Cloud Operating Systems

Booting x86

Fabian Rauscher, Daniel Gruss, Andreas Kogler

2022-03-21

Real Mode

Starting in Real Mode

Starting in Real Mode

Starting in Real Mode

Starting in Real Mode

- 16 bit mode

Starting in Real Mode

- 16 bit mode
- Address space: 1 MB

Starting in Real Mode

- 16 bit mode
- Address space: 1 MB
- How is that possible?

Real Mode - Segementation

Real Mode - Segementation

Real Mode - Segementation

Real Mode - Segementation

- 16 bit segment resgisters (CS, SS, DS, ES, FS, GS)

Real Mode - Segementation

- 16 bit segment resgisters (CS, SS, DS, ES, FS, GS)
- Every memory access uses a segment register and a 16 bit offset

Real Mode - Segementation

- 16 bit segment resgisters (CS, SS, DS, ES, FS, GS)
- Every memory access uses a segment register and a 16 bit offset

$\rightarrow$ Actual address is (segment register $\ll$ 4) $+$ offset

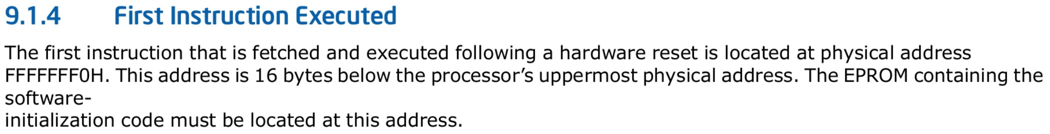
Real Mode - Segementation

- 16 bit segment resgisters (CS, SS, DS, ES, FS, GS)
- Every memory access uses a segment register and a 16 bit offset

$\rightarrow$ Actual address is (segment register $\ll$ 4) $+$ offset

- Direct physical memory access

Booting x86 Intel



9.1.4 First Instruction Executed

The first instruction that is fetched and executed following a hardware reset is located at physical address FFFFFFF0H. This address is 16 bytes below the processor’s uppermost physical address. The EPROM containing the software-
initialization code must be located at this address.

Booting in Real Mode

- Address: `0xFFFFFFF0`

Booting in Real Mode

- Address: `0xFFFFFFF0`
- How is that possible?

Booting in Real Mode

- Address: `0xFFFFFFF0`
- How is that possible?
    - CS register also has a 32-bit base address (initialized to `0xFFFF0000`)

Booting in Real Mode

- Address: `0xFFFFFFF0`
- How is that possible?
    - CS register also has a 32-bit base address (initialized to `0xFFFF0000`)
- What if I have $< 4\ GB$ RAM?

Booting in Real Mode

- Address: `0xFFFFFFF0`
- How is that possible?
    - CS register also has a 32-bit base address (initialized to `0xFFFF0000`)
- What if I have $< 4\,GB$ RAM?
    - physical address space $\neq$ RAM directly mapped

BIOS

BIOS

BIOS

BIOS

- BIOS initializes hardware platform

BIOS

- BIOS initializes hardware platform
- Select a device to boot from

BIOS

- BIOS initializes hardware platform
- Select a device to boot from
- Load MBR from device into memory (`0x7C00`)

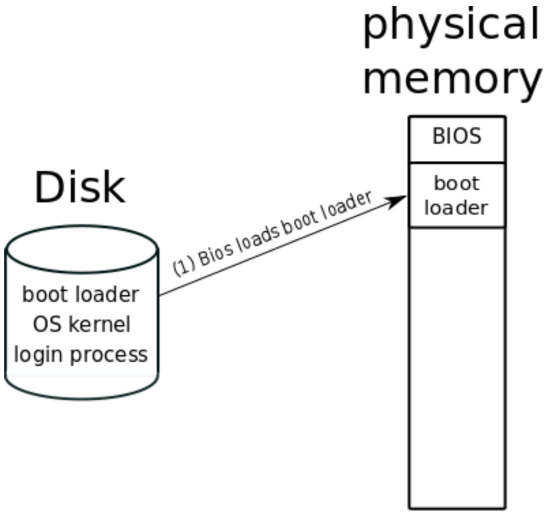
BIOS

- BIOS initializes hardware platform
- Select a device to boot from
- Load MBR from device into memory (`0x7C00`)
- Execute code from the MBR

BIOS

- BIOS initializes hardware platform
- Select a device to boot from
- Load MBR from device into memory (`0x7C00`)
- Execute code from the MBR
- MBR code usually loads more data from the disk into memory

Booting x86 Intel (Illustration)



Real Mode - Hardware Interaction

- How can we interact with the hardware?

