Universidad Autónoma de Baja California
Facultad de Ingeniería Arquitectura y Diseño

Ingeniería en Software y Tecnologías Emergentes

Materia: Organización de Computadoras

Alumno: Jesus Eduardo Rodríguez Ramírez

Profesor: Jonatan Crespo Ragland

Grupo 932

Trabajo: Taller 6

Ensenada, B.C; a 7 de octubre del 2024

Características principales de una máquina multinivel.

Una máquina multinivel es una representación conceptual que organiza un sistema de cómputo en diferentes niveles de abstracción. Cada nivel tiene una función específica y se comunica con los otros niveles para que las instrucciones pasen de ser comprensibles para el ser humano a ser ejecutables por el hardware. Los niveles superiores son más abstractos y los inferiores están más cercanos al hardware.

Niveles de una máquina multinivel:

1. **Hardware:** Nivel físico de la máquina, incluye componentes como el procesador (CPU), la memoria, y dispositivos de entrada/salida.
2. **Microarquitectura:** Define cómo el procesador ejecuta las instrucciones y maneja los datos dentro del CPU.
3. **Sistema Operativo:** Administra los recursos del hardware y ofrece una plataforma sobre la cual los programas pueden correr.
4. **Lenguaje de Bajo Nivel (Ensablador):** Se compone de instrucciones que el procesador puede ejecutar directamente.
5. **Lenguaje de Alto Nivel (C, Python, etc.):** Lenguajes más abstractos que permiten escribir instrucciones sin preocuparse directamente por cómo interactuar con el hardware.
6. **Aplicaciones:** Programas que los usuarios utilizan para realizar tareas, como procesadores de texto o navegadores.

Comunicación entre los niveles

1. **Hardware - Microarquitectura:**
    - El hardware realiza operaciones a nivel de circuitos eléctricos, mientras que la microarquitectura organiza estas operaciones para que se ejecuten correctamente en el procesador.
    - Interfaz: Las señales eléctricas y el diseño lógico del procesador definen cómo las instrucciones se interpretan y ejecutan.
2. **Microarquitectura - Sistema Operativo:**
    - El sistema operativo usa las capacidades del procesador para gestionar tareas como la memoria y los dispositivos de entrada/salida.
    - Interfaz: El sistema operativo utiliza interrupciones y operaciones de entrada/salida para comunicarse con el hardware.
3. **Sistema Operativo - Lenguaje de Bajo Nivel:**
    - Los programas en ensamblador dependen del sistema operativo para tareas como la gestión de la memoria o el control de periféricos.
    - Interfaz: Los programas ensamblador hacen uso de llamadas al sistema (system calls) para interactuar con el sistema operativo.

4. **Lenguaje de Bajo Nivel - Lenguaje de Alto Nivel:**
    - Los programas escritos en lenguajes de alto nivel se convierten en instrucciones en ensamblador (o código de máquina) mediante un compilador.
    - Interfaz: El compilador convierte el código fuente en un código optimizado para el hardware.
5. **Lenguaje de Alto Nivel - Aplicaciones:**
    - Las aplicaciones están escritas en lenguajes de alto nivel y son ejecutadas en el sistema operativo una vez que han sido compiladas o interpretadas.
    - Interfaz: El compilador o intérprete traduce el código de alto nivel en instrucciones comprensibles para la máquina.

Características de los lenguajes de bajo nivel

- Son lenguajes muy cercanos al código máquina, es decir, se relacionan directamente con el hardware.
- Ofrecen un control detallado sobre los recursos del sistema, como los registros y la memoria.
- Son rápidos y eficientes, pero difíciles de escribir y entender.
- Se utilizan para optimizar el rendimiento y obtener control preciso sobre el hardware.

Características de los lenguajes de alto nivel

- Abstraen los detalles del hardware, facilitando su uso y comprensión.
- Son más fáciles de aprender y usar, ya que están diseñados para que los humanos los entiendan mejor.
- No dependen de una arquitectura de hardware específica, lo que los hace portables.
- Aunque ofrecen menos control directo sobre el hardware, permiten desarrollar software más rápidamente.

Comparación entre lenguajes de bajo nivel y lenguajes de alto nivel

- **Nivel de abstracción:**

Los lenguajes de bajo nivel están muy cerca del hardware, lo que significa que el programador debe manejar detalles como registros, memoria y operaciones del procesador de manera directa.

