**Originalmente escrito por plas @ cin.ufpe.br**

- BIOS
    - Primeiro programa a ser executado
    - Onde fica a bios?
        - Numa memoria especial fixa - ROM.
            - A bios é acessada pelo processador por meio de um mapeamento da memória principal (ram) para a fixa (não volátil, rom)
            - Esse **mapeamento** é fixo na ram - fica lá em cima naquela imagem clássica da low ram x86)
            - Mapeamento = certos endereços da memória principal apontam diretamente para certos endereços da bios.
            - Pelo que me lembro, é assim: Quando liga-se o pc, o processador lê um “ponteiro de reset”, que aponta justamente para o primeiro endereço mapeado da bios na ram, fazendo com que o processador leia uma parte inicial da bios e vá executando suas instruções até carregar o bootloader.
            - resumo: start -> reset pointer -> cpu lê parte da bios mapeada na ram -> “faz as parada” (executa as instruções dessa parte da bios, como POST, por exemplo) -> carrega bootloader.
    - POST - verificação do hardware
    - processo de boot iniciado pela bios
- Bootloader
    - Processo de carregamento da segunda etapa e o motivo principal de existir uma segunda etapa
    - Processo do carregamento do kernel pela segunda etapa
    - Onde fica o bootsector (disco! -não na ram- O que fica na ram é tipo um espaço reservado para o que virá do bootsector) (me refiro a ram no sentido de memória principal)
    - assinatura de boot
- Interrupções
    - Processo como se dá uma system call (o programa é interrompido, salva-se o contexto, executa-se a interrupção e depois o programa é retomado)
        - Da bios (só mencionar que existem os dois tipos)
        - Do SO
- Modos de operação do processador (citar o motivo de ter-se criado o modo protegido e que o modo real sempre é o modo pelo qual o pc começa)
    - modo real
    - modo protegido (foi criado, entre outros motivos, para dar maior suporte aos programas, segurança, e aumentar o espaço de endereçamento)
- Arquitetura x86

    - Little endian ou big endian? (little. A ordem crescente dos bits dos dados se dá no mesmo sentido que a ordem crescente dos endereços dos bytes na memória)
    - Registradores de segmento. Quais são e pra quê servem. Exemplos de códigos onde estão implicitamente referenciados.
        - mov bx, ax (somente o cs está sendo implicitamente utilizado, pois ele SEMPRE É IMPLICITAMENTE UTILIZADO nas instruções -já que é por meio de CS:IP que a cpu realiza o *fetch* das instruções.
        - mov bx, [ax] -->(estão sendo implicitamente utilizados o ds e o cs)
            - = mov bx, [ds:ax]
            - vale frisar que **mov bx, [ax] = mov bx, [ds:ax]**
        - push ax → CS e SS
    - Segment:offset e desenvolver exemplos de combinações para se chegar a um mesmo endereço.
    - Cada segmento tem 64KB e existem 16 segmentos (= 1mb de espaço de endereçamento no modo real)
- Aspectos de programação
    - Procedimentos (o uso de call e ret)
    - Procedimento vs Macro (vantagens e desvantagens de um e de outro)
    - pseudo-instrução **times** (vai cair questão sobre essa pseudo-instrução)
    - DB e DW. Qual a diferença de se fazer db ‘a’ e dw ‘a’ ?? (o primeiro caso aloca ‘a’ em 1 byte. O segundo caso aloca ‘a’ em 2 bytes.)

Extra para quem estiver perdido sobre o que se deve fazer para criar o bootloader do projeto:

Vamos fornecer um script e a especificação.
Antes de saber para que serve o script, você deve saber como funciona um bootloader usual para a arquitetura x86.

Em linhas gerais, quando o pc inicia, o processador é resetado e com isso inicia a BIOS, por meio do mapeamento que há na memória principal para ela.
Por meio da BIOS, o processador executa o POST, que consiste numa série de verificações no ambiente computacional, e depois procura por um boot válido (programa em asm assinado com a word 0xAA55).
Geralmente espera-se que esse boot válido esteja no disco (HD). Daí, por padrão, a bios verifica (= o processador está executando o código da bios e esse código faz uma verificação) se o que estiver nos primeiros 512bytes do setor 0, trilha 0 e cabeça 0 do HD é um boot válido. Esses 512 bytes dessa posição são o bootsector. Daí, se o bootsector contiver um boot válido, a BIOS carrega os tais 512 bytes para o endereço 0x7c00 da memória principal. Dessa forma o processador executa o que está nesse código.

O que vocês vão ter que fazer é criar um código desse tipo. Bootável. Porém,  o objetivo desse código é carregar um outro código, também presente no disco (porém em outro setor, não no bootsector) ao qual chamaremos de segundo estágio do bootloader.

Ou seja, qemu carrega disco. BIOS emulada carrega boot1  e boot1 carrega boot2.
Ainda acrescento que o boot2 deve carregar o kernel, que será um programa desenvolvido por vocês. Totalizando:
qemu carrega disco. **BIOS** emulada pelo qemu carrega **boot1** e boot1 carrega **boot2** e boot2 carrega **kernel**.

O procedimento de carregamento é simples:

- reseta-se o leitor do disco (via interrupção da bios)
- Lê-se um número de setores desejado, carregando-os para alguma posição na memória.
- Modifica-se o CS:IP para que o código carregado comece a ser executado (=pula-se para onde foi carregado o código)

Sabendo de tudo isso, pode-se passar para o que faz o script fornecido.
O script serve para criar o disco que o qemu vai considerar como HD da máquina virtual. Daí esse script escreve nos setores do disco o boot1, o boot 2 e o kernel que o grupo desenvolveu. Sendo assim, toda vez que for rodar o projeto, o grupo deve ter os arquivos boot1, boot2, kernel e script. E para rodar o projeto, deverão executar o script, um arquivo imagem será criado, e rodar o qemu com esse arquivo de imagem como parâmetro.

PS: No meu public há materiais básicos sobre nasm (tasm tb) e alguns mapas mentais de assuntos específicos da cadeira de software. Pode ajudar (no cap de deadlock ajuda!)

http://www.cin.ufpe.br/~plas
http://powerplas.site44.com
https://dl.dropboxusercontent.com/s/fvdnd7u0ccjmidj/plas_map.swf

Abraço, powerplas.