Cloud Operating Systems

Booting x86

Fabian Rauscher, Daniel Gruss, Andreas Kogler

2023-03-20

Real Mode

Starting in Real Mode

Starting in Real Mode

Fabian Rauscher, Daniel Gruss, Andreas Kogler

Starting in Real Mode

Fabian Rauscher, Daniel Gruss, Andreas Kogler

Starting in Real Mode

- 16 bit mode

Fabian Rauscher, Daniel Gruss, Andreas Kogler

Starting in Real Mode

- 16 bit mode
- Address space: 1 MB

Fabian Rauscher, Daniel Gruss, Andreas Kogler

Starting in Real Mode

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

- 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

- 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

  - 0x3000 segment, 0xd463 offset = 0x3d463

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

- 0x3000 segment, 0xd463 offset = 0x3d463
- 0x1004 segment, 0x11c8 offset = 0x128c8

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

- 0x3000 segment, 0xd463 offset = 0x3d463
- 0x1004 segment, 0x11c8 offset = 0x128c8
- 0x345 segment, 0xf478 offset = 0x128c8

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

- 0x3000 segment, 0xd463 offset = 0x3d463
- 0x1004 segment, 0x11c8 offset = 0x128c8
- 0x345 segment, 0xf478 offset = 0x128c8
- 0xff13 segment, 0xfff0 offset = 0x10f120 = 0xf120

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

  - 0x3000 segment, 0xd463 offset = 0x3d463
  - 0x1004 segment, 0x11c8 offset = 0x128c8
  - 0x345 segment, 0xf478 offset = 0x128c8
  - 0xff13 segment, 0xfff0 offset = 0x10f120 = 0xf120

- Direct physical memory access

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.

- Address: `0xFFFFFFF0`

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

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

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

- Address: `0xFFFFFFF0`
- How is this 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

- BIOS initializes hardware platform

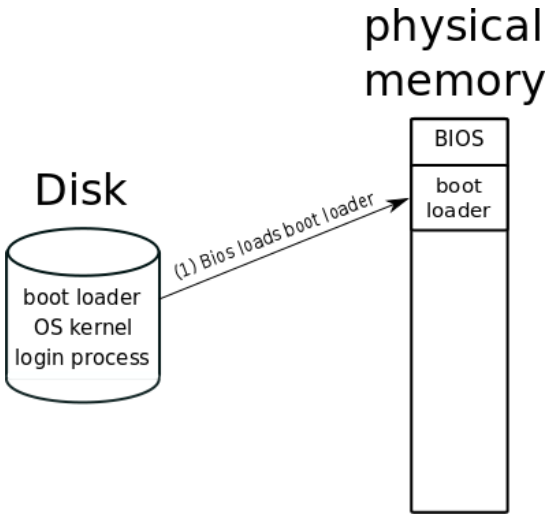
- BIOS initializes hardware platform
- Select a device to boot from

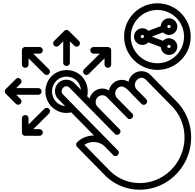
- BIOS initializes hardware platform
- Select a device to boot from
- Load MBR from device into memory (`0x7C00`)

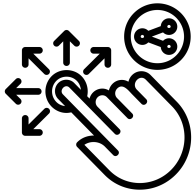
- BIOS initializes hardware platform
- Select a device to boot from
- Load MBR from device into memory (`0x7C00`)
- Execute code from the MBR

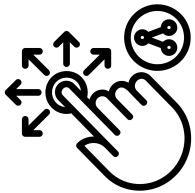
- BIOS initializes hardware platform
- Select a device to boot from
- Load MBR from device into memory (`0x7C00`)
- Execute code from the MBR
- MBR code loads more data from the disk into memory

Booting x86 Intel (Illustration)

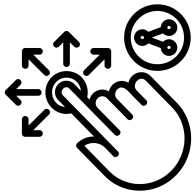




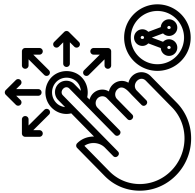




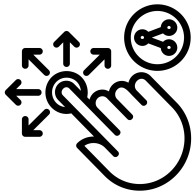
- How can we interact with the hardware?



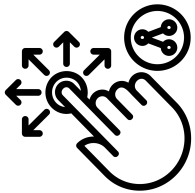
- How can we interact with the hardware?
  - Where do we get the memory layout from?



