***Curso de Assembly***

**O que é Assembly?**

Assembly é uma linguagem feita para conversar diretamente com o processador do computador. Ele basicamente entende sinais muito simples de se entender como por exemplo:

* "Coloque isso aqui."
* "Some com isso."
* "Guarde esse resultado ali."
* "Se for igual a tal coisa, pule para outro lugar."

O Assembly foi inventado porque no começo da computação os programas eram feitos todos em binários, utilizando 0 e 1. Isso era muito difícil e leva muito tempo para construir os programas. Assim o Assembly já deixava muita coisa pronta, por exemplo o uso da tabela ASCII ao invés de ter que digitar todos os zeros e uns para digitar uma letra ou símbolo, além de sequências para mover, somar e subtrair dados por exemplo.

Nos dias atuais ele é amplamente usado para criar Sistemas Operacionais, fazer programas onde cada segundo deste importa, controlar hardwares diretamente. Acaba sendo diferente de outras linguagens como C por exemplo que já vem com praticamente tudo pronto.

Temos então a divisão dos tipos de linguagens, sendo o sistema em **Lógica Digital (0 e 1)**, o **Assembly** utilizando instruções como MOV e ADD para mover, somar, e a tabela ASCII, e por último as linguagens de **Alto Nível** como C. Sendo assim a Linguagem C por exemplo é compilada para a Linguagem Assembly e posteriormente compilada para a Linguagem de Máquina.

**Arquitetura x86?**

A arquitetura x86 é um conjunto de instruções criado pela Intel que acabou se tornando padrão para processadores de computadores pessoais, tendo sido implementada no processador **Intel 8086** de 16 bits, evoluindo posteriormente para os processadores de 32 bits e 64 bits. Na Intel esse tipo de arquitura se chama **x86-64** e na AMD é chamada de **AMD64**.

**Linha do tempo**

Intel 8086 (1978) → Introduziu a arquitetura x86 com 16 bits.

**Intel 8086 (1978)**: Foi o primeiro processador a introduzir a arquitetura x86, com 16 bits. Seu sucessor, o **Intel 8088**, usava um barramento de dados de 8 bits, mas a lógica interna era baseada no 8086.

**Intel 80286 (1982)**: Introduziu a proteção de memória, o que permitiu a execução de múltiplos programas de forma mais segura.

**Intel 80386 (1985)**: Introduziu o modo de 32 bits e a virtualização.

**Intel Pentium (1993)**: Introduziu o processamento de 64 bits e as instruções de SIMD (Single Instruction, Multiple Data), que permitem realizar várias operações de dados em paralelo.

**AMD64 (2003)**: Uma extensão de 64 bits da arquitetura x86, introduzida pela AMD, permitindo endereçamento de memória superior a 4 GB e outras melhorias, que se tornou o padrão para os processadores modernos.

### ****3. Componentes Principais de um Processador x86****

Para entender a arquitetura, você precisa entender os principais **componentes** que o processador x86 possui:

**Registradores**: Pequenos espaços de memória dentro do processador que armazenam dados temporários.

**EAX, EBX, ECX, EDX**: Registradores de propósito geral, usados para operações aritméticas e manipulação de dados.

**ESP (Stack Pointer)**: Aponta para o topo da pilha de memória (importante para chamadas de função e controle de fluxo).

**EBP (Base Pointer)**: Usado para acessar variáveis locais em uma função.

**EIP (Instruction Pointer)**: Aponta para a próxima instrução a ser executada.

**Flags (EFLAGS)**: Registrador de status, usado para controlar operações como comparações e saltos condicionais.

**Barramentos de Dados e Endereços**:

O **barramento de dados** transporta dados entre os componentes do processador e a memória.

O **barramento de endereços** transporta endereços de memória para acessar dados ou instruções.

**Cache**: Memória de acesso ultrarrápido que armazena dados frequentemente acessados para melhorar o desempenho.

**Unidade Lógica e Aritmética (ALU)**: Responsável por executar operações matemáticas e lógicas (como soma, subtração, e comparações).

