UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA ARQUITECTURA Y DISEÑO

Universidad Autónoma de Baja California
Facultad de ingeniería, Arquitectura y Diseño
Ingeniero en Software y Tecnologías Emergentes

MATERIA: Organización de computadoras

Taller 6

373488

GRUPO: 932

Balderas Rosas Adrian

Jonatan Crespo Ragland

07/10/2024

Características principales de una máquina multinivel

Una **máquina multinivel** es una abstracción computacional que divide las operaciones del sistema en diferentes niveles de funcionalidad, desde el hardware físico hasta las aplicaciones de software. Cada nivel se comunica con otros a través de interfaces y capas que permiten transformar las instrucciones y datos de un nivel al siguiente. Las características principales de este tipo de máquina incluyen:

1. **Jerarquía de niveles**: Se organizan en capas desde el hardware hasta las aplicaciones. Cada capa tiene funciones especializadas que dependen de la capa inferior.
2. **Abstracción**: Las capas superiores están más abstraídas del hardware físico, lo que permite a los programadores escribir código sin preocuparse por los detalles de bajo nivel.
3. **Interfaces definidas**: Cada nivel se comunica con otros a través de interfaces claras y bien definidas, como compiladores, sistemas operativos y protocolos.
4. **Modularidad**: Al dividirse en niveles, los sistemas pueden desarrollarse y mantenerse en partes independientes.

Comunicación entre niveles de una máquina multinivel

1. Hardware (Nivel físico)

- **Función**: Es el nivel más bajo, compuesto por los componentes físicos como la CPU, memoria, buses y dispositivos de E/S.
- **Interfaz**: El hardware se controla mediante señales eléctricas e instrucciones de lenguaje de máquina, lo que incluye interacciones directas con registros y memoria.
- **Comunicación**: La microarquitectura se comunica con el hardware a través de microinstrucciones que controlan directamente el comportamiento de la CPU y otros componentes.

2. Microarquitectura

- **Función**: Define cómo se ejecutan las instrucciones a nivel de hardware. Se encarga de traducir instrucciones del conjunto de instrucciones de la máquina en señales de control para el hardware.
- **Interfaz**: La microarquitectura se encarga de interpretar las instrucciones de bajo nivel (lenguaje ensamblador o código máquina) y traducirlas en operaciones de hardware.
- **Comunicación**: Las instrucciones de lenguaje de máquina pasan a través de la microarquitectura, que las traduce en señales eléctricas para el hardware.

3. Sistema operativo

- **Función**: Proporciona una capa de abstracción sobre el hardware, manejando la administración de recursos (memoria, procesos, dispositivos) y permitiendo la ejecución de programas de usuario.
- **Interfaz**: El sistema operativo usa controladores y llamadas al sistema para interactuar con el hardware y la microarquitectura.
- **Comunicación**: Los programas en lenguajes de alto nivel realizan llamadas al sistema que el sistema operativo traduce en operaciones de bajo nivel.

4. Lenguaje de alto nivel

- **Función**: Ofrece a los programadores un conjunto de abstracciones (variables, funciones, estructuras de control) que se compilan o interpretan en instrucciones de bajo nivel (ensamblador o código máquina).
- **Interfaz**: Se utilizan compiladores o intérpretes para traducir el código de alto nivel en código ensamblador o directamente en lenguaje máquina.
- **Comunicación**: El compilador convierte el código fuente en código objeto, que el enlazador y el sistema operativo procesan para su ejecución.

5. Aplicación

- **Función**: Son los programas que interactúan directamente con el usuario, como navegadores web o editores de texto.
- **Interfaz**: Las aplicaciones realizan llamadas al sistema y usan APIs para comunicarse con el sistema operativo, que a su vez interactúa con el hardware.
- **Comunicación**: A través de llamadas a bibliotecas y APIs, las aplicaciones interactúan con el sistema operativo y, por extensión, con el hardware subyacente.

Características del lenguaje de bajo nivel

1. **Cercanía al hardware**: Ofrecen control directo sobre los registros y la memoria.
2. **Velocidad y eficiencia**: Son más rápidos debido a la mínima abstracción y al control granular sobre los recursos del sistema.
3. **Difícil portabilidad**: El código de bajo nivel está estrechamente ligado a la arquitectura del hardware, lo que dificulta su portabilidad.
4. **Complicado de programar**: Requieren un conocimiento profundo de la arquitectura de la máquina.

