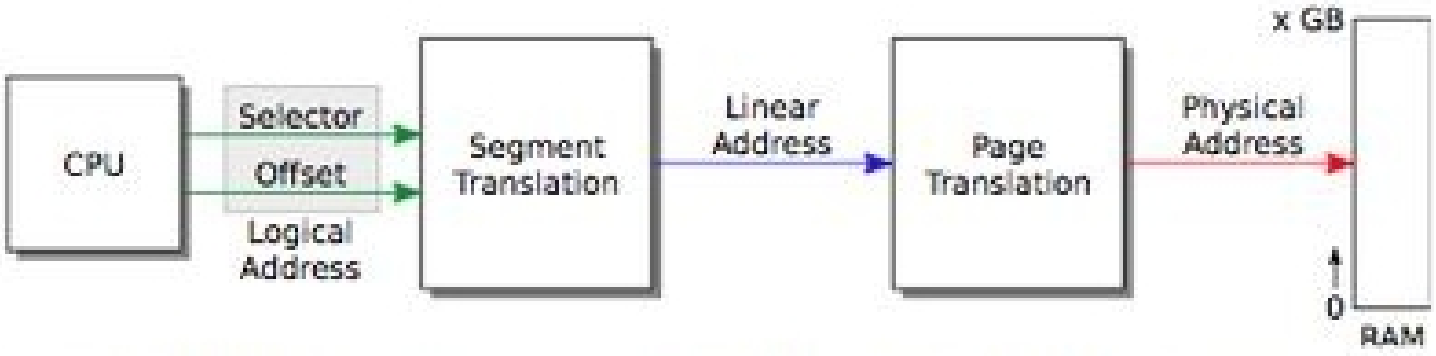
Computer Boot To Protected Mode And C Program

x86

Environment

- Linux: ubuntu 18.04 LTS
- GCC
  - sudo apt-get -y install build-essential libelf-dev binutils-dev
- Bochs
  - sudo apt-get install bochs
  - sudo apt-get install bochs-x