Real Mode - Hardware Interaction

- How can we interact with the hardware?
  - Where do we get the memory layout from?

Real Mode - Hardware Interaction

- How can we interact with the hardware?
  - Where do we get the memory layout from?
  - How can we access the disk?

Real Mode - Hardware Interaction

- How can we interact with the hardware?
  - Where do we get the memory layout from?
  - How can we access the disk?
  - How can we configure the video output?

Real Mode - Hardware Interaction

- How can we interact with the hardware?
    - Where do we get the memory layout from?
    - How can we access the disk?
    - How can we configure the video output?
    - ...

Real Mode - Hardware Interaction

- How can we interact with the hardware?
    - Where do we get the memory layout from?
    - How can we access the disk?
    - How can we configure the video output?
    - ...

$\rightarrow$ BIOS calls!

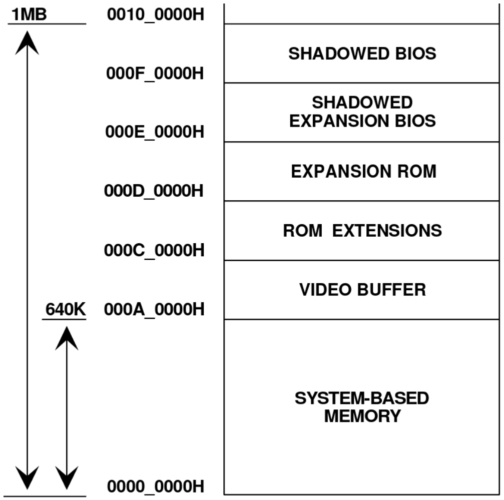
Real Mode - Hardware Interaction

- How can we interact with the hardware?
  - Where do we get the memory layout from?
  - How can we access the disk?
  - How can we configure the video output?
  - ...

→ BIOS calls!

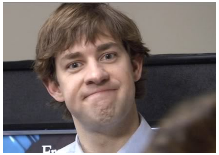
- We can trigger standardized software interrupts and let the BIOS handle it!

Real Mode - Memory Layout

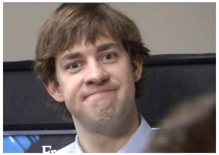


Real Mode - A20

Real Mode - A20

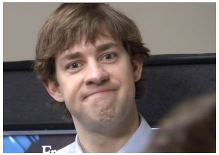


Real Mode - A20

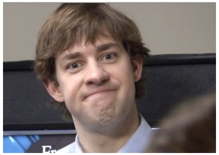


Real Mode - A20

- Memory accesses above 1MB wrap around

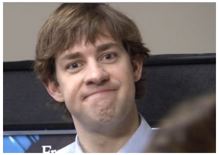


Real Mode - A20



- Memory accesses above 1MB wrap around
- Used to fix software compatability issues

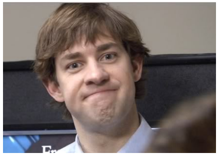
Real Mode - A20



- Memory accesses above 1MB wrap around
- Used to fix software compatability issues

→ Memory line 20 (A20) needs to be enabled by software to disable this "feature"

Real Mode - A20

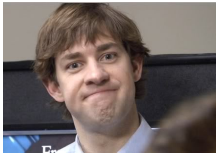


- Memory accesses above 1MB wrap around
- Used to fix software compatability issues

→ Memory line 20 (A20) needs to be enabled by software to disable this "feature"

- Multiple ways of doing this:

Real Mode - A20

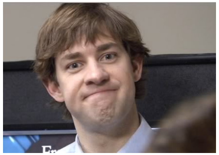


- Memory accesses above 1MB wrap around
- Used to fix software compatability issues

→ Memory line 20 (A20) needs to be enabled by software to disable this "feature"

- Multiple ways of doing this:
  - Keyboard controller

Real Mode - A20

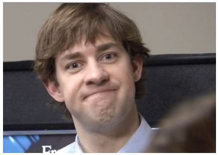


- Memory accesses above 1MB wrap around
- Used to fix software compatability issues

→ Memory line 20 (A20) needs to be enabled by software to disable this "feature"

- Multiple ways of doing this:
  - Keyboard controller
  - Fast A20 gate

Real Mode - A20

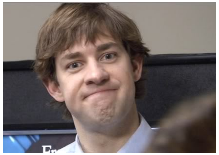


- Memory accesses above 1MB wrap around
- Used to fix software compatability issues

→ Memory line 20 (A20) needs to be enabled by software to disable this "feature"

- Multiple ways of doing this:
  - Keyboard controller
  - Fast A20 gate
  - BIOS call

Real Mode - A20



- Memory accesses above 1MB wrap around
- Used to fix software compatability issues

→ Memory line 20 (A20) needs to be enabled by software to disable this "feature"

- Multiple ways of doing this:
  - Keyboard controller
  - Fast A20 gate
  - BIOS call
  - ...

