

x86 Assembly Mini-Course

Part 1

Bruno P. Evangelista

bpevangelista@gmail.com

Introdução

- Assembly é uma linguagem de programação de baixo nível em formato mnemonico
- Ela possibilita trabalhar diretamente com o processador da máquina, executando operações que muitas vezes não são possíveis em linguagens de alto nível

Porque utilizar?

- Muito utilizado no desenvolvimento de aplicativos que exigem resposta em tempo real
- Tirar proveito de conjuntos de instruções específicas dos processadores: SSE, SSE2, SSE3, 3DNOW, outros
- Obter conhecimento do funcionamento do hardware, visando desenvolver softwares de melhor qualidade

Utilização em softwares

- Muitas vezes os aplicativos precisam de um desempenho maior em partes críticas do código
- Nesses techos devem ser utilizados algoritmos otimizados, com baixa ordem de complexidade
- Mesmo com o uso de um algoritmo adequado, pode não ser possível atingir o tempo de resposta necessário
- Nesses casos, podemos tentar melhorar a performance utilizando otimizações de baixo nível

Problemas

- Otimizações de baixo nível não podem ser programadas em linguagens de alto nível como C/C++ ou Java
- Quando fazemos otimizações de baixo nível, estamos confiando mais em nós mesmo, que no compilador para fazer a geração de código
- Muitas vezes otimizações de baixo nível não são a solução para gargalos no desempenho, sendo necessário trocar a ordem de complexidade do algoritmo utilizado

Ferramentas utilizadas

- Principais:

- Editor de código

- Não existem IDEs com grandes recursos, devido as limitações da linguagem

- Montador

- Utilitários:

- Depurador

- Editor hexadecimal

Montadores

- Um dos primos do compilador
- O papel do montador é transformar os *mnemonics* em código de máquina
- Existem vários montadores para a linguagem Assembly:
 - MASM - Microsoft Assembler
 - NASM - Netwide Assembler
(Versões para diferentes plataformas)
 - TASM - Turbo Assembler (Borland)

Depurador

- Ferramenta extremamente importante para encontrar erros cometidos durante a programação
- Erros cometidos durante a programação geralmente travam a máquina, não sendo possível detectá-lo com facilidade
- Um erro cometido na 10 linha de código geralmente só é percebido centenas de linhas para frente, longe de onde o erro foi cometido

Editor hexadecimal

- Ferramenta importante para trabalhar com arquivos binários
- Utilizada para editar e modificar informações internas de arquivos binários

Inlining Assembly

- Algumas linguagens possuem mecanismos para permitir a inserção de códigos assembly dentro linguagem
- Com isso podemos ter maior controle do noso aplicativo, não precisando manter modulos externos em Assembly que devam ser linkados juntos com a aplicação

Inlining Assembly

- Exemplo do uso de código Assembly na linguagem C/C++

```
__asm() {  
    pusha  
    xor ax, ax  
    popa  
}
```

Arquitetura 8086

| | |
|---------|---|
| CPU | Registradores de 16 bits |
| Memória | Memória limitada a 1MB |
| | Dividida em segmentos 64kb |
| | Somente modo real |
| | Bytes na memória não possuem endereço único |
| | Organização <i>little endian</i> |

Arquitetura 8086

- Endereçamento de memória

$$\text{Endereço} = 16 * \text{segmento} + \text{offset}$$

- Memória endereçável: $2^{20} = 1MB$

Arquitetura 8086

- Mapeamento da memória

| | |
|-------------------|--------------------------------|
| 0x00000 - 0x003FF | Tabela de interrupções (ISR) |
| 0x00400 - 0x005FF | Área de BIOS (BDA) |
| 0x00600 - 0x9FFFF | Área livre |
| 0xA0000 - 0xAFFFF | Memória de vídeo EGA/VGA |
| 0xB0000 - 0xB7FFF | Memória de texto monocromático |
| 0xB8000 - 0xBFFFF | Memória de vídeo CGA |
| 0xC0000 - 0xDFFFF | ROM instalada |
| 0xE0000 - 0xFDFFF | ROM fixa |
| 0xFE000 - 0xFFFFF | ROM da BIOS |

Arquitetura 80386

| | |
|---------|---|
| CPU | Registradores de 32 bits |
| Memória | Memória limitada a 4GB |
| | Modo real e protegido |
| | Bytes na memória possuem endereço único |
| | Organização <i>little endian</i> |

Sintaxes de comando

| Intel | AT&T |
|------------------|--------------------|
| mov eax, 1 | movl \$1, %eax |
| mov ebx, 0ffh | movl \$0xff, %ebx |
| int 80h | int \$0x80 |
| mov eax, [ebx] | movl (%ebx), %eax |
| mov eax, [ebx+3] | movl 3(%ebx), %eax |

Diretiva de dados

- Diretivas de dados são utilizadas para definir espaço na memória
- Existem duas diretivas de dados

| Diretivas de dados | Descrição |
|--------------------|--|
| RESX | Define espaço para memória |
| DX | Define espaço para memória e inicializa a área |

- O símbolo X deve ser trocado pelo tipo de dado desejado

Diretiva de dados

| Tipos de dado | Letra | Tamanho(bits) |
|---------------|-------|---------------|
| byte | B | 8 |
| word | W | 16 |
| double word | D | 32 |
| quad word | Q | 64 |
| ten bytes | T | 80 |

Registradores

- A arquitetura 8086 possui 14 registradores de 16 bits, sendo:
 - 4 de propósito geral
 - 5 de *offset* para acesso a memória
 - 4 de *segmento* para acesso a memória
 - 1 registrador de *flags*

Registradores

- Registradores de propósito geral

| Registrador | Propósito geral |
|-------------|-----------------|
| AX | Acumulador |
| BX | Base |
| CX | Contador |
| DX | Dados |

Registadores

- Registadores de offset para acesso a memória

| Registrador | Offset |
|-------------|---------------------|
| IP | Instruction pointer |
| SP | Stack pointer |
| BP | Base pointer |
| SI | Source Index |
| DI | Destination Index |

Registadores

- Registradores de segmento para acesso a memória

| Registrador | Segmento |
|-------------|---------------|
| CS | Code segment |
| SS | Stack Segment |
| DS | Data Segment |
| ES | Extra segment |

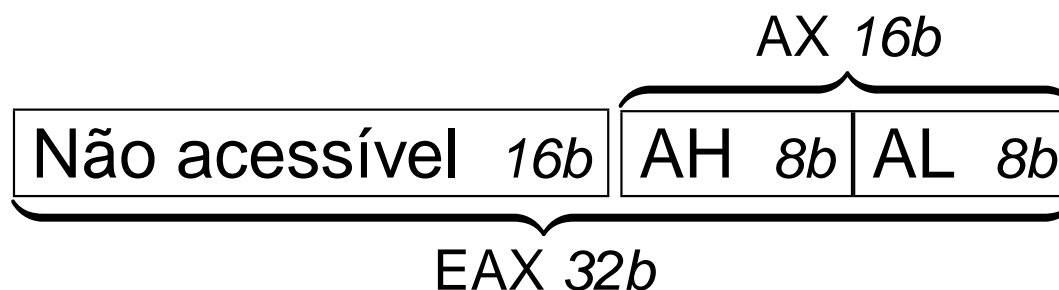
Registadores

- Registrador de *flags*

| Flags | Descrição |
|-------|----------------|
| OF | Overflow flag |
| DF | Direction flag |
| IF | Interrupt flag |
| TF | Trap flag |
| SF | Sign flag |
| ZF | Zero flag |
| AF | Auniliary flag |
| PF | Parity flag |
| CF | Carry flag |

Registradores

- A arquitetura 80386 possui registradores de *32 bits* que estendem os registradores anteriores mantendo compatibilidade
- Os registradores de *32 bits* possuem o prefixo “E”



Instruções

| Instrução | Descrição |
|-----------|----------------------|
| mov | Move dados |
| add | Adição aritmética |
| sub | Subtração aritmética |
| push | Empilha |
| pop | Desempilha |
| jmp | Salto incondicional |
| int | Interrupção |
| call | Chamada |

Interrupções

- São chamadas de funções externas
- A tabela interrupções se encontram no início da memória, e é chamada ISR(Interrupt Service Routine)
- A ISR possui o endereço das funções de cada interrupção, contendo o segmento e offset de cada uma delas
- Quando o comando: *int 21h* é executado, estamos na verdade saltando para o endereço de memória que contém o código da interrupção

Interrupções da BIOS

| Interrupção | Descrição |
|-------------|---|
| 10h | Escreve caracter |
| 13h | Reseta drive Lê do disco Escreve no disco |
| 16h | Lê caracter |
| 18h | Chama "Basic" |
| 19h | Reseta |

x86 Assembly Mini-Course

Part 3

Bruno P. Evangelista

bpevangelista@gmail.com

Sistema de arquivos

- Quando trabalhamos com arquivos, necessitamos de um mecanismo capaz de manipular os mesmos, chamado *sistema de arquivos*
- O sistema de arquivos deve disponibilizar as operações necessárias para manipulação arquivos, como:
 - Listar
 - Leitura
 - Gravação

Qual sistema utilizar?

- Atualmente existem diversos sistemas de arquivos, cada um para um propósito específico
 - FAT12, FAT32, NTFS
 - VFAT, EXT3, EXT3, REISER
- Os sistemas de arquivos são desenvolvidos baseado no dispositivo que será utilizado
- Alguns formatos não são adequados para dispositivos de pequenos porte, pois utilizam muito espaço para fazer o gerenciamento dos arquivos

Qual sistema utilizar?

- Um sistema de arquivos simples pode ser facilmente criado, no entanto, não isso demanda tempo
- Ao utilizarmos um sistema já existente, podemos fazer uso das diversas ferramentas e recursos disponíveis para o mesmo

Sistema FAT12

- FAT12 é um sistema de arquivos TODO
- Na FAT12 existe um ponteiro para o primeiro bloco de cada arquivo e para cada bloco lido existe, um ponteiro para o próximo bloco
- O sistema FAT12 é formado por:
 - Duas cópias da tabela de alocação de arquivos(FAT)
 - Tabela de entrada de arquivos
 - Dados dos arquivos

Estrutura da FAT12

- Dos 2.880 setores disponíveis, 1 é utilizado para boot, 32 para cabeçalhos, e o resto para dados

| Setor | Utilização | Espaço |
|---------|----------------|--------|
| 0 | Boot sector | 512b |
| 1-9 | FAT Copy 1 | 4,5kb |
| 10-18 | FAT Copy 2 | 4,5kb |
| 19-32 | Root Directory | 7kb |
| 33-2879 | Data | 1,39mb |

Cabeçalho do boot para FAT12

- A FAT12 utiliza o início do setor de boot para guardar as configurações do dispositivo e verificar se o sistema é válido

| | | |
|-------------------|----|------------|
| OSID | db | 'EVANG_OS' |
| bytesPerSector | dw | 0x0200 |
| sectorsPerCluster | db | 0x01 |
| leadingSectors | dw | 0x0001 |
| numFAT | db | 0x02 |
| maxRootDirEntries | dw | 0x00E0 |
| totalSectors | dw | 0x0B40 |
| mediaType | db | 0x0F0 |

Cabeçalho do boot para FAT12

| | | |
|-----------------|----|----------------|
| sectorsPerFAT | dw | 0x0009 |
| sectorsPerTrack | dw | 0x0012 |
| numberOfHeads | dw | 0x0002 |
| hiddenSectors | dd | 0x00000000 |
| totalSectors2 | dd | 0x00000000 |
| driveNumber | db | 0x00 |
| reserved | db | 0x00 |
| bootSignaure | db | 0x29 |
| volumeID | dd | 0x26185454 |
| volumeLabel | db | 'EVANG_OSbbb' |
| volumeLabel | db | 'OST4bbbbbbbb' |
| FATID | db | 'FAT12bbb' |

Endereçamento da FAT12

- Cada entrada da tabela FAT possui 12 bits
- Com isso temos $2^{12} = 4096$ (4kb) possíveis endereços de setor
- Na FAT12 cada setor possui 512 bytes
- O que disponibiliza um endereçamento total de 2mb

Entradas da FAT

- Cada entrada da FAT12 possui 12 bits
- Por isso para acessar cada entrada é necessário ler dois bytes, mas apenas 1 byte e meio é utilizado
- Por causa disso é preciso um tratamento especial para as entradas na FAT

Entradas da FAT

- Trecho da FAT exibida em hexadecimal, cada caracter representa um *nibble* de 4 bits

| | | | | |
|----|----|----|----|----|
| 01 | 23 | 45 | 67 | 89 |
|----|----|----|----|----|

- Endereço 0 endereço: 0 1 2
- Endereço 1 endereço: 3 4 5
- Endereço 2 endereço: 6 7 8

Entradas de arquivo

- Para localizarmos os arquivos em disco é necessário consultar a tabela de arquivos

| Byte | Descrição |
|-------|------------------------------|
| 0-7 | Nome do arquivo |
| 8-10 | Extensão |
| 11 | Atributos |
| 12-21 | Reservados |
| 22-23 | Hora |
| 24-25 | Data |
| 26-27 | Endereço do primeiro cluster |
| 28-31 | Tamanho do arquivo |

Modos de endereçamento

- O tipo de endereçamento utilizado pela FAT12 é diferente do utilizado pela controladora do disco
- Basicamente existem dois tipos de endereçamento
 - CHS - Cylinder, Head, Sector
 - LBA - Logical block address
- A controladora do disco trabalha com o endereçamento CHS

Modos de endereçamento

- A BIOS possui interrupções para trabalhar com LBA, no entanto o código utilizado é grande, por isso geralmente é utilizado o endereçamento CHS no boot
- Para acessarmos o endereço do arquivo é necessário então fazer uma conversão entre os tipos de arquivo, como ilustra o desenho abaixo

FAT12 → LBA → CHS

Transformação LBA para CHS

- $\text{Sector} = \text{LBA} \% \text{SectorsPerTrack} + 1$
- $\text{Head} = (\text{LBA} / \text{SectorsPerTrack}) \% \text{NumberOfHeads}$
- $\text{Cylinder} = (\text{LBA} / \text{SectorsPerTrack}) / \text{NumberOfHeads}$

Transformação FAT para LBA

- O FAT começa a endereçar a partir da área livre, setor 33 (0 a 32 estão ocupados)
- Mas é importante lembrar que os dois primeiros endereços de entrada do FAT são reservados
- Então o endereço do bloco 33 na FAT12 é 2
- Portanto para converter um FAT para LBA basta somar 31 ao número do bloco

Funções básicas do driver

- FAT12 START
 - Carrega todos os cabeçalhos do sistema para a memória, em uma área segura
- FAT12 READ
 - Carrega um arquivo pelo nome, e grava em uma área disponível da memória
- LBA READ
 - Carrega um setor LBA para a memória

Testando o driver

- Crie vários arquivos com 512 caracteres, cada um formado por uma sequência do mesmo caracter
 - aaaaaaaaaa
 - bbbbbbbb
- Execute a operação de leitura e em seguida de exibição para cada um dos arquivos e veja se o resultado está correto
- Concatene os arquivos em grupos de 3, tamanho 1.5kb, e execute a operação de leitura e exibição novamente
- Se após vários testes tudo ocorreu bem, seu sistema de arquivos está funcionando corretamente

Informações adicionais

- Google - www.google.com
 - Um dos seus melhores amigos, não deixe de perguntar para ele! =D
- Intel - www.intel.com
 - Disponibiliza manual sobre a arquitetura x86, além de várias ferramentas para desenvolvedores