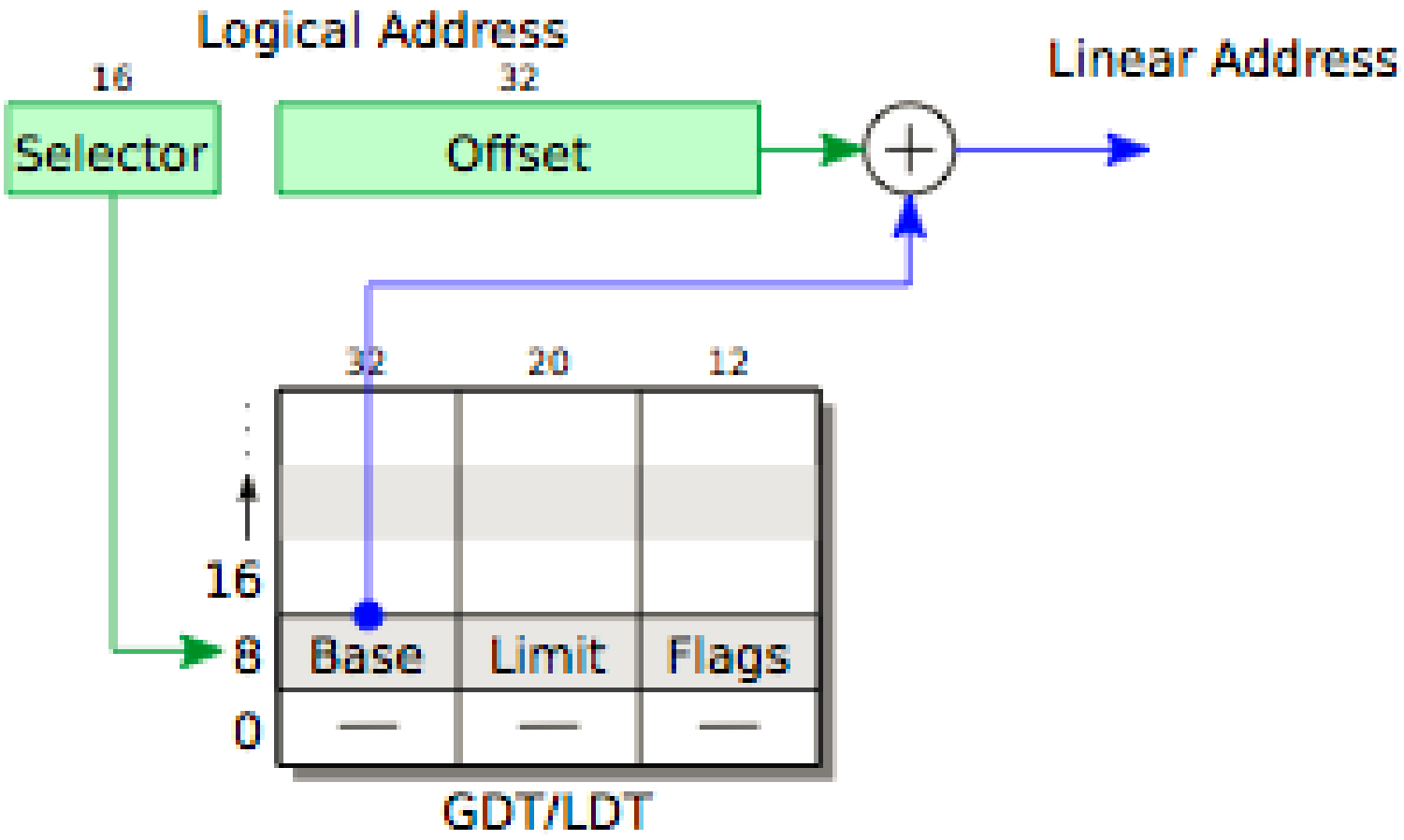
Memory Address



- Logical Address
- Linear Address
- Physical Address

- Real Mode
  - Segment + Offset
  - Address: 20 bits

Protected Mode



Files

```
albert@alosvm:~/jos2016fallal/lab00osboot/lab12-helloworldc-protected$ ls -liaR
.:
total 72
1579876 drwxrwxr-x  3 albert albert  4096 12月 13 12:51 .
1578305 drwxrwxr-x 16 albert albert  4096 12月 13 12:51 ..
1597468 -rwxrwxr-x  1 albert albert 33269 11月 22  2017 bochsrc.txt
1597470 -rwxrwxr-x  1 albert albert  2879 11月 22  2017 boot.S
1586235 drwxrwxr-x  2 albert albert  4096 11月 22  2017 inc
1598749 -rw-rw-r--  1 albert albert   470 11月 22  2017 main.c
1597471 -rwxrwxr-x  1 albert albert   959 11月 22  2017 Makefile
1597474 -rw-rw-r--  1 albert albert    54 11月 22  2017 Readme.txt
1597475 -rwxrwxr-x  1 albert albert   400 11月 22  2017 sign.pl
1597512 -rw-rw-r--  1 albert albert   128 11月 22  2017 tar.sh

./inc:
total 24
1586235 drwxrwxr-x 2 albert albert 4096 11月 22  2017 .
1579876 drwxrwxr-x 3 albert albert 4096 12月 13 12:51 ..
1597472 -rwxrwxr-x 1 albert albert  676 11月 22  2017 mmu.h
1575610 -rw-rw-r-- 1 albert albert 1950 11月 22  2017 types.h
1596603 -rw-rw-r-- 1 albert albert 7071 11月 22  2017 x86.h
albert@alosvm:~/jos2016fallal/lab00osboot/lab12-helloworldc-protected$
```

inc/mmu.h

```
/*
 * Macros to build GDT entries in assembly.
 */
#define SEG_NULL                                \
    .word 0, 0;                             \
    .byte 0, 0, 0, 0
#define SEG(type,base,lim)                          \
    .word (((lim) >> 12) & 0xffff), ((base) & 0xffff);   \
    .byte (((base) >> 16) & 0xff), (0x90 | (type)),      \
        (0xC0 | (((lim) >> 28) & 0xf)), (((base) >> 24) & 0xff)

// Application segment type bits
#define STA_X       0x8     // Executable segment
#define STA_E       0x4     // Expand down (non-executable segments)
#define STA_C       0x4     // Conforming code segment (executable only)
#define STA_W       0x2     // Writeable (non-executable segments)
#define STA_R       0x2     // Readable (executable segments)
#define STA_A       0x1     // Accessed
```

inc/types.h

```c
#ifndef NULL
#define NULL ((void*) 0)
#endif

// Represents true-or-false values
typedef _Bool bool;
enum { false, true };

// Explicitly-sized versions of integer types
typedef __signed char int8_t;
typedef unsigned char uint8_t;
typedef short int16_t;
typedef unsigned short uint16_t;
typedef int int32_t;
typedef unsigned int uint32_t;
typedef long long int64_t;
typedef unsigned long long uint64_t;

// Pointers and addresses are 32 bits long.
// We use pointer types to represent virtual addresses,
// uintptr_t to represent the numerical values of virtual addresses,
// and physaddr_t to represent physical addresses.
typedef int32_t intptr_t;
typedef uint32_t uintptr_t;
typedef uint32_t physaddr_t;

// Page numbers are 32 bits long.
typedef uint32_t ppn_t;