Real Mode $\rightarrow$ Protected Mode

- Disable interrupts

Real Mode → Protected Mode

- Disable interrupts
- Setup Global Descriptor Table (GDT)

Real Mode → Protected Mode

- Disable interrupts
- Setup Global Descriptor Table (GDT)
- Load GDT

Real Mode $\rightarrow$ Protected Mode

- Disable interrupts
- Setup Global Descriptor Table (GDT)
- Load GDT
- Enable protected mode by setting bit 0 in CR0

Real Mode → Protected Mode

- Disable interrupts
- Setup Global Descriptor Table (GDT)
- Load GDT
- Enable protected mode by setting bit 0 in CR0
- Long jump to set CS

Real Mode → Protected Mode

- Disable interrupts
- Setup Global Descriptor Table (GDT)
- Load GDT
- Enable protected mode by setting bit 0 in CR0
- Long jump to set CS
- Load DS, ES, FS, GS, SS, ESP

Protected Mode

Protected Mode

Protected Mode

Protected Mode

Protected Mode

- 32 bit mode

Protected Mode

- 32 bit mode
- More 32 bit registers

Protected Mode

- 32 bit mode
- More 32 bit registers
- Access to up to 4GB of memory

Protected Mode

- 32 bit mode
- More 32 bit registers
- Access to up to 4GB of memory
- Segmentation uses the GDT

Protected Mode

- 32 bit mode
- More 32 bit registers
- Access to up to 4GB of memory
- Segmentation uses the GDT
    - It is possible to address the whole address space without switching segments

Protected Mode

- 32 bit mode
- More 32 bit registers
- Access to up to 4GB of memory
- Segmentation uses the GDT
  - It is possible to address the whole address space without switching segments

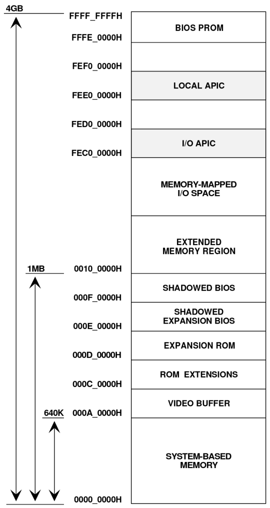
  → Segmentation more optional!

Protected Mode

- 32 bit mode
- More 32 bit registers
- Access to up to 4GB of memory
- Segmentation uses the GDT
  - It is possible to address the whole address space without switching segments

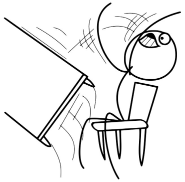
  → Segmentation more optional!
- Optional paging

Protected Mode - Memory Layout

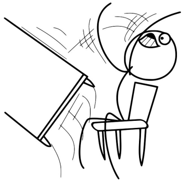


Global Descriptor Table (GDT)

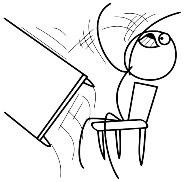
Global Descriptor Table (GDT)



Global Descriptor Table (GDT)

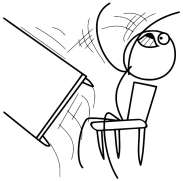


Global Descriptor Table (GDT)



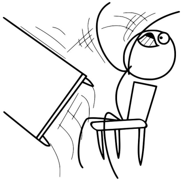
- Segment registers are offsets into the GDT

Global Descriptor Table (GDT)



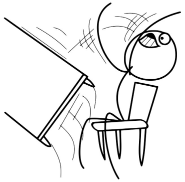
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...

