#1
```
make lab1
make qemu-gdb
```
#2
```
gdb // tui enable
qemu
debug-loader
break *0x7c00
continue
```

0. Boot_entry()
> break *0x7c00
Note: breakpoint 1 also set at pc 0x7c00: file boot.S, line 17.

Set breakpoint to <boot_entry> cli - Clear Interrupt-Enable Flag

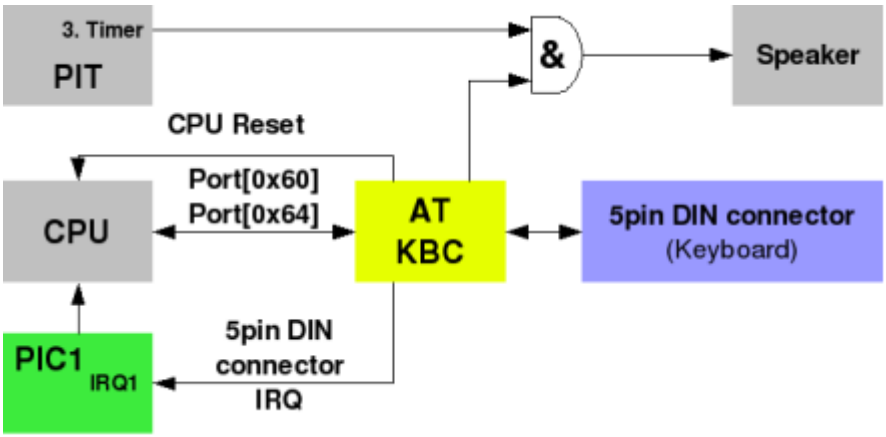
```
    | // first of all: disable interrupts
B+  | cli
```

Эта команда сбрасывает interrupt flag (IF) в регистре EFLAGS. Когда этот флаг сброшен процессор игнорирует все прерывания (кроме NMI) от внешних устройств.

(gdb) b memory_detected
Breakpoint 3 at 0x7c54: file boot.S, line 81.
(gdb) c
Continuing.

Breakpoint 3, memory_detected () at boot.S:81
(gdb) si
wait_8042 () at boot.S:46
boot_entry () at boot.S:29
wait_8042 () at boot.S:46
boot_entry () at boot.S:35
wait_8042 () at boot.S:46
boot_entry () at boot.S:40

 # Source: https://wiki.osdev.org/%228042%22_PS/2_Controller
1. Function wait_8042()



Overview of the AT-Controller

The 8042 was a powerful micro-controller. To reduce costs, some of the general purpose input/output capabilities of the AT controller was used to control various functions unrelated to the keyboard, including:

- System Reset
- The A20-Gate

;Wait for a empty Input Buffer
```
wait_8042:
in   al, 0x64
test al, 00000010b
jne  wait_8042
```

2. Enable A20:

```
48          a20_enabled:
49
50          // Store memory map into 0x7e00 (0x7c00 + 512)
51          movw $0x7de8, %di // 0x7e00 – 24
```

512 = 2^9 = 200h

 # Source:
https://wiki.osdev.org/Detecting_Memory_(x86)#BIOS_Function:_INT_0x15.2C_EAX_.3D_0xE820

3. Detect high memory

Getting an E820 Memory Map

```
; use the INT 0x15, eax= 0xE820 BIOS function to get a memory map
; note: initially di is 0, be sure to set it to a value so that the BIOS code
will not be overwritten.
;       The consequence of overwriting the BIOS code will lead to problems like
getting stuck in `int 0x15`

do_e820:
        mov di, 0x8004          ; Set di to 0x8004. Otherwise this code will get
                              stuck in `int 0x15` after some entries are fetched
        xor ebx, ebx            ; ebx must be 0 to start
        xor bp, bp              ; keep an entry count in bp
        mov edx, 0x0534D4150    ; Place "SMAP" into edx
        mov eax, 0xe820
        mov [es:di + 20], dword 1       ; force a valid ACPI 3.X entry
        mov ecx, 24             ; ask for 24 bytes
        int 0x15
```

Basic Usage:
For the first call to the function, point ES:DI at the destination buffer for the list. Clear EBX. Set EDX to the magic number 0x534D4150. Set EAX to 0xE820 (note that the upper 16-bits of EAX should be set to 0). Set ECX to 24. Do an INT 0x15.

If the first call to the function is successful, EAX will be set to 0x534D4150, and the Carry flag will be clear. EBX will be set to some non-zero value, which must be preserved for the next call to the function. CL will contain the number of bytes actually stored at ES:DI (probably 20).

For the subsequent calls to the function: increment DI by your list entry size, reset EAX to 0xE820, and ECX to 24. When you reach the end of the list, EBX may reset to 0. If you call the function again with EBX = 0, the list will start over. If EBX does not reset to 0, the function will return with Carry set when you try to access the entry after the last valid entry.

4. Какой командой загрузчик переключает процессор в защищённый режим?

boot.S:81

// enable protected mode (set first bit in cr0)

```
movl %cr0, %eax

orl  $CR0_PE_ON, %eax

movl %eax, %cr0
```

5. Как первый загрузчик инициализирует регистр указателя стека? Где находится стек при работе загрузчика?

// setup stack, and start our C part of the bootloader

```
movl $boot_entry, %esp
```

6. Как устроена GDT загрузчика?

gdt:

```
SEG(0x0, 0x0, 0x0) // null seg

SEG(UST_X|USF_D|USF_P|USF_S|USF_G|UST_R, 0x0, 0xfffff) // code seg

SEG(USF_D|USF_P|USF_S|USF_G|UST_W, 0x0, 0xfffff) // data seg
```

7. Начиная с какого физического адреса загружается первый загрузчик?

0x7c00

8. Начиная с какого физического адреса загружается второй загрузчик? А с какого виртуального?

0x10000 (kernel/boot/main.c)

```
#define KERNEL_LOADER_ADDRESS    0x10000
```

9. Как загрузчик выделяет память для чтения второго загрузчика с диска?

10. Какая последняя исполняемая команда первого загрузчика

    call (*di)

11. Какая первая исполняемая команда второго загрузчика?

    mov $boot_stack_top, %esp

Как работает

Начинает в реальном режиме, переходит в защищенный: открывает линию A20 через контроллер PS/2, сохраняет memory map путем использования INT 15h с AX=E820h, инициализирует GDT -- дескрипторы для сегмента кода, данных и стека. Затем переходит в C часть первого загрузчика. C часть читает ELF со вторым загрузчиком с диска через ATA и запускает его.