	UNIVERSIDAD AUTÓNOMA TOMAS FRÍAS **CARRERA DE INGENIERÍA DE SISTEMAS**
	ESTUDIANTE: Univ. Beimar Hernán Escudero Apaza

MATERIA: Arquitectura de Computadoras	**SIGLA:** SIS-522
DOCENTE: Ing. Gustavo Puita	**PRÁCTICA:** 9
AUXILIAR: Univ. Aldrin Roger Pérez Miranda	**GRUPO:** 1

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

Ees un área de memoria utilizada para el almacenamiento temporal de datos. Se utiliza para manejar llamadas a funciones, almacenar variables locales y administrar el paso de parámetros entre funciones.

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

La programación ensambladora tiene ventajas sobre el uso de lenguajes de alto nivel es en el desarrollo de controladores de dispositivos o sistemas integrados que requieren un control preciso del hardware y del rendimiento. Además, al ejecutar tareas críticas en tiempo real donde cada ciclo de CPU cuenta, el ensamblador puede optimizar el código para reducir la latencia y aumentar la eficiencia.

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

Explicación del código:

- `MOV AX, 5`: Esta línea mueve el valor inmediato 5 al borrador de AX. - `MOV BX, 10`: Aquí el valor inmediato 10 se mueve al registro BX. - 'ADD AX, BX': Esta instrucción suma el contenido de los registros AX y BX y almacena el resultado en AX. - `MOV CX, AX`: Finalmente el valor obtenido en AX se mueve al scratchpad de CX. En resumen, este código ensamblador realiza una simple suma de los números 5 y 10 y almacena el resultado en el registro CX.

4) Explique detalladamente cómo funcionan los compiladores

muestra el proceso de compilación en el que un archivo de código fuente (`miPrimerPrograma.pas`) se convierte en un archivo ejecutable de código máquina (`miPrimerPrograma.exe`). A continuación, se explica detalladamente cómo funcionan los compiladores, siguiendo las fases del proceso de compilación:

Funcionamiento de los Compiladores

Fases del Proceso de Compilación

1. **Análisis Léxico (Lexical Analysis)**

    - **Función**: Convertir el código fuente en una secuencia de tokens. Un token es una secuencia de caracteres con significado conjunto, como palabras clave, identificadores, operadores y delimitadores.
    - **Herramienta**: Analizador léxico o lexer.
    - **Resultado**: Secuencia de tokens.

```
plaintext
Copiar código
begin, writeln, "Calcular la raíz cuadrada de 12", ;, readln, (,
x, ), ;, y, :=, sqrt, (, abs, (, x, ), ), ;, if, (, x, <, 0, ),
then, {, writeln, (, "La raíz cuadrada de ", x, " es el número
imaginario ", y, ), ;, }, else, {, writeln, (, "La raíz cuadrada
de ", x, " es ", y, ), ;, }, writeln, (, "Fin", ), ;, end, .
```

2. **Análisis Sintáctico (Syntactic Analysis)**

    - **Función**: Organizar los tokens en una estructura de árbol que refleja la estructura gramatical del programa. Este árbol se llama árbol de sintaxis.
    - **Herramienta**: Analizador sintáctico o parser.
    - **Resultado**: Árbol de sintaxis.

```
plaintext
Copiar código
Programa
├─ Instrucción: writeln("Calcular la raíz cuadrada de 12")
├─ Instrucción: writeln("Entrar x (> 0): ")
├─ Instrucción: readln(x)
├─ Instrucción: y := sqrt(abs(x))
├─ Condicional: if (x < 0) then
│   ├─ Instrucción: writeln("La raíz cuadrada de ", x, " es el
número imaginario ", y)
├─ Else
│   ├─ Instrucción: writeln("La raíz cuadrada de ", x, " es ", y)
├─ Instrucción: writeln("Fin")
```

Análisis Semántico (Semantic Analysis)

Función: Verificar que el árbol de sintaxis cumpla con las reglas semánticas del lenguaje, como la verificación de tipos y la correcta declaración y uso de variables.

Herramienta: Analizador semántico.

Resultado: Árbol de sintaxis anotado con información semántica.

Optimización del Código Intermedio

Función: Transformar el árbol de sintaxis en una representación intermedia del código y aplicar optimizaciones para mejorar la eficiencia del código.

Resultado: Código intermedio optimizado.

Generación de Código (Code Generation)

Función: Convertir el código intermedio optimizado en código máquina específico para la arquitectura de destino.

Resultado: Código máquina o código objeto.

Optimización de Código a Nivel de Máquina

Función: Aplicar optimizaciones específicas de la arquitectura para mejorar el rendimiento del código máquina.

Resultado: Código máquina optimizado.

Enlazado (Linking)

Función: Combinar varios módulos de código objeto y bibliotecas en un solo ejecutable, resolviendo referencias entre módulos y asignando direcciones a las variables y funciones.

Herramienta: Enlazador o linker.

Resultado: Ejecutable final (`miPrimerPrograma.exe`).

Diagrama del Proceso de Compilación

```
plaintext
Copiar código
Código Fuente -> [Análisis Léxico] -> Tokens -> [Análisis Sintáctico]
-> Árbol de Sintaxis -> [Análisis Semántico] -> Árbol de Sintaxis
Anotado -> [Optimización] -> Código Intermedio Optimizado ->
[Generación de Código] -> Código Máquina -> [Enlazado] -> Ejecutable
```

Ejemplo

Código Fuente (Pascal)

```
pascal
Copiar código
begin
    writeln('Calcular la raíz cuadrada de 12');
    writeln('Entrar x (> 0): ');
    readln(x);
    y := sqrt(abs(x));
    if (x < 0) then
        writeln('La raíz cuadrada de ', x, ' es el número imaginario
', y)
    else
        writeln('La raíz cuadrada de ', x, ' es ', y);
    writeln('Fin');
end.
```

Proceso de Compilación

1. **Análisis Léxico**:

    - Tokens:

```
begin, writeln, 'Calcular la raíz cuadrada de 12', ;, readln, (, x,
), ;, y, :=, sqrt, (, abs, (, x, ), ), ;, if, (, x, <, 0, ), then, writeln, (, 'La
raíz cuadrada de ', ,, x, ,, ' es el número imaginario ', ,, y, ), ;,
else, writeln, (, 'La raíz cuadrada de ', ,, x, ,, ' es ', ,, y, ), ;,
writeln, (, 'Fin', ), ;, end, .
```

2. **Análisis Sintáctico**:

    - Árbol de sintaxis:

```
plaintext
Copiar código
Programa
├─ Instrucción: writeln('Calcular la raíz cuadrada de 12')
├─ Instrucción: writeln('Entrar x (> 0): ')
├─ Instrucción: readln(x)
├─ Instrucción: y := sqrt(abs(x))
├─ Condicional: if (x < 0) then
│   ├─ Instrucción: writeln('La raíz cuadrada de ', x, ' es
el número imaginario ', y)
├─ Else
│   ├─ Instrucción: writeln('La raíz cuadrada de ', x, ' es
', y)
├─ Instrucción: writeln('Fin')
```

Análisis Semántico: Verificación de tipos y uso de variables.

Optimización: Eliminación de código innecesario y optimización de expresiones.

Generación de Código: Generación de código máquina específico.

Enlazado: Creación del ejecutable final.