```

inc/x86.h

```
#include <inc/types.h>

static __inline void breakpoint(void) __attribute__((always_inline));
static __inline uint8_t inb(int port) __attribute__((always_inline));
static __inline void insb(int port, void *addr, int cnt) __attribute__((always_inline));
static __inline uint16_t inw(int port) __attribute__((always_inline));
static __inline void insw(int port, void *addr, int cnt) __attribute__((always_inline));
static __inline uint32_t inl(int port) __attribute__((always_inline));
static __inline void insl(int port, void *addr, int cnt) __attribute__((always_inline));
static __inline void outb(int port, uint8_t data) __attribute__((always_inline));
static __inline void outsb(int port, const void *addr, int cnt) __attribute__((always_inline));
static __inline void outw(int port, uint16_t data) __attribute__((always_inline));
static __inline void outsw(int port, const void *addr, int cnt) __attribute__((always_inline));
static __inline void outsl(int port, const void *addr, int cnt) __attribute__((always_inline));
static __inline void outl(int port, uint32_t data) __attribute__((always_inline));
static __inline void invlpg(void *addr) __attribute__((always_inline));
static __inline void lidt(void *p) __attribute__((always_inline));
static __inline void lldt(uint16_t sel) __attribute__((always_inline));
static __inline void ltr(uint16_t sel) __attribute__((always_inline));
static __inline void lcr0(uint32_t val) __attribute__((always_inline));
static __inline uint32_t rcr0(void) __attribute__((always_inline));
static __inline uint32_t rcr2(void) __attribute__((always_inline));
static __inline void lcr3(uint32_t val) __attribute__((always_inline));
static __inline uint32_t rcr3(void) __attribute__((always_inline));
static __inline void lcr4(uint32_t val) __attribute__((always_inline));
static __inline uint32_t rcr4(void) __attribute__((always_inline));
static __inline void tlbflush(void) __attribute__((always_inline));
static __inline uint32_t read_eflags(void) __attribute__((always_inline));
static __inline void write_eflags(uint32_t eflags) __attribute__((always_inline));
static __inline uint32_t read_ebp(void) __attribute__((always_inline));
```

boot.S

```
#include <inc/mmu.h>

.set PROTECT_MODE_CSEG, 0x8         # kernel code segment selector
.set PROTECT_MODE_DSEG, 0x10        # kernel data segment selector
.set CR0_PE_ON,      0x1         # protected mode enable flag

.globl start
start:
  .code16                     # Assemble for 16-bit mode
  cli                         # Disable interrupts
  cld                         # String operations increment

  xorw    %ax,%ax             # Segment number zero
  movw    %ax,%ds             # initiate Data Segment ax->ds
  movw    %ax,%es             # Extra Segment
  movw    %ax,%ss             # Stack Segment

  movw    $0xb800,%ax       #display msg1 directly in read mode
  movw    %ax,%es
  movw    $msg1,%si         #"in real mode  "
  movw    $0xbe2,%di
  movw    $24,%cx
  rep     movsb

  movw    $hellostring,%si
  movw    $0xc04,%di
  movw    $28,%cx
  rep     movsb               # print "hello world" in real mode

```

boot.S

```
seta20.1: # to enable a20
    #read a byte from prort 0x64
  inb     $0x64,%al                # Wait 8042 keyboard for not busy
  testb   $0x2,%al
  jnz     seta20.1

  movb    $0xd1,%al                # 0xd1 -> port 0x64
  outb    %al,$0x64

seta20.2:
  inb     $0x64,%al                # Wait 8042 keyboard for not busy
  testb   $0x2,%al
  jnz     seta20.2

    #enable a20
  movb    $0xdf,%al                # 0xdf -> port 0x60
  outb    %al,$0x60

lgdtload:
  lgdt    gdtdesc
  movl    %cr0, %eax
  orl     $CR0_PE_ON, %eax
  movl    %eax, %cr0

  ljmp    $PROTECT_MODE_CSEG, $protcseg
```

