1. NEMU在什么时候进入了保护模式?

在config.h中 #define IA32_SEG 的时候引入

2. 在GDTR中保存的段表首地址是虚拟地址、线性地址、还是物理地址？为什么?

是线性地址。

机器开机后首先进入实模式，加载操作系统，操作系统初始化段表，拨动一个‘开关’，从实模式切换到保护模式（开启分段机制），进入保护模式后，程序给出48位逻辑地址（16位段选择符 + 32位有效地址），使用段选择符来查段表，进行段级地址转换得到线性（现在就是物理）地址

代码修改

/include/config.h

```
#define IA32_SEG
```

/nemu/include/cpu/reg.h

```
typedef struct {
    uint32_t limit :16;
    uint32_t base :32;
}GDTR;

typedef union {
    struct {
        uint32_t pe :1;
        uint32_t mp :1;
        uint32_t em :1;
        uint32_t ts :1;
        uint32_t et :1;
        uint32_t reserve :26;
        uint32_t pg :1;
    };
    uint32_t val;
}CR0;

typedef struct {
    // the 16-bit visible part, i.e., the selector
    union {
        uint16_t val;
        struct {
            uint32_t rpl :2;
            uint32_t ti :1;
            uint32_t index :13;
        };
    };

    // the invisible part, i.e., cache part
    struct {
        uint32_t base;
        uint32_t limit;
```

```
        uint32_t type :5;
        uint32_t privilege_level :2;
        uint32_t soft_use :1;
    };
}SegReg;
```

/nemu/src/cpu/cpu.c

```
#ifdef IA32_SEG
    cpu.cr0.val = 0x0;
    cpu.gdtr.base = cpu.gdtr.limit = 0x0;
    for (i = 0; i < 6; i++)
    {
        cpu.segReg[i].val = 0x0;
    }
#endif
```

/nemu/include/memory/mmu/segment.h

```
typedef union SegmentDescriptor {
    struct
    {
        uint32_t limit_15_0 : 16;
        uint32_t base_15_0 : 16;
        uint32_t base_23_16 : 8;
        uint32_t type : 4;
        uint32_t segment_type : 1;
        uint32_t privilege_level : 2;
        uint32_t present : 1;
        uint32_t limit_19_16 : 4;
        uint32_t soft_use : 1;
        uint32_t operation_size : 1;
        uint32_t pad0 : 1;
        uint32_t granularity : 1;
        uint32_t base_31_24 : 8;
    };
    uint32_t val[2];
} SegDesc;

uint32_t segment_translate(uint32_t vaddr, uint8_t sreg);
void load_sreg(uint8_t sreg);
```

/nemu/src/memory/mmu/segment.c

```
uint32_t segment_translate(uint32_t offset, uint8_t sreg)
{
    /* TODO: perform segment translation from virtual address to linear address
     * by reading the invisible part of the segment register 'sreg'
     */
    return cpu.segReg[sreg].base + offset;
}

// load the invisible part of a segment register
```

```
void load_sreg(uint8_t sreg)
{
    /* TODO: load the invisibile part of the segment register 'sreg' by reading
the GDT.
     * The visible part of 'sreg' should be assigned by mov or ljmp already.
     */
     SegDesc desc;
     desc.val[0] = laddr_read(cpu.gdtr.base+8*cpu.segReg[sreg].index,4);
     desc.val[1] = laddr_read(cpu.gdtr.base+8*cpu.segReg[sreg].index+4,4);
     cpu.segReg[sreg].base = desc.base_15_0 | (desc.base_23_16<<16) |
(desc.base_31_24<<24);
     cpu.segReg[sreg].limit = desc.limit_15_0 | (desc.limit_19_16<<16);
     assert(cpu.segReg[sreg].base == 0);
     assert(cpu.segReg[sreg].limit == 0xfffff);
     assert(desc.granularity == 1);
}
```

/nemu/src/memory/memory.c

```
uint32_t vaddr_read(vaddr_t vaddr, uint8_t sreg, size_t len)
{
    assert(len == 1 || len == 2 || len == 4);
    if(cpu.cr0.pe){
        vaddr = segment_translate(vaddr, sreg)
    }
    return laddr_read(vaddr, len);
}

void vaddr_write(vaddr_t vaddr, uint8_t sreg, size_t len, uint32_t data)
{
    assert(len == 1 || len == 2 || len == 4);
    if(cpu.cr0.pe){
        vaddr = segment_translate(vaddr, sreg)
    }
    laddr_write(vaddr, len, data);
}
```

/nemu/src/cpu/instr/lgdt.c

```
make_instr_func(lgdt){
    int len = 1;
    opr_dest.data_size = data_size;
    opr_src.data_size = 16;
    len += modrm_rm(eip+1, &opr_src);
    modrm_rm(eip+1, &opr_dest);
    opr_dest.addr += 2;
    operand_read(&opr_dest);
    operand_read(&opr_src);
    cpu.gdtr.base = opr_dest.val;
    cpu.gdtr.limit = opr_src.val;
    return len;
}
```

/nemu/src/cpu/instr/jmp.c

```
make_instr_func(jmp_far_imm){
    opr_dest.type = opr_src.type = OPR_IMM;
    opr_dest.sreg = opr_src.sreg = SREG_CS;
    opr_dest.data_size = data_size;
    opr_dest.addr = eip + 1;
    opr_src.data_size = 16;
    opr_src.addr = eip + (1 + data_size/8);
    operand_read(&opr_src);
    operand_read(&opr_dest);
    cpu.cs.val = (uint16_t)opr_src.val;
    load_sreg(SREG_CS);
    cpu.eip = opr_dest.val;
    return 0;
}
```

/nemu/src/cpu/instr/mov.c

```
make_instr_func(mov_rm2s_w){
    int len = 1;
    opr_src.data_size = opr_dest.data_size = data_size;
    len += modrm_r_rm(eip + 1, &opr_dest, &opr_src);
    opr_dest.type = OPR_SREG;
    instr_execute_2op();
    return len;
}


make_instr_func(mov_c2r_l){
    int len = 1;
    opr_src.data_size = opr_dest.data_size = data_size;
    len += modrm_r_rm(eip + 1, &opr_dest, &opr_src);
    opr_src.type = OPR_CREG;
    instr_execute_2op();
    return len;
}

make_instr_func(mov_r2c_l){
    int len = 1;
    opr_src.data_size = opr_dest.data_size = data_size;
    len += modrm_r_rm(eip + 1, &opr_dest, &opr_src);
    opr_src.type = OPR_CREG;
    instr_execute_2op();
    return len;
}
```