Características del lenguaje de alto nivel

1. **Abstracción**: Proporcionan una mayor abstracción sobre el hardware, utilizando conceptos más cercanos al lenguaje humano (variables, funciones, objetos).

2. **Portabilidad**: El código escrito en lenguajes de alto nivel puede ser compilado o interpretado en diversas plataformas.
3. **Facilidad de uso**: Son más fáciles de leer, escribir y mantener.
4. **Menor control sobre el hardware**: No permiten un control detallado sobre los recursos del sistema.

Comparación entre lenguajes de bajo y alto nivel

Características	Lenguajes de Bajo Nivel	Lenguajes de Alto Nivel
Abstracción	Muy baja (cercanía al hardware)	Alta (más cerca del lenguaje humano)
Velocidad y Eficiencia	Muy alta	Menor en comparación
Facilidad de programación	Difícil de programar y depurar	Fácil de aprender y usar
Portabilidad	Baja	Alta (multi-plataforma)
Control sobre el hardware	Total control	Control limitado

Ejemplos de uso de lenguajes de bajo nivel en la industria

1. **Desarrollo de sistemas operativos (C y ensamblador)**:
    - Los sistemas operativos como **Linux** o **Windows** usan lenguajes de bajo nivel para un control eficiente del hardware.
    - **Por qué lo utilizan**: Se requiere acceso directo a la memoria y a los registros para gestionar procesos, interrupciones y el uso eficiente de los recursos.
2. **Firmware en dispositivos embebidos (ensamblador, C)**:
    - El firmware de **microcontroladores** y otros sistemas embebidos usa lenguajes de bajo nivel.
    - **Por qué lo utilizan**: Se necesita un código altamente optimizado que funcione con recursos limitados (memoria y CPU).
3. **Desarrollo de videojuegos y motores gráficos (C++, ensamblador)**:
    - Motores como **Unreal Engine** o **Unity** implementan partes críticas de su código en ensamblador y C++ para garantizar el mejor rendimiento gráfico.
    - **Por qué lo utilizan**: La necesidad de ejecutar cálculos gráficos de manera extremadamente eficiente para obtener altos FPS y un rendimiento estable.

Opcodes en ensamblador x86

1. **MOV**: Copia datos de una fuente a un destino.
    - Ejemplo: `MOV AX, 5` (Copia el valor 5 al registro AX).
2. **ADD**: Suma dos operandos.
    - Ejemplo: `ADD AX, BX` (Suma el valor de BX a AX).
3. **SUB**: Resta dos operandos.
    - Ejemplo: `SUB AX, BX` (Resta el valor de BX de AX).
4. **INC**: Incrementa el valor de un operando en 1.
    - Ejemplo: `INC AX` (Incrementa AX en 1).
5. **DEC**: Decrementa el valor de un operando en 1.
    - Ejemplo: `DEC AX` (Decrementa AX en 1).
6. **CMP**: Compara dos operandos.
    - Ejemplo: `CMP AX, BX` (Compara AX con BX).
7. **JMP**: Salta a una dirección de memoria.
    - Ejemplo: `JMP etiqueta` (Salta a la instrucción en la etiqueta).
8. **JE/JZ**: Salta si es igual o si el resultado es cero.
    - Ejemplo: `JE etiqueta` (Salta si la comparación anterior fue igual).
9. **JNE/JNZ**: Salta si no es igual o si el resultado no es cero.
    - Ejemplo: `JNE etiqueta` (Salta si no fueron iguales).
10. **PUSH/POP**: Almacena y recupera datos de la pila.

- Ejemplo: `PUSH AX` / `POP AX`.

11. **CALL/RET**: Llama y regresa de una subrutina.

- Ejemplo: `CALL subrutina` / `RET`.

12. **AND/OR/XOR/NOT**: Realizan operaciones lógicas.

- Ejemplo: `AND AX, BX`.

13. **SHR/SHL**: Desplazan bits hacia la derecha o izquierda.

- Ejemplo: `SHL AX, 1`.

Proceso de enlazamiento en ensamblador

El proceso de enlazamiento transforma el código ensamblador en un programa ejecutable. Los pasos son:

1. **Compilación**: El código fuente en ensamblador se convierte en código objeto.
2. **Enlazamiento**: Los archivos objeto se combinan con bibliotecas necesarias, creando un ejecutable.
3. **Cargador**: El ejecutable se carga en la memoria para ser ejecutado por la CPU.