boot.S (W)

```
  .code32                     # Assemble for 32-bit mode
protcseg:
  # Set up the protected-mode data segment registers
  movw    $PROTECT_MODE_DSEG, %ax
  movw    %ax, %ds                # initiate Data Segment
  movw    %ax, %es                # Extra Segment
  movw    %ax, %fs                #
  movw    %ax, %gs                #
  movw    %ax, %ss                # Stack Segment

  movl    $msg2,%esi
  movl    $0xb8d22,%edi
  movl    $62,%ecx
  rep     movsb                   #print "hello world" in protected mode

#Set up the stack pointer and call into C
  movl   $start, %esp
  call bootmain

#loop forver
spin:
  jmp spin
```

boot.S

```
.p2align 2                                  # force 4 byte alignment
gdt:
  SEG_NULL                                # null seg
  SEG(STA_X|STA_R, 0x0, 0xffffffff)       # code seg
  SEG(STA_W, 0x0, 0xffffffff)             # data seg

gdtdesc:
  .word   0x17                                # sizeof(gdt) - 1
  .long   gdt                                 # address gdt

#string to print
msg1:
  .byte 'i',0x7,'n',0x7,' ',0x7,'r',0x7,'e',0x7,'a',0x7,'l',0x7,' ',0x7,
  'm',0x7,'o',0x7,'d',0x7,'e',0x7
msg2:
  .byte 'i',0x7,'n',0x7,' ',0x7,'p',0xf,'r',0xf,'o',0xf,'t',0xf,'e',0xf,
  'c',0xf,'t',0xf,'e',0xf,'d',0xf,' ',0x7,'m',0x7,'o',0x7,'d',0x7,'e',0x7
hellostring:
  .byte ':',0xf,' ',0xc,' ',0xc,'h',0xc,'e',0xc,'l',0xc,'l',0xc,'o',0xc,
  ' ',0xc,'w',0xc,'o',0xc,'r',0xc,'l',0xc,'d',0xc
```

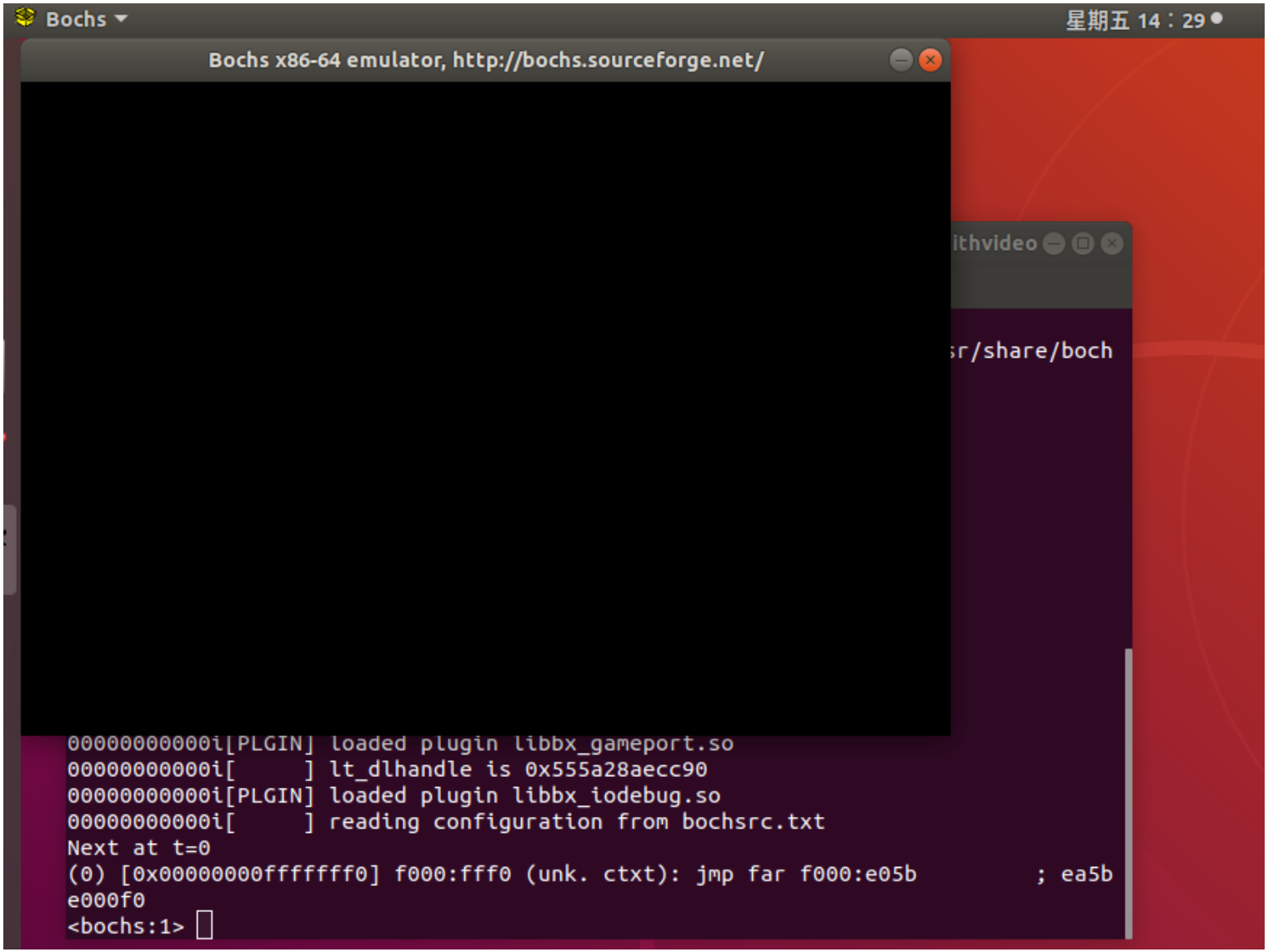
main.c (w)

```c
#include <inc/x86.h>
/*****************************************************************
 *****************************************************************/

char msg3[]={'H',0xc,'i',0xc,' ',0xc,'A',0xc,'l',0xc,'b',0xc,'e',0xc,'r',
0xc,'t',0xc,'!',0xc};
void
bootmain(void)
{
  __asm __volatile("movl %0,%%esi\n\tmovl $0xb8dea,%%edi\n\tmovl
  $20,%%ecx\n\t rep movsb"
        ::"r"(msg3));

    outw(0x8A00, 0x8A00);
    outw(0x8A00, 0x8E00);
    while (1)
        /* do nothing */;
}
```