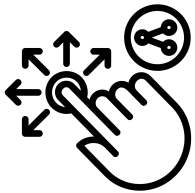
- How can we interact with the hardware?
  - Where do we get the memory layout from?
  - How can we access the disk?



- 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?

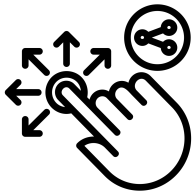


- How can we interact with the hardware?
  - Where do we get the memory layout from?
  - How can we access the disk?
  - How can we configure the video output?
  - ...



- 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!

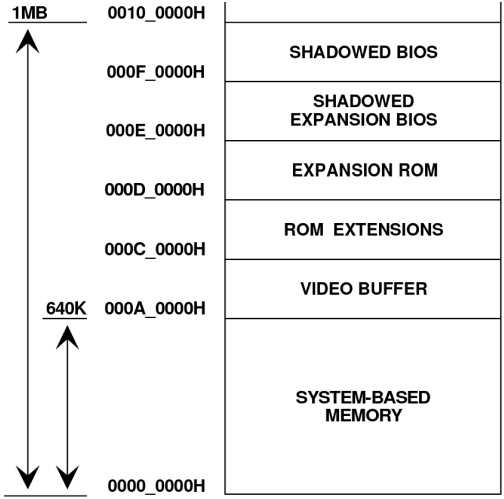


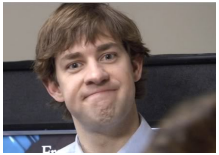
- How can we interact with the hardware?
  - Where do we get the memory layout from?
  - How can we access the disk?
  - How can we configure the video output?
  - ...

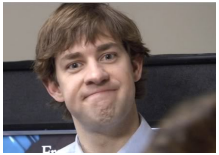
→ BIOS calls!

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

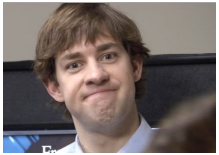
Real Mode - Memory Layout



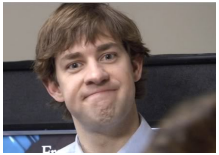


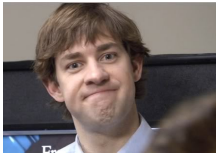


- Memory accesses above 1 MB wrap around



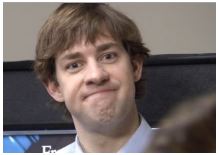
- Memory accesses above 1 MB wrap around
- Done to fix software compatability issues





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

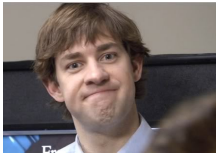
→ Memory line 20 (A20) needs to be enabled by software to disable this "feature"



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

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

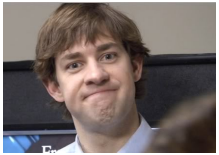
- 65520 Bytes of extra memory!



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

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

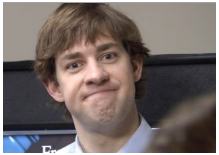
- 65520 Bytes of extra memory!
- Multiple ways of doing this:



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

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

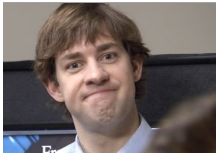
- 65520 Bytes of extra memory!
- Multiple ways of doing this:
  - Keyboard controller



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

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

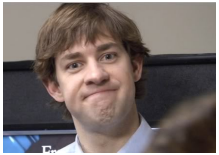
- 65520 Bytes of extra memory!
- Multiple ways of doing this:
  - Keyboard controller
  - Fast A20 gate



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

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

- 65520 Bytes of extra memory!
- Multiple ways of doing this:
  - Keyboard controller
  - Fast A20 gate
  - BIOS call



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

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

- 65520 Bytes of extra memory!
- Multiple ways of doing this:
  - Keyboard controller
  - Fast A20 gate
  - BIOS call
  - ...

- Disable interrupts

- Disable interrupts
- Setup Global Descriptor Table (GDT)

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

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

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

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

Protected Mode

- 32 bit mode

- 32 bit mode
- More 32 bit registers

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

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

- 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

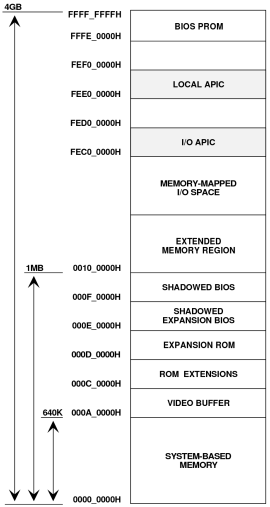
- 32 bit mode
- More 32 bit registers
- Access to up to 4GB of memory
- Segmentation uses the GDT
  - It is possible to address the whole address space without switching segments

  $\rightarrow$ Segmentation more optional!

