora contiene el valor 15, que es el resultado de la suma anterior.

**Resumen del código:**

El código en ensamblador realiza una serie de operaciones aritméticas simples. Primero, carga los valores 5 y 10 en los registros AX y BX, respectivamente. Luego, suma estos valores y almacena el resultado en AX. Finalmente, mueve el resultado al registro CX.

4) Explique detalladamente cómo funcionan los compiladores



**Funcionamiento de los Compiladores**

**1. Código Fuente**

El código fuente es el conjunto de instrucciones escritas por el programador en un lenguaje de programación de alto nivel (como C, C++, Java, Python, etc.). Estos lenguajes son más fáciles de entender y escribir para los humanos, ya que utilizan una sintaxis similar al lenguaje natural y abstraen los detalles del hardware.

Ejemplo de código fuente en C:

#include <stdio.h>

int main() {

printf("Hola, mundo!\n");

return 0;

}

**2. Compilador**

El compilador es un programa que traduce el código fuente escrito en un lenguaje de alto nivel al código máquina, que es el conjunto de instrucciones que una computadora puede ejecutar directamente. Este proceso se realiza en varias etapas:

* **Análisis Léxico**: El compilador divide el código fuente en tokens, que son las unidades básicas de significado (palabras clave, identificadores, operadores, etc.).
* **Análisis Sintáctico**: El compilador verifica la estructura gramatical del código, asegurándose de que esté escrito de acuerdo con las reglas del lenguaje de programación.
* **Análisis Semántico**: El compilador revisa el código para asegurar que las operaciones tengan sentido (por ejemplo, que no se intente sumar una cadena de texto a un número).
* **Optimización**: El compilador puede realizar varias transformaciones para mejorar la eficiencia del código sin cambiar su comportamiento.
* **Generación de Código Intermedio**: El código fuente se traduce a un formato intermedio que es más fácil de optimizar.
* **Generación de Código Máquina**: El formato intermedio se traduce a instrucciones específicas de la arquitectura del hardware.
* **Enlazado**: Las diferentes partes del programa y las bibliotecas externas se combinan para producir un ejecutable completo.

**3. Código Máquina**

El código máquina es el conjunto de instrucciones binarias que el procesador de la computadora puede ejecutar directamente. Cada instrucción en el código máquina realiza una operación específica en la CPU, como mover datos entre registros, realizar cálculos aritméticos o controlar el flujo del programa.

Ejemplo simplificado de código máquina:

Copiar código

10111000 00000101 ; MOV AX, 5

10111011 00001010 ; MOV BX, 10

00000011 ; ADD AX, BX

10111101 00000000 ; MOV CX, AX

**Ejemplo del Proceso de Compilación**

Consideremos el siguiente código fuente en C:

#include <stdio.h>

int main() {

int a = 5;

int b = 10;

int c = a + b;

printf("La suma es %d\n", c);

return 0;

}

**Etapas del Compilador:**

1. **Análisis Léxico**: Se generan tokens como #include, <stdio.h>, int, main, a, =, 5, ;, etc.
2. **Análisis Sintáctico**: El compilador verifica que el código tiene una estructura válida, como una definición correcta de la función main y la declaración y asignación de variables.
3. **Análisis Semántico**: Se comprueba que las operaciones tienen sentido, como que a y b son enteros y pueden ser sumados.
4. **Optimización**: El compilador puede optimizar el código para mejorar la eficiencia. Por ejemplo, podría simplificar las expresiones o eliminar el código innecesario.
5. **Generación de Código Intermedio**: El código fuente se traduce a un formato intermedio, más abstracto que el código máquina pero más cercano al hardware que el código fuente original.
6. **Generación de Código Máquina**: El código intermedio se traduce a instrucciones específicas para la CPU, como mover valores a registros y realizar la suma.
7. **Enlazado**: El código generado se combina con las bibliotecas necesarias (como la implementación de printf) para producir un ejecutable.

**Código Máquina (Simplificado)**

El código máquina resultante podría incluir instrucciones como mover valores a registros, realizar sumas y llamadas a funciones de la biblioteca estándar.

5) Realizar capturas de pantalla del siguiente procedimiento: