Objetivos:

***Explicar de forma didática o que seria o Bootloader de 2 fases;***

***Falar dos registradores da arquitetura x86, endereçamento, alguns comandos básicos de ASM (jmp, mov...);***

***BIOS;***

***Exemplo – (código e execução);***

***Roteiro:***

***(O início do vídeo será reservado para a apresentação do conceito do Bootloader, e o seu funcionamento.)***

- O Bootloader é o primeiro programa a ser carregado e executado pelo processador, e possui a responsabilidade de organizar e levantar o S.O. Ele que vai passar o processador para o modo protegido, e carregar o Kernel na memória.

O Bootloader é inicializado em duas etapas (ou mais) para ter acesso a mais memória (Se fosse realizado em apenas uma etapa, só teríamos disponível 512 bytes, o que é insuficiente para toda a execução).

No primeiro estágio de inicialização, a BIOS (Que consiste num código armazenado na memória não volátil ligada a placa mãe do PC) carrega o primeiro setor do disco de assinatura 0xAA55 no endereço de memória linear 0x7C00, e realiza um JMP (Instrução de código Assembly) para esse endereço.

No segundo estágio, serão carregadas algumas estruturas que serão úteis ao Kernel de memória, em seguida carrega o próprio Kernel na memória, sai do modo real e passa para o modo protegido, e passa o controle pro Kernel.

***(Em seguida, será explicado o que é um processador de arquitetura x86 e o uso de seus registradores)***

***-*** x86 é o nome (genérico) dos processadores que foram baseados no Intel 8086 (para que pudessem passar a registrar a marca, pararam de usar tal denominação e passaram a dar nomes como Pentium e etc). A arquitetura x86 possui:

- 8 registradores gerais

- AX: (accumulator register) usado em operações aritméticas

- CX: (counter register) usado em operações de shift e loops

- DX: (data register) usado em operações aritméticas e de entrada e saída

- BX: (base register) usado como ponteiro de dados

- SP: (stack pointer register) aponta para o topo da pilha

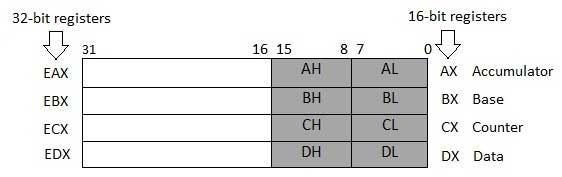
- BP: (stack base pointer register) aponta para a base da pilha

- SI: (source index register) usado como fonte em operações de memória

- DI: (destination index register) usado como destino em operações de memória

É importante ressaltar que o processador baseado na arquitetura x86 é Little Endian, ou seja, a ordem crescente dos bits de dados se dá no mesmo sentido que a ordem crescente dos endereços dos bytes na memória.(Ou seja, o bit mais significativo é o bit mais à esquerda). É possível acessar ‘partes’ dos registradores trocando o X por H ou L (high ou low), como na foto abaixo:

~exibir a foto abaixo para ilustrar o uso do H e L no acesso aos registradores~



- 6 registradores de segmento

- SS: (stack segment): ponteiro para a pilha

- CS: (code segment): ponteiro para o código

- DS: (data segment): ponteiro para os dados

- ES/FS/GS: ponteiros extras para dados

- 1 registrador de flag

Guarda bits representando o estado do processador e resultados de operações.

- 1 ponteiro de instrução:

Contém o endereço da próxima instrução a ser executada caso não haja desvio.

***(Com a explicação dos registradores x86, serão apresentados os primeiros comandos em ASM, o uso de seus registradores e endereçamentos)***

**Declarando variáveis (reservando espaços de memória)**

**site legal →** [**https://www.oocities.org/codeteacher/x86asm/asml1002.html**](https://www.oocities.org/codeteacher/x86asm/asml1002.html)

*nome* db *conteudo*

*myvar db ‘variavel’, 0 → uma variável (de tamanho 1 byte) com a string ‘variável’ e um 0 ao final (para indicar o fim da string)*

*nome* times *tamanho* db *conteudo*

myvar times 50 db 0

**Instruções**

Cada instrução pode ter de 0 a 3 operandos, que são separados por vírgula. Os operandos das instruções podem ser IMEDIATOS (valores constantes), REGISTRADORES (valor guardado nos registradores) ou MEMÓRIA (valores guardados em memória).

Elas podem ser de transferência de dados (mov), operações aritméticas (add, sub, and, or), controle do fluxo do programa (jmp, cmp, je, jne, jge, jl, call, ret) ou instruções/chamadas ao sistema (int) .

**Transferência de dados**

**mov destino, fonte →** move dados da memória/registrador ou imediato para a memória/registrador

**lodsb →** carrega os dados (byte) apontados pelo reg SI no reg AL

**stosb →** coloca no endereço indicado pelo reg DI o conteúdo (byte) do reg AL

**Operações aritméticas**

**add destino, fonte→** o destino é um registrador ou espaço de memória e a fonte pode ser um registrador, espaço de memória ou imediato. O resultado é armazenado no destino

**xor destino, fonte →** realiza a operação xor (ou exclusivo) bit a bit entre o destino e a fonte e armazena o resultado no destino. Enquanto que a fonte pode ser um imediato/registrador/memória, o destino pode ser apenas registrador/memória. Note que ambos não podem ser memória.

**inc destino →** incrementa o registrador/memória de destino em 1

**outras: sub, or, and, dec**

**Controle de fluxo**

**jmp endereço →** pula para o endereço indicado (usado para pular para outras funções/instruções do código

**cmp registrador/valor, registrador/valor →** compara os valores inseridos e deve ser seguido por uma instrução de ‘pulo’

**je endereço →** caso o cmp anterior seja ‘igual’, o ‘jump if equal’ é ativado e a próxima instrução passa a ser a indicada pelo endereço (ou nome de função)

**jne (jump if not equal), jge (jump if greater than or equal), jl (jump if less than)** são ativados igualmente a depender do resultado do cmp anterior:

**call endereço →** chama uma função. Diferente do jump, a execução do programa retornará ao próximo comando após executar a função.

**ret** → retorna (É a instrução responsável por retornar a execução do processo (programa) para a função que a chamou. Semelhante à função **return** de linguagens de alto nível como C,C++ e Python)

**Chamadas ao Sistema**

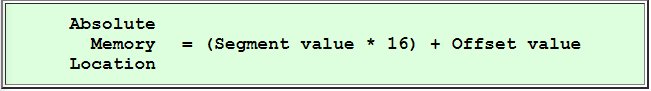
**int código →** interrompe o programa para solicitar entrada/saída ao sistema operacional

int 10h - escreve na tela o conteúdo de AL

int 16h - recebe a entrada do teclado

**Endereçamento**

O endereçamento em x86 se dá pela forma do SEGMENT:OFFSET. É uma forma relativa de endereçar a memória (vários endereços podem indicar o mesmo local na memória). Indica-se a base (segment) e o valor a partir dele que é o desejado (offset). Usando a seguinte fórmula se pode calcular o endereço absoluto:

~exibir a fórmula abaixo~ 

Como o mais comum é que o endereço esteja em hexadecimal, para realizar a multiplicação por 16 basta acrescentar um zero à direita (1 shift left) e depois somar ao valor do offset.

Exemplo: endereço relativo → 8ACE:3CFF

endereço absoluto → 8ACE0+3CFF = 8E9DF

***(Apresentação da BIOS)***

* A BIOS (Basic Input/Output System) é de maneira simples e direta, o primeiro programa a ser executado pelo sistema computacional.

**Note a diferença entre o Bootloader e a BIOS:**

O Bootloader é o primeiro programa a ser CARREGADO E EXECUTADO pelo processador.

A BIOS é diretamente executada pelo processador. O motivo disso é que a BIOS não se encontra numa parte comum da memória, ela está numa parte especial chamada ROM. Essa parte da memória é não volátil, significa que diferente da memória RAM, cache, e dos registradores, ela não depende da energia elétrica e se manter mesmo que o computador seja desligado e não pode ser sobrescrita.

* A BIOS é também responsável pela execução do POST(POWER-ON-SELF-TEST), que é uma série de testes ao hardware do computador. Se for detectado algum problema, uma série de bips sonoros é emitida.
* A última função da BIOS é carregar o Bootloader na memória.

***(Ao fim será mostrado e executado um código de exemplo em ASM-x86 e feito um resumo de tudo o que foi apresentado)***

* Será apresentado o seguinte código:

https://github.com/hcct-cin/Bootloader-Asmx86