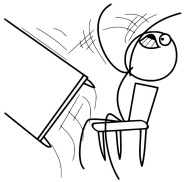
- 32 bit mode
- More 32 bit registers
- Access to up to 4GB of memory
- Segmentation uses the GDT
  - It is possible to address the whole address space without switching segments

  $\rightarrow$ Segmentation more optional!
- Optional paging

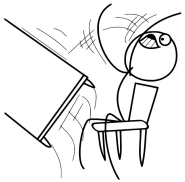
Protected Mode - Memory Layout

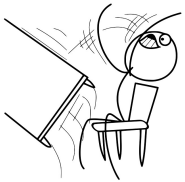


Global Descriptor Table (GDT)



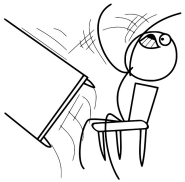
Global Descriptor Table (GDT)





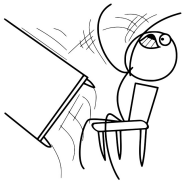
- Segment registers are offsets into the GDT

Global Descriptor Table (GDT)



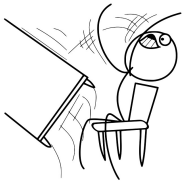
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...

Global Descriptor Table (GDT)



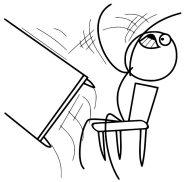
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
  - Base address

Global Descriptor Table (GDT)



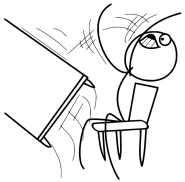
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
  - Base address
  - Limit

Global Descriptor Table (GDT)



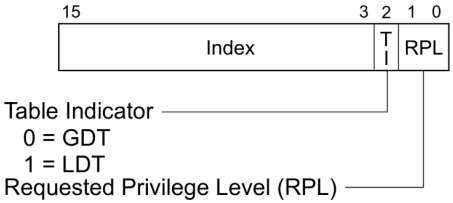
- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
  - Base address
  - Limit
  - Access rights

Global Descriptor Table (GDT)

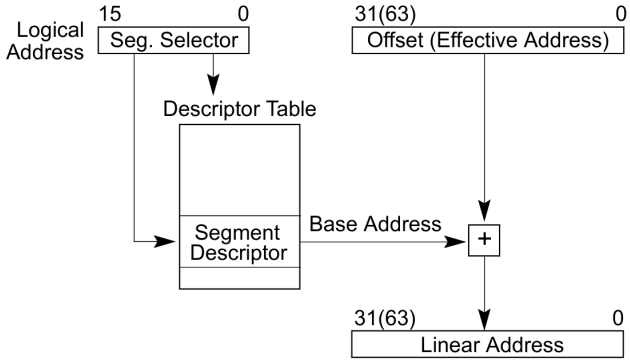


- Segment registers are offsets into the GDT
- GDT is an array of segment descriptors that each hold ...
  - Base address
  - Limit
  - Access rights
  - Flags

