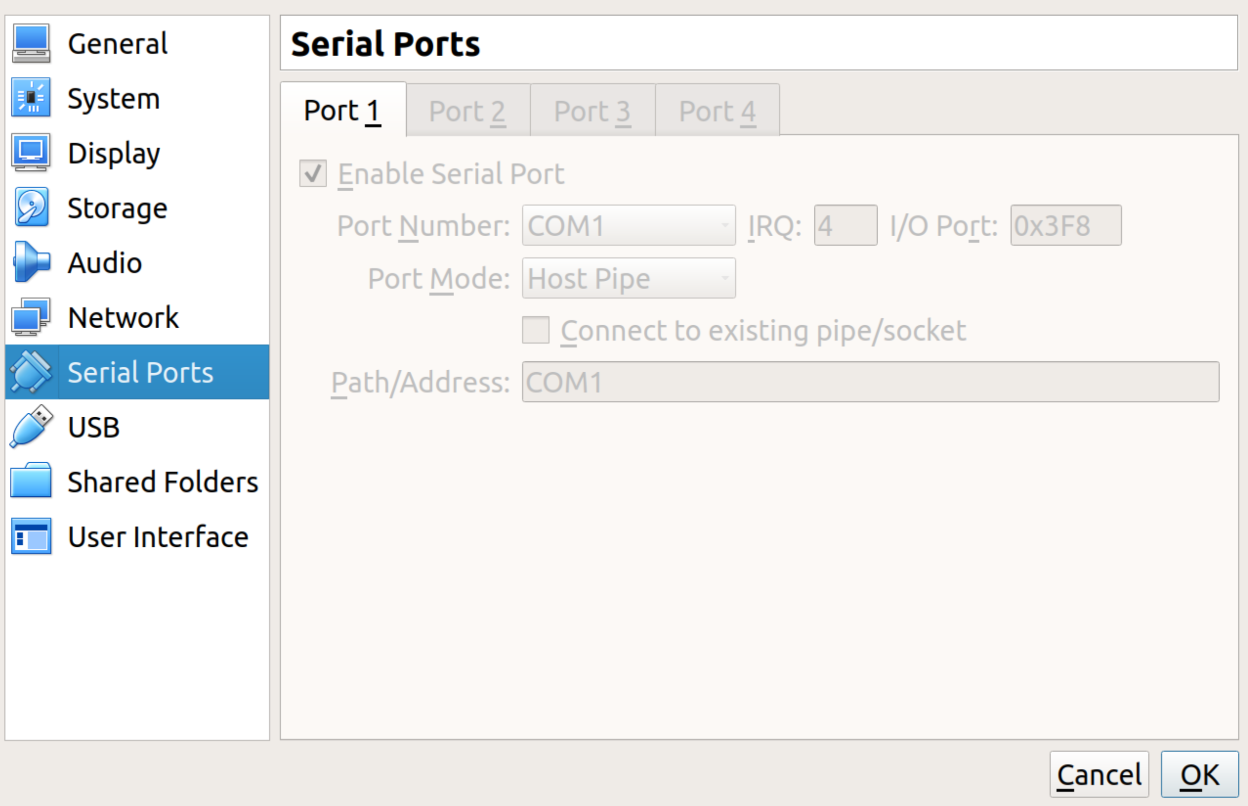
Kernel Debugging for Segmentation and Paging Analysis in IA32

Setup for Kernel Debugging

VM Setup

Debugger VM Configuration



Serial Ports

☑ Enable Serial Port

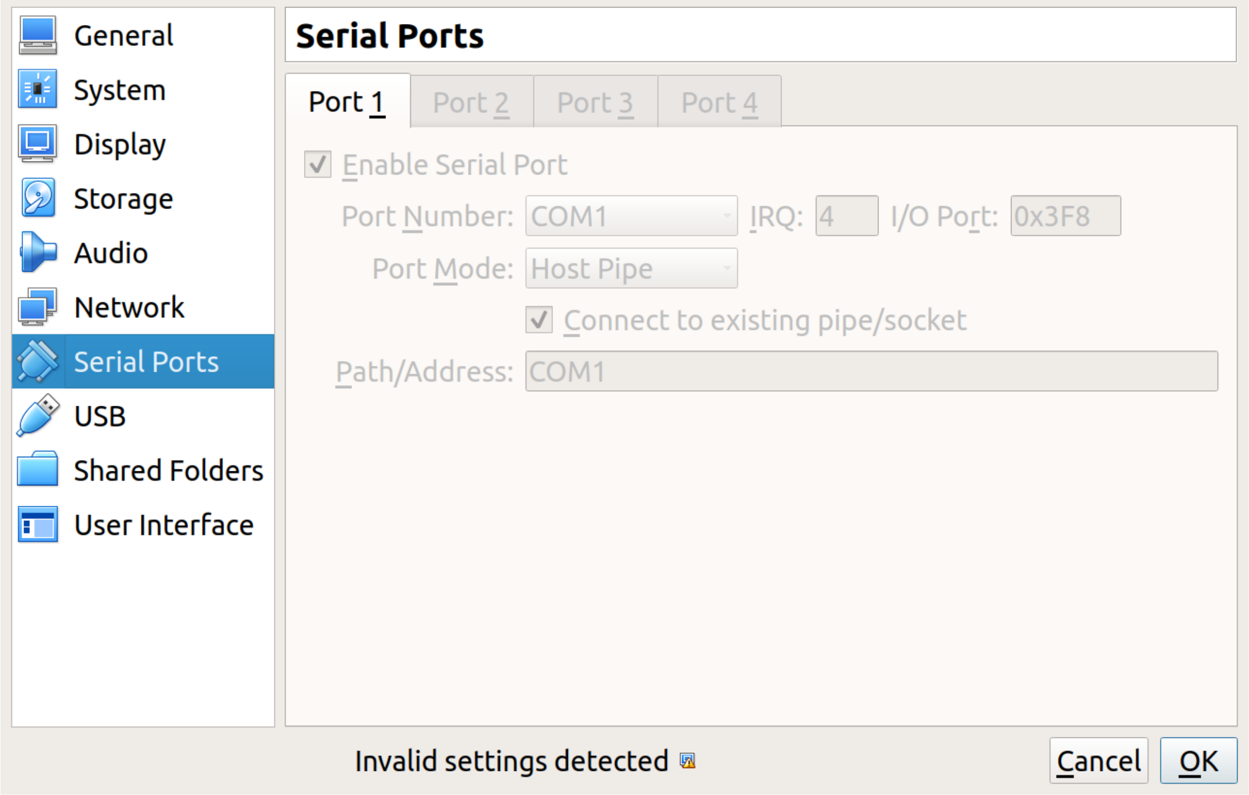
Port Number: COM1 IRQ: 4 I/O Port: 0x3F8

Port Mode: Host Pipe

☐ Connect to existing pipe/socket

Path/Address: COM1

Debuggee VM Configuration



Serial Ports

Port 1

☑ Enable Serial Port

Port Number: COM1 IRQ: 4 I/O Port: 0x3F8

Port Mode: Host Pipe

☑ Connect to existing pipe/socket

Path/Address: COM1

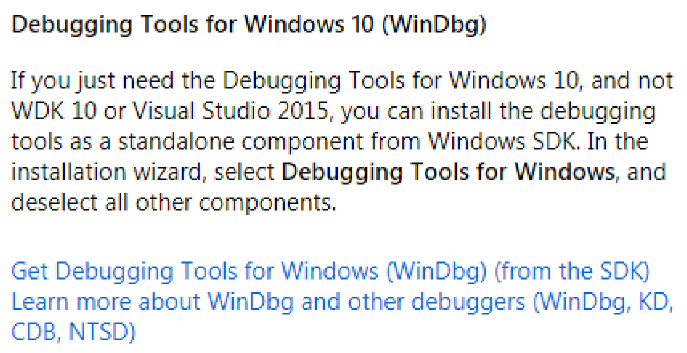
Invalid settings detected

VM Software Setup

Debugger System

WinDBG Installation

https://developer.microsoft.com/en-us/windows/hardware/download-windbg



Debugging Tools for Windows 10 (WinDbg)

If you just need the Debugging Tools for Windows 10, and not WDK 10 or Visual Studio 2015, you can install the debugging tools as a standalone component from Windows SDK. In the installation wizard, select **Debugging Tools for Windows**, and deselect all other components.

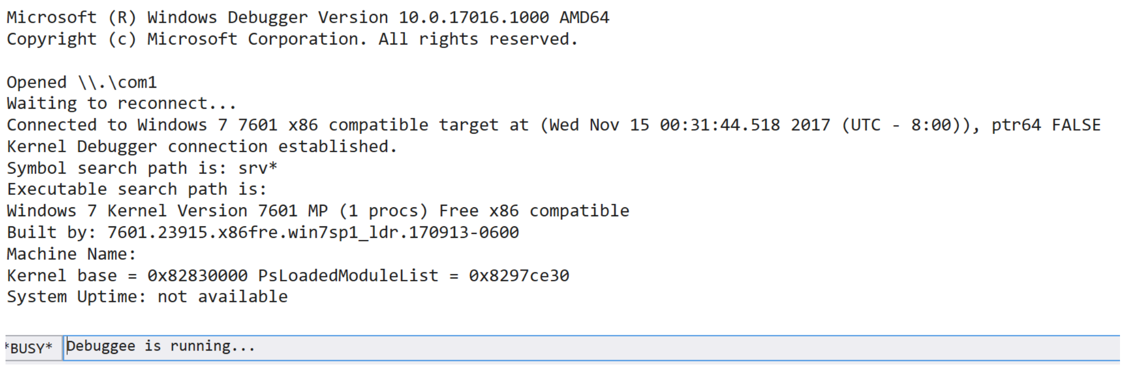
Get Debugging Tools for Windows (WinDbg) (from the SDK)
Learn more about WinDbg and other debuggers (WinDbg, KD, CDB, NTSD)

You can also install the WinDBG Preview version from the Microsoft App Store.

WinDBG Usage

1. Run WinDBG and press Ctrl+K to invoke Kernel Debugging.
2. in COM, enter: Baudrate: 115200, Port: COM1, Resets: 0 and verify that Pipe and Reconnect are unchecked (important).

You'll be presented with the following output: Opened \\.\com1 Waiting to reconnect…



```
Microsoft (R) Windows Debugger Version 10.0.17016.1000 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Opened \\.\com1
Waiting to reconnect...
Connected to Windows 7 7601 x86 compatible target at (Wed Nov 15 00:31:44.518 2017 (UTC - 8:00)), ptr64 FALSE
Kernel Debugger connection established.
Symbol search path is: srv*
Executable search path is:
Windows 7 Kernel Version 7601 MP (1 procs) Free x86 compatible
Built by: 7601.23915.x86fre.win7sp1_ldr.170913-0600
Machine Name:
Kernel base = 0x82830000 PsLoadedModuleList = 0x8297ce30
System Uptime: not available

*BUSY* Debuggee is running...
```

Debuggee System

1. In an Administrator command line console execute:

For Windows Vista+

```
bcdedit /dbgsettings serial debugport:1 baudrate:115200
```

For Windows XP and Windows Server 2003

```
bootcfg /debug on /port com1 /baud 115200 /id 1
```

This command sets COM1 and a baud rate of 115200.

2. Reboot

Quite early during the booting stage, WinDBG on the other machine should detect the debuggee is running.