Global Descriptor Table (GDT)



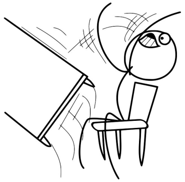
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
    - Base address

Global Descriptor Table (GDT)



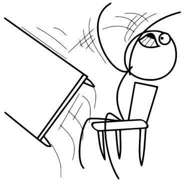
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
    - Base address
    - Limit

Global Descriptor Table (GDT)



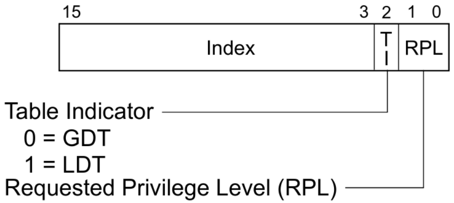
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
  - Base address
  - Limit
  - Access rights

Global Descriptor Table (GDT)

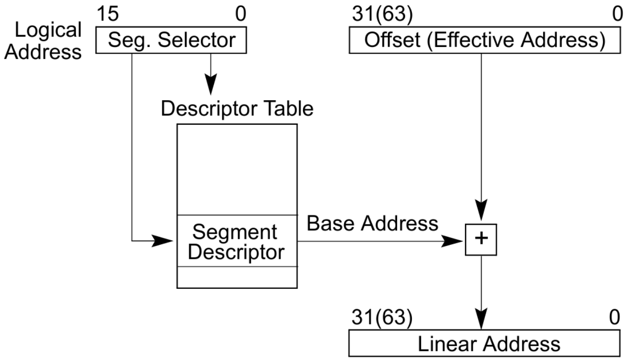


- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
    - Base address
    - Limit
    - Access rights
    - Flags

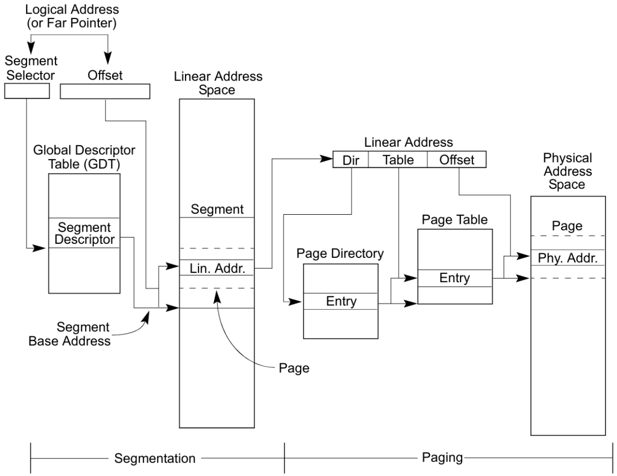
Segment Register



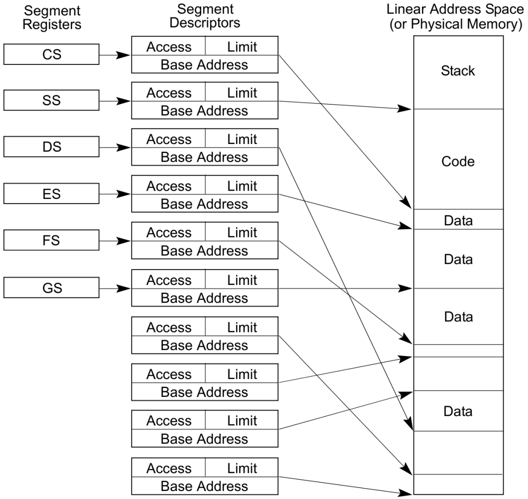
Segmentation Inner Workings



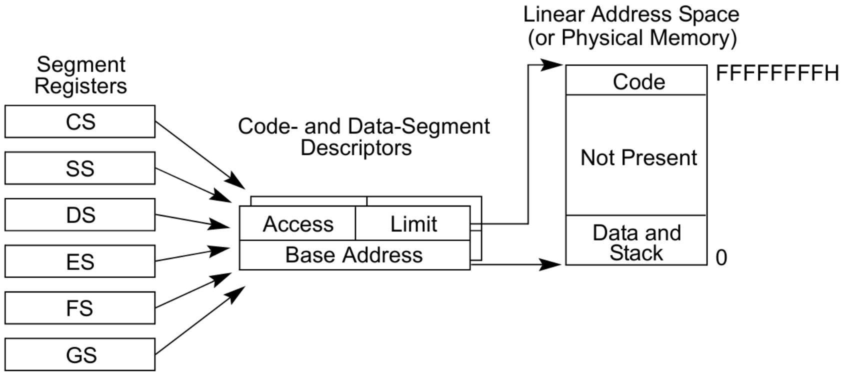
Segmentation with Paging



Segmentation Dream



Segmentation Reality



Protected Mode → Long Mode

- Disable interrupts

Protected Mode → Long Mode

- Disable interrupts
- Setup 64 bit segmend descriptors

Protected Mode → Long Mode

- Disable interrupts
- Setup 64 bit segmend descriptors
- Setup paging structure and set CR3