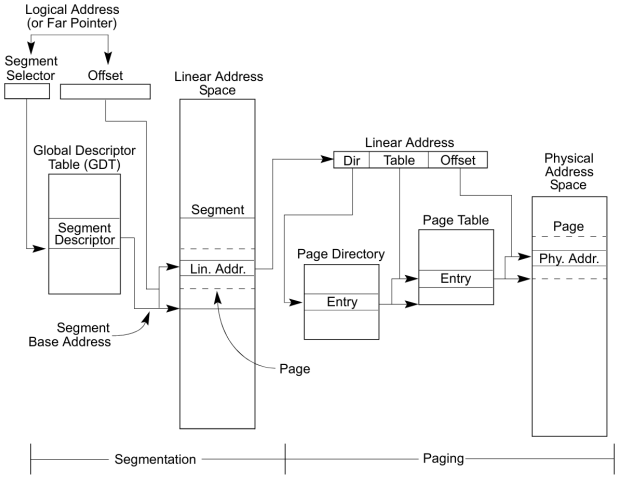
Segment Register



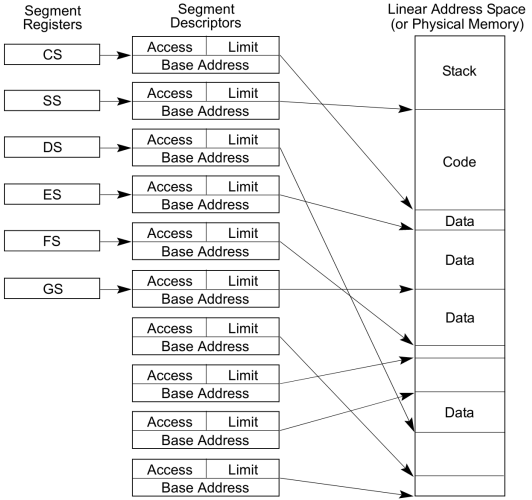
Segmentation Inner Workings

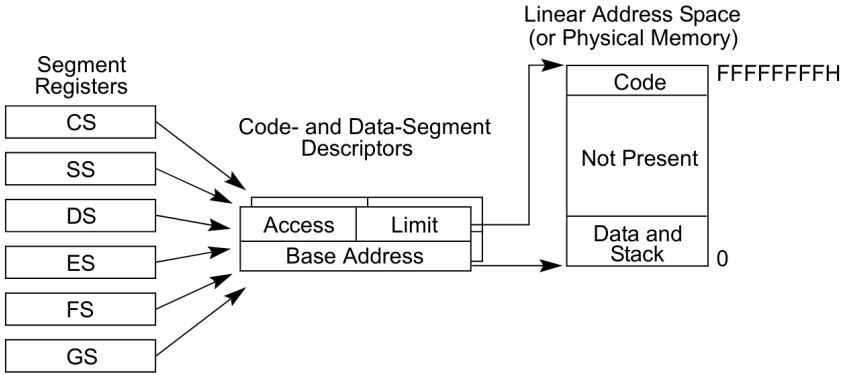


Segmentation with Paging

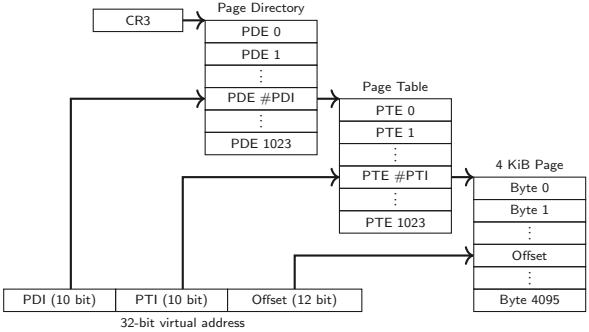


Segmentation Dream




Linear Address Space
(or Physical Memory)
Segment
Registers
CS
SS
DS
ES
FS
GS
Code- and Data-Segment
Descriptors
Access
Limit
Base Address
Code
FFFFFFFFH
Not Present
Data and
Stack
0

Paging: x86-32 with page size 4 KiB



Who could ever need more than 4GB of RAM?

SYSTEM REQUIREMENTS

MINIMUM:	RECOMMENDED:
Requires a 64-bit processor and operating system	Requires a 64-bit processor and operating system
OS: Windows 10 (64 bit)	OS: Windows 10 (64 bit)/Windows 11 (64 bit)
Processor: AMD Ryzen 3 1200 / Intel Core i5-7500	Processor: AMD Ryzen 5 3600 / Intel Core i7 8700
Memory: 8 GB RAM	Memory: 16 GB RAM
Graphics: AMD Radeon RX 560 with 4GB VRAM / NVIDIA GeForce GTX 1050 Ti with 4GB VRAM	Graphics: AMD Radeon RX 5700 / NVIDIA GeForce GTX 1070
DirectX: Version 12	DirectX: Version 12
Network: Broadband Internet connection	Network: Broadband Internet connection
Additional Notes: Estimated performance (when set to Prioritize Performance): 1080p/45fps. • Framerate might drop in graphics-intensive scenes. • AMD Radeon RX 6700 XT or NVIDIA GeForce RTX 2060 required to support ray tracing.	Additional Notes: Estimated performance: 1080p/60fps • Framerate might drop in graphics-intensive scenes. • AMD Radeon RX 6700 XT or NVIDIA GeForce RTX 2070 required to support ray tracing.