Los lenguajes de alto nivel, en cambio, abstraen esos detalles, lo que permite al programador concentrarse en la lógica y el propósito del programa sin preocuparse por cómo funciona internamente el hardware.

- **Velocidad de ejecución:**

Los lenguajes de bajo nivel suelen ser más rápidos, ya que sus instrucciones están muy cerca del código de máquina que ejecuta el procesador. Esto significa que no hay mucha traducción o procesamiento adicional.

Los lenguajes de alto nivel, debido a su abstracción, pueden ser más lentos en comparación, ya que requieren más pasos para que sus instrucciones se traduzcan y ejecuten en el hardware.

- **Control sobre el hardware:**

En los lenguajes de bajo nivel, el programador tiene un control completo y preciso sobre el hardware, pudiendo gestionar directamente recursos como la memoria y los dispositivos.

En los lenguajes de alto nivel, el control sobre el hardware es más limitado, ya que el sistema operativo y otros componentes manejan esos detalles automáticamente.

- **Facilidad de uso:**

Los lenguajes de bajo nivel son más difíciles de aprender y utilizar porque requieren que el programador entienda el funcionamiento interno de la máquina.

Los lenguajes de alto nivel son mucho más fáciles de aprender y usar, ya que están diseñados para ser más intuitivos y legibles para los humanos.

- **Portabilidad:**

Los programas escritos en lenguajes de bajo nivel suelen ser específicos de la arquitectura de hardware para la que fueron creados, lo que significa que no se pueden ejecutar fácilmente en otro tipo de máquina sin cambios importantes.

Los lenguajes de alto nivel son portables, lo que permite que el mismo programa se ejecute en diferentes plataformas sin modificaciones significativas, gracias a su abstracción del hardware.

Ejemplos de lenguajes de bajo nivel en la industria actual

1. **Sistemas embebidos (C o ensamblador):** Empresas como **Texas Instruments** utilizan lenguajes de bajo nivel para programar microcontroladores porque necesitan aprovechar al máximo los recursos limitados de estos dispositivos.
2. **Desarrollo de sistemas operativos:** Tanto **Microsoft** como **Linux** utilizan ensamblador para gestionar funciones críticas como interrupciones o control de hardware en áreas que necesitan un rendimiento máximo.
3. **Drivers de hardware (NVIDIA/AMD):** Las compañías de tarjetas gráficas como **NVIDIA** y **AMD** usan ensamblador y C para optimizar el rendimiento de sus controladores y aprovechar al máximo el hardware de sus productos.

Instrucciones (opcodes) en ensamblador x86

- **MOV:** Copia un valor de un registro a otro, o entre un registro y la memoria.
- **ADD:** Suma dos valores y almacena el resultado en el destino.
- **SUB:** Resta un valor de otro.
- **INC:** Incrementa en uno el valor de un registro o una variable.
- **DEC:** Decrementa en uno el valor de un registro o una variable.
- **CMP:** Compara dos valores y actualiza las banderas del procesador.
- **JMP:** Salta a una dirección específica en el código.
- **JE/JZ:** Salta si la comparación anterior fue igual a cero.
- **JNE/JNZ:** Salta si la comparación anterior no fue igual a cero.
- **PUSH/POP:** Coloca un valor en la pila o lo recupera de la pila.
- **CALL/RET:** Llama a una subrutina y regresa de ella.
- **AND/OR/XOR/NOT:** Realizan operaciones lógicas bit a bit.
- **SHR/SHL:** Desplaza los bits de un valor hacia la derecha o izquierda.

Proceso de enlazamiento en ensamblador

El **enlazamiento** es el proceso de combinar diferentes partes del código ensamblador y convertirlas en un programa ejecutable. Se lleva a cabo en tres etapas:

1. **Compilación:** El código fuente en ensamblador se traduce a código objeto, que es un conjunto de instrucciones en código de máquina.
2. **Enlazado:** El enlazador toma el código objeto y lo combina con otros módulos y bibliotecas, resolviendo referencias a funciones o variables externas. Al final, genera un archivo ejecutable.
3. **Cargador:** El archivo ejecutable se carga en memoria y se asignan direcciones finales a las instrucciones antes de que el programa comience a ejecutarse.

Este proceso asegura que todas las partes del programa estén listas para ejecutarse en el hardware correcto.