**Controlador de Interrupts**: Responsável por interromper o fluxo normal de execução e desviar a execução para rotinas de interrupção (como tratar uma entrada de teclado ou uma solicitação de rede).

### ****4. Modos de Operação****

A arquitetura x86 possui **vários modos de operação**, e entender como eles funcionam é fundamental:

**Modo Real (Real Mode)**:

Este é o modo mais simples, e o processador x86 funciona como se fosse um processador de 16 bits (como o 8086). Ele não oferece proteção de memória, e o sistema pode acessar diretamente a memória sem restrições.

O limite de endereçamento de memória é 1 MB.

**Modo Protegido (Protected Mode)**:

Introduzido no 80286 e aperfeiçoado nos processadores posteriores. Permite proteção de memória, multitarefa, e acesso a memória superior a 1 MB (até 4 GB em sistemas de 32 bits).

Aqui, o processador pode gerenciar diferentes áreas de memória de forma segura, impedindo que um programa sobrescreva a memória de outro.

**Modo Longo (Long Mode)**:

Este é o modo utilizado pelos processadores modernos de 64 bits (como o x86-64). Ele oferece acesso a um vasto espaço de memória (mais de 4 GB) e melhora a performance através de instruções de 64 bits.

### ****5. Instruções x86****

As instruções da arquitetura x86 são codificadas em **opcodes**, e cada instrução realiza uma operação específica. Algumas das instruções mais comuns incluem:

**MOV**: Move dados entre registradores e memória.

**ADD, SUB**: Operações aritméticas de adição e subtração.

**CMP**: Compara dois valores.

**JMP, JE, JNE**: Instruções de salto (como if, goto).

**PUSH, POP**: Manipulação da pilha, usada para chamadas de função.

**CALL, RET**: Chama uma função ou retorna de uma função.

### ****6. A Pilha (Stack)****

A pilha é uma estrutura de dados crucial na arquitetura x86. Ela é usada para armazenar dados temporários, como variáveis locais e endereços de retorno de funções. O processador x86 usa **ESP** (Stack Pointer) para apontar para o topo da pilha.

A pilha segue o princípio **LIFO** (Last In, First Out), ou seja, o último item inserido é o primeiro a ser removido.

Exemplo de manipulação de pilha em Assembly:

assembly

CopiarEditar

PUSH EAX ; Empilha o valor de EAX

POP EBX ; Desempilha o valor para EBX

### ****7. Programação em Assembly x86****

Se você estiver estudando a arquitetura x86 para escrever código em **Assembly**, é importante compreender a sintaxe de Assembly x86. Em Assembly, cada instrução corresponde a uma operação de baixo nível no processador. Isso permite um controle mais direto sobre os recursos do sistema.

Aqui está um exemplo simples de código Assembly x86 para somar dois números:

assembly

CopiarEditar

section .data

num1 db 5

num2 db 10

section .text

global \_start

\_start:

mov al, [num1] ; Carrega num1 em AL

add al, [num2] ; Soma o valor de num2 a AL

; AL agora contém a soma de num1 e num2

; Aqui você poderia usar uma syscall para imprimir ou finalizar

; A execução do código no Linux

### ****8. Interrupções e Chamadas de Sistema****

As interrupções são um conceito fundamental em arquiteturas de processadores, permitindo que o processador desvie de sua execução normal para tratar eventos externos, como a chegada de dados de um dispositivo de entrada ou uma requisição de sistema.

No contexto de sistemas operacionais como Linux, as **chamadas de sistema** são feitas através de interrupções de software, usando um código de operação específico (como a interrupção 0x80 no Linux x86).

### ****Dicas para sua Apostila:****

**Mapeie as instruções mais importantes**: Liste as instruções de Assembly, como MOV, ADD, CMP, etc., e mostre exemplos de como elas funcionam.

**Inclua diagramas**: Desenhe diagramas da arquitetura x86, mostrando como os registradores se conectam com a ALU e a memória.

**Explique o fluxo de controle**: Explique como o processador lida com as instruções de controle, como saltos (JMP) e chamadas de funções (CALL).

**Crie exemplos práticos**: Mostre pequenos exemplos de código em Assembly x86, explicando cada linha.