SYSTEM REQUIREMENTS

MINIMUM:

Requires a 64-bit processor and operating system

OS: Windows 10 64-bit (Version 1909) | For Ray Tracing or VR: Windows 10 64-bit (Version 2004)

Processor: Intel Core i3-2130 or AMD FX 4300 |For VR: Intel Core i5-9600k or AMD Ryzen 5 2600X

Memory: 8 GB RAM

Graphics: NVIDIA GTX 1050 Ti or AMD RX 470 | For Ray Tracing: GeForce RTX 2060 or Radeon RX 6700 XT | For VR: NVIDIA GTX 1660 Ti or AMD RX 590

DirectX: Version 12

Network: Broadband Internet connection

Storage: 80 GB available space

Sound Card: DirectX Compatible

VR Support: SteamVR. Keyboard and mouse required

RECOMMENDED:

Requires a 64-bit processor and operating system

OS: Windows 10 64-bit (Version 1909) | For Ray Tracing or VR: Windows 10 64-bit (Version 2004)

Processor: Intel Core i5 9600K or AMD Ryzen 5 2600X

Memory: 16 GB RAM

Graphics: NVIDIA GTX 1660 Ti or AMD RX 590 | For Ray Tracing: GeForce RTX 3070 or Radeon RX 6800 | For VR: NVIDIA RTX 2070 or AMD RX 6700 XT

DirectX: Version 12

Network: Broadband Internet connection

Storage: 80 GB available space

Sound Card: DirectX Compatible

SYSTEM REQUIREMENTS

MINIMUM:	RECOMMENDED:
Requires a 64-bit processor and operating system	Requires a 64-bit processor and operating system
OS: Windows 10 - April 2018 Update (v1803)	OS: Windows 10 - April 2018 Update (v1803)
Processor: Intel® Core™ i7-4770K / AMD Ryzen 5 1500X	Processor: Intel® Core™ i7-4770K / AMD Ryzen 5 1500X
Memory: 12 GB RAM	Memory: 12 GB RAM
Graphics: Nvidia GeForce GTX 1060 6GB / AMD Radeon RX 480 4GB	Graphics: Nvidia GeForce GTX 1060 6GB / AMD Radeon RX 480 4GB
Network: Broadband Internet connection	Network: Broadband Internet connection
Storage: 150 GB available space	Storage: 150 GB available space
Sound Card: Direct X Compatible	Sound Card: Direct X Compatible

SYSTEM REQUIREMENTS

MINIMUM:	RECOMMENDED:
Requires a 64-bit processor and operating system	Requires a 64-bit processor and operating system
OS: Windows 10	OS: Windows 10/11
Processor: INTEL CORE I5-8400 or AMD RYZEN 3 3300X	Processor: INTEL CORE I7-8700K or AMD RYZEN 5 3600X
Memory: 12 GB RAM	Memory: 16 GB RAM
Graphics: NVIDIA GEFORCE GTX 1060 3 GB or AMD RADEON RX 580 4 GB	Graphics: NVIDIA GEFORCE GTX 1070 8 GB or AMD RADEON RX VEGA 56 8 GB
DirectX: Version 12	DirectX: Version 12
Storage: 60 GB available space	Storage: 60 GB available space
Sound Card: Windows Compatible Audio Device	Sound Card: Windows Compatible Audio Device
Additional Notes:	Additional Notes:

Physical Address Extension (PAE)

Physical Address Extension (PAE)

Physical Address Extension (PAE)

- 4 GB of physical address space is not enough

Physical Address Extension (PAE)

- 4 GB of physical address space is not enough
- Increase of page table entry sizes from 4 to 8 Bytes

Physical Address Extension (PAE)

- 4 GB of physical address space is not enough
- Increase of page table entry sizes from 4 to 8 Bytes
- More address bits in page table entries

Physical Address Extension (PAE)

- 4 GB of physical address space is not enough
- Increase of page table entry sizes from 4 to 8 Bytes
- More address bits in page table entries

→ Access to a bigger physical address space with paging

Physical Address Extension (PAE)

- 4 GB of physical address space is not enough
- Increase of page table entry sizes from 4 to 8 Bytes
- More address bits in page table entries

→ Access to a bigger physical address space with paging

- Has no effect when paging is disabled

Physical Address Extension (PAE)

- 4 GB of physical address space is not enough
- Increase of page table entry sizes from 4 to 8 Bytes
- More address bits in page table entries

→ Access to a bigger physical address space with paging

- Has no effect when paging is disabled
- Can be enabled with bit 5 in CR4

32 bit PAE paging

32 bit PAE paging

32 bit PAE paging

- Adds PDPT to account for larger entry sizes

32 bit PAE paging

- Adds PDPT to account for larger entry sizes
- Linear address is still 32 bit

32 bit PAE paging

- Adds PDPT to account for larger entry sizes
- Linear address is still 32 bit

$\rightarrow$ Virtual address space is still 4 GB

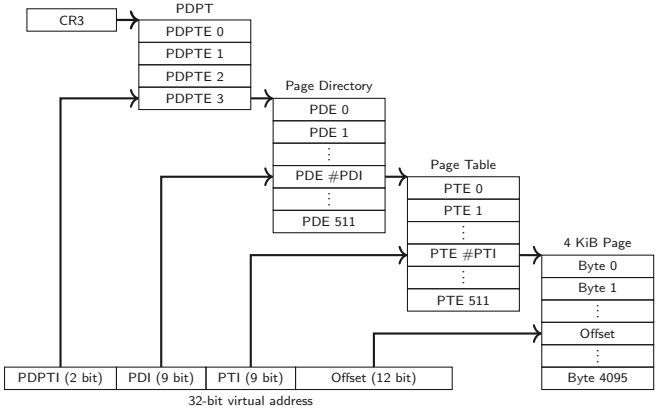
32 bit PAE paging

- Adds PDPT to account for larger entry sizes
- Linear address is still 32 bit

→ Virtual address space is still 4 GB

but physical pages above 4 GB can be mapped

Paging: x86-32-PAE with page size 4 KiB



- Disable interrupts

- Disable interrupts
- Setup 64 bit segmend descriptors

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

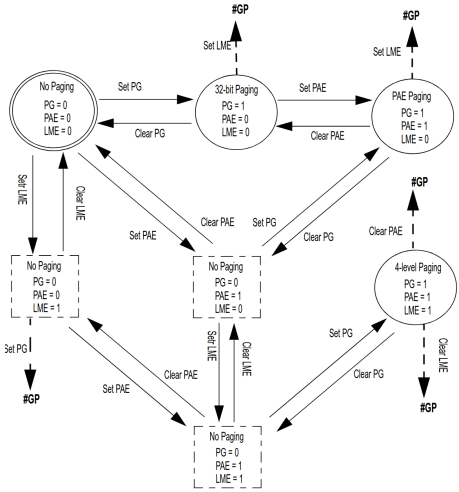
- Disable interrupts
- Setup 64 bit segmend descriptors
- Setup paging structure and set CR3
- Enable long mode via bit 8 of the EFER MSR

- 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

- 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

- 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

Paging Transitions



Long Mode

- Only supported with paging enabled

- 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

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

- 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
  - → 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
  - → RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)
- Segmentation is simplified
  - Code and data segment limits are ignored

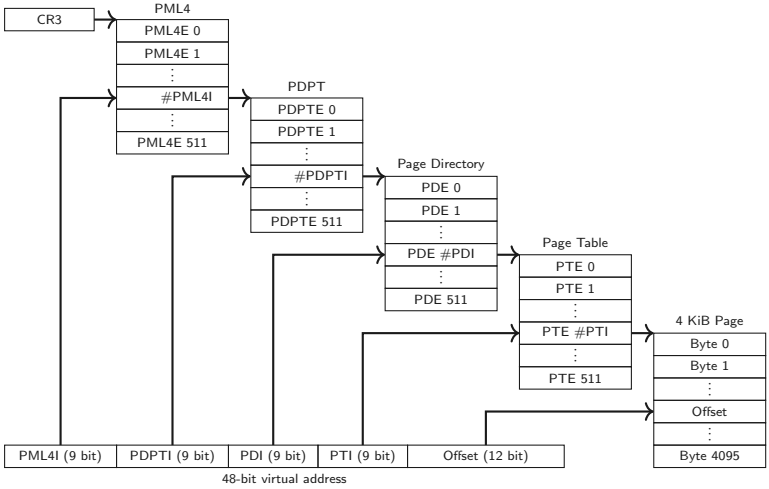
Long Mode

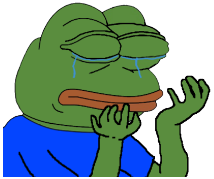
- Only supported with paging enabled
- Register extensions and new 64 bit registers
  - → RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)
- Segmentation is simplified
  - Code and data segment limits are ignored
  - CS, SS, ES, DS have a base of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