make run

```
gcc -pipe  -nostdinc -m32 -Os -fno-builtin -I. -Wall -Wno-unused -Werror -Wno-fo
rmat -c -o boot.o boot.S
gcc -pipe  -nostdinc -m32 -Os -fno-builtin -I. -Wall -Wno-unused -Werror -Wno-fo
rmat -c -o main.o main.c
ld -m elf_i386 -N -e start -Ttext 0x7C00 -o boot.out boot.o main.o
objdump -S boot.out >boot.asm
objcopy -S -O binary boot.out boot
perl sign.pl boot
boot block is 352 bytes (max 510)
dd if=/dev/zero of=./.bochs.img~ count=10000 2>/dev/null
dd if=./boot of=./.bochs.img~ conv=notrunc 2>/dev/null
mv ./.bochs.img~ ./bochs.img
bochs -f bochsrc.txt
```

make run



Press “**c**”

albert@alosvm: ~/jos2016fallal/lab00osboot/lab12-helloworldc-protected

```
------------------------------

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
configuration file (typically
could be found.  When you are
ahead and start the simulation

You can also start bochs with

1. Restore factory default con
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state fro
6. Begin simulation
7. Quit now

Please choose one: [6]
Next at t=0
(0) [0x0000fffffff0] f000:fff
f0
<bochs:1> c
```

Bochs x86 emulator, http://bochs.sourceforge.net/

```
Plex86/Bochs VGABios (PCI) current-cvs 08 Jul 2014
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
 . http://bochs.sourceforge.net
 . http://www.nongnu.org/vgabios

NO Bochs VBE Support available!

Bochs BIOS - build: 03/17/16
$Revision: 12898 $ $Date: 2016-03-17 18:14:27 +0100 (Do, 17. Mär 2016) $
Options: apmbios pcibios pnpbios eltorito rombios32

ata0 master: Generic 1234 ATA-6 Hard-Disk (    4 MBytes)

Press F12 for boot menu.

Booting from Hard Disk...

 in real mode      :  hello world

 in protected mode:  hello world
                     Hi Albert!
```

IPS: 26.016M | NUM | CAPS | SCRL | HD:0-M

End