Protected Mode → Long Mode

- Disable interrupts
- Setup 64 bit segmend descriptors
- Setup paging structure and set CR3
- Enable long Mode via bit 8 of the EFER MSR

Protected Mode $\rightarrow$ Long Mode

- Disable interrupts
- Setup 64 bit segmend descriptors
- Setup paging structure and set CR3
- Enable long Mode via bit 8 of the EFER MSR
- Enable PAE via bit 5 of CR4

Protected Mode → Long Mode

- Disable interrupts
- Setup 64 bit segmend descriptors
- Setup paging structure and set CR3
- Enable long Mode via bit 8 of the EFER MSR
- Enable PAE via bit 5 of CR4
- Enable paging via bit 31 in CR0

Protected Mode → Long Mode

- Disable interrupts
- Setup 64 bit segmend descriptors
- Setup paging structure and set CR3
- Enable long Mode via bit 8 of the EFER MSR
- Enable PAE via bit 5 of CR4
- Enable paging via bit 31 in CR0
- Reload segment selectors to enter long mode proper

Long Mode

Long Mode

Long Mode

Long Mode

Long Mode

- Only supported with paging enabled

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers
  → RAX, RBX, RCX RDX, ..., R8-R15

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers
  $\rightarrow$ RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers
  → RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers
  - $\rightarrow$ RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)
- Segmentation is simplified

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers
  $\rightarrow$ RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)
- Segmentation is simplified
  - All segment limits are ignored

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers
  - → RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)
- Segmentation is simplified
  - All segment limits are ignored
  - CS, SS, ES, DS have a base of 0

Long Mode

- Only supported with paging enabled
- Register extensions and new 64 bit registers

  → RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)
- Segmentation is simplified
  - All segment limits are ignored
  - CS, SS, ES, DS have a base of 0
  - Bases for FS and GS can be set via MSRs

Booting SWEB

GRUB

GRUB

GRUB

GRUB

- Grand Unified Bootloader (GRUB)

GRUB

- Grand Unified Bootloader (GRUB)
- Runs in real mode and protected mode

GRUB

- Grand Unified Bootloader (GRUB)
- Runs in real mode and protected mode

  $\rightarrow$ Protected mode: most of the time

GRUB

- Grand Unified Bootloader (GRUB)
- Runs in real mode and protected mode
  - → Protected mode: most of the time
  - → Real mode: entry and BIOS calls

GRUB

- Grand Unified Bootloader (GRUB)
- Runs in real mode and protected mode
  - → Protected mode: most of the time
  - → Real mode: entry and BIOS calls
- Basic hardware detection and framebuffer setup

GRUB

- Grand Unified Bootloader (GRUB)
- Runs in real mode and protected mode
  - → Protected mode: most of the time
  - → Real mode: entry and BIOS calls
- Basic hardware detection and framebuffer setup
- Loads SWEB into memory

GRUB

- Grand Unified Bootloader (GRUB)
- Runs in real mode and protected mode
  - → Protected mode: most of the time
  - → Real mode: entry and BIOS calls
- Basic hardware detection and framebuffer setup
- Loads SWEB into memory
- Provides hardware information according to the Multiboot spec

GRUB

- Grand Unified Bootloader (GRUB)
- Runs in real mode and protected mode
  - → Protected mode: most of the time
  - → Real mode: entry and BIOS calls
- Basic hardware detection and framebuffer setup
- Loads SWEB into memory
- Provides hardware information according to the Multiboot spec
- Runs SWEB

Multiboot

- Provides standardized format for passing information from the Bootloader to the OS

Multiboot

- Provides standardized format for passing information from the Bootloader to the OS
    - Memory areas and their type

Multiboot

- Provides standardized format for passing information from the Bootloader to the OS
  - Memory areas and their type
  - Location of the framebuffer and the configured video mode

Multiboot

- Provides standardized format for passing information from the Bootloader to the OS
  - Memory areas and their type
  - Location of the framebuffer and the configured video mode
  - Drive infos

Multiboot

- Provides standardized format for passing information from the Bootloader to the OS
  - Memory areas and their type
  - Location of the framebuffer and the configured video mode
  - Drive infos
  - ...

Multiboot

- Provides standardized format for passing information from the Bootloader to the OS
    - Memory areas and their type
    - Location of the framebuffer and the configured video mode
    - Drive infos
    - ...
- Pointer to the multiboot header passed via EBX to the kernel entry