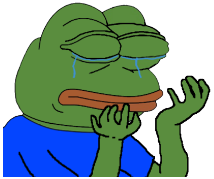
Long Mode

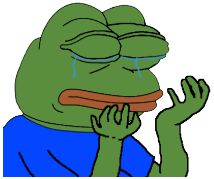
- Only supported with paging enabled
- Register extensions and new 64 bit registers
  - → RAX, RBX, RCX RDX, ..., R8-R15
- 64 bit pointers
- 48 bit virtual addresses (57 bit with 5-level paging)
- Segmentation is simplified
  - Code and data segment limits are ignored
  - CS, SS, ES, DS have a base of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

Paging: x86-64 with page size 4 KiB



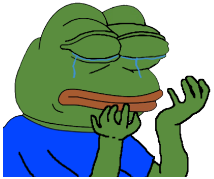




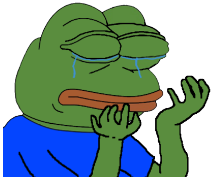


- Most bases and limits are ignored

Long Mode Segmentation

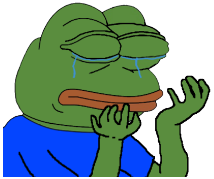


- Most bases and limits are ignored
- Why do we still need it?



- Most bases and limits are ignored
- Why do we still need it?

→ CS holds the current execution privilege level and execution mode



- Most bases and limits are ignored
- Why do we still need it?

→ CS holds the current execution privilege level and execution mode

→ task state segment

Task State Segments (TSS)

- Can be used for task switches in protected mode

- Can be used for task switches in protected mode
- Mostly a list of stack addresses in long mode

Task State Segments (TSS)

- Can be used for task switches in protected mode
- Mostly a list of stack addresses in long mode
- Holds RSP0, RSP1, RSP2, ISTs and an IO bitmap

- Can be used for task switches in protected mode
- Mostly a list of stack addresses in long mode
- Holds RSP0, RSP1, RSP2, ISTs and an IO bitmap
  - RSPx: RSP is set to RSPx when an interrupt causes a privilege change from a ring $<$ x to x

- Can be used for task switches in protected mode
- Mostly a list of stack addresses in long mode
- Holds RSP0, RSP1, RSP2, ISTs and an IO bitmap
  - RSPx: RSP is set to RSPx when an interrupt causes a privilege change from a ring $< x$ to x
  - ISTs: can be used to force a stack switch on an interrupt to a specific address

- Can be used for task switches in protected mode
- Mostly a list of stack addresses in long mode
- Holds RSP0, RSP1, RSP2, ISTs and an IO bitmap
  - RSPx: RSP is set to RSPx when an interrupt causes a privilege change from a ring $< x$ to x
  - ISTs: can be used to force a stack switch on an interrupt to a specific address
  - IO bitmap: manages IO port permissions for ring 3

Interrupt Descriptor Table

- Tells the CPU what to do in case of an interrupt

- Tells the CPU what to do in case of an interrupt
- Linear address and size stored in the IDTR

- Tells the CPU what to do in case of an interrupt
- Linear address and size stored in the IDTR
  (loaded with the lidt instruction)

Interrupt Descriptor Table

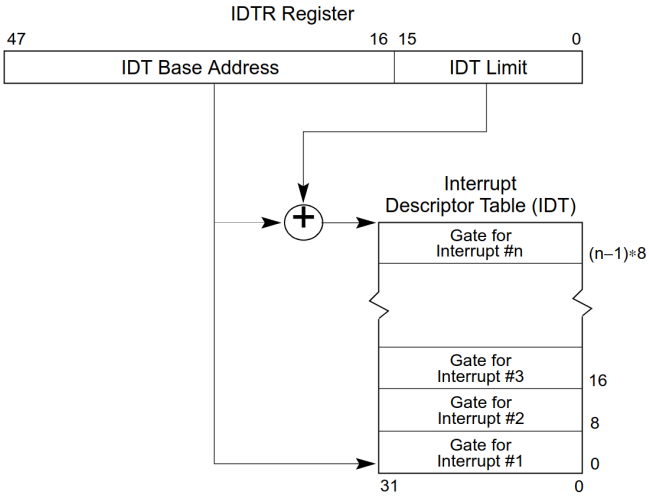
- Tells the CPU what to do in case of an interrupt
- Linear address and size stored in the IDTR
  (loaded with the lidt instruction)
- Up to 256 entries (vectors)

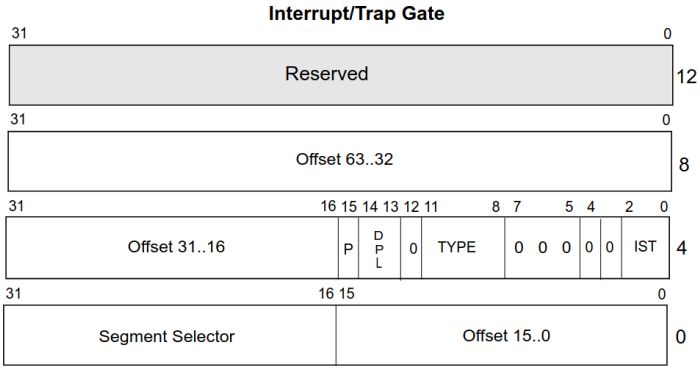
- Tells the CPU what to do in case of an interrupt
- Linear address and size stored in the IDTR
  (loaded with the lidt instruction)
- Up to 256 entries (vectors)
- Two types of entries:

- Tells the CPU what to do in case of an interrupt
- Linear address and size stored in the IDTR
  (loaded with the lidt instruction)
- Up to 256 entries (vectors)
- Two types of entries:
  - Interrupt gates: disable interrupts

- Tells the CPU what to do in case of an interrupt
- Linear address and size stored in the IDTR
  (loaded with the lidt instruction)
- Up to 256 entries (vectors)
- Two types of entries:
  - Interrupt gates: disable interrupts
  - Trap gates: don't disable interrupts

- Tells the CPU what to do in case of an interrupt
- Linear address and size stored in the IDTR
  (loaded with the lidt instruction)
- Up to 256 entries (vectors)
- Two types of entries:
  - Interrupt gates: disable interrupts
  - Trap gates: don't disable interrupts
- Vector numbers 0-31 are reserved


IDTR Register
47
16
15
0
IDT Base Address
IDT Limit
Interrupt
Descriptor Table (IDT)
Gate for
Interrupt #n
(n–1)*8
Gate for
Interrupt #3
16
Gate for
Interrupt #2
8
Gate for
Interrupt #1
0
31
0



DPL	Descriptor Privilege Level
Offset	Offset to procedure entry point
P	Segment Present flag
Selector	Segment Selector for destination code segment
IST	Interrupt Stack Table

- **not** the same as C++ or Java exceptions

- **not** the same as C++ or Java exceptions
- "Exceptions occur when the processor detects an error condition while executing an instruction [...]"

- **not** the same as C++ or Java exceptions
- "Exceptions occur when the processor detects an error condition while executing an instruction [...]"
- Located in vector numbers 0-31

- **not** the same as C++ or Java exceptions
- "Exceptions occur when the processor detects an error condition while executing an instruction [...]"
- Located in vector numbers 0-31

→ Other interrupts have to be remapped to not use this range

- **not** the same as C++ or Java exceptions
- "Exceptions occur when the processor detects an error condition while executing an instruction [...]"
- Located in vector numbers 0-31

→ Other interrupts have to be remapped to not use this range

- Some exceptions push an error code

Table 6-1. Protected-Mode Exceptions and Interrupts

Vector	Mnemonic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD instruction or reserved opcode.
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[1]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.

Table 6-1. Protected-Mode Exceptions and Interrupts (Contd.)

Vector	Mnemonic	Description	Type	Error Code	Source
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[2]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[3]
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions[4]
20	#VE	Virtualization Exception	Fault	No	EPT violations[5]
21	#CP	Control Protection Exception	Fault	Yes	RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump.
22-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT *n* instruction.

Booting SWEB

- Grand Unified Bootloader (GRUB)

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

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

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

- 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

- 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

- 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

- 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

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

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

- 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

- 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

- 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
  - ...

- 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

Questions?