WinDBG Kernel Cheatsheet

http://windbg.info/doc/1-common-cmds.html

- Command help

Open a new window with command help

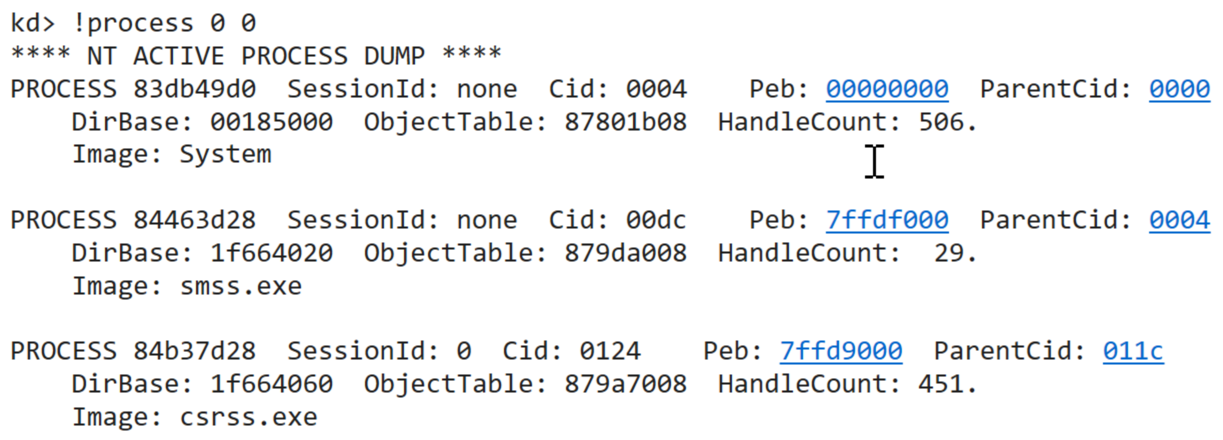
```
.hh <command>
```

WinDBG inline documentation

```
.hh <command>
```

- List all processes

```
!process 0 0
```



```
kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 83db49d0  SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 00185000  ObjectTable: 87801b08  HandleCount: 506.
    Image: System

PROCESS 84463d28  SessionId: none  Cid: 00dc    Peb: 7ffdf000  ParentCid: 0004
    DirBase: 1f664020  ObjectTable: 879da008  HandleCount:  29.
    Image: smss.exe

PROCESS 84b37d28  SessionId: 0  Cid: 0124    Peb: 7ffd9000  ParentCid: 011c
    DirBase: 1f664060  ObjectTable: 879a7008  HandleCount: 451.
    Image: csrss.exe
```

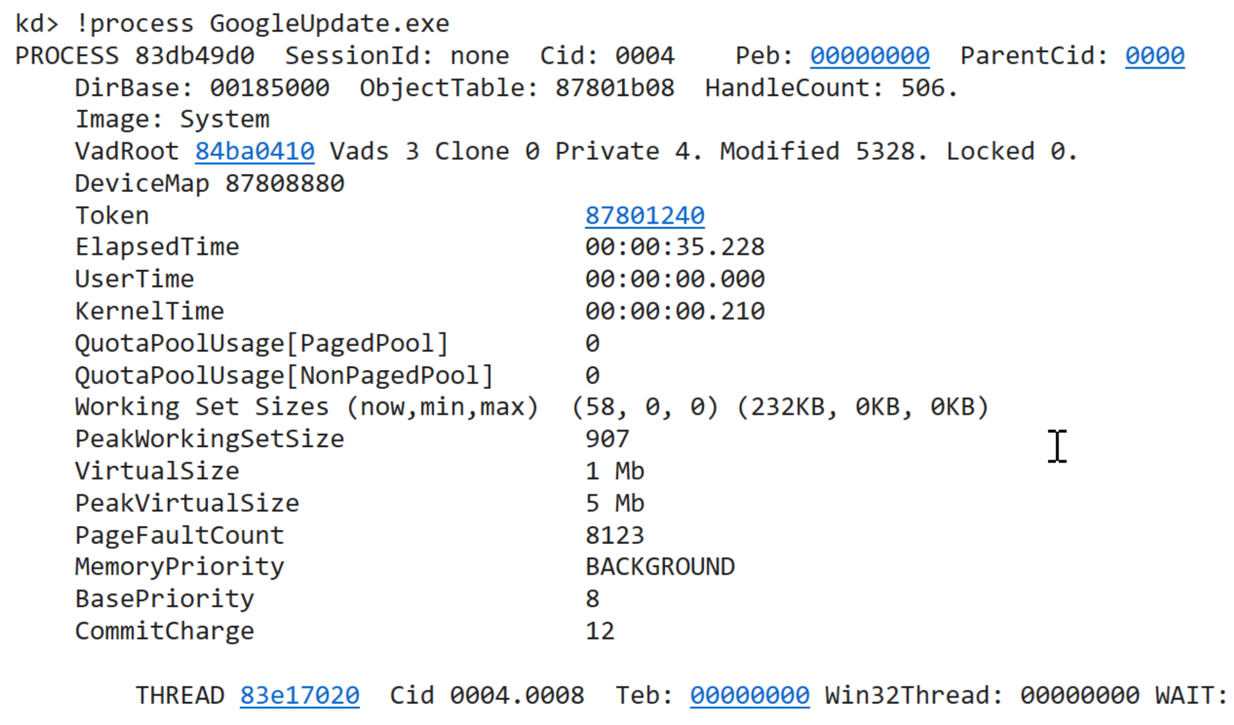
- Explanation:

0: All active processes
0: Minimal information

https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/-process

- Information about a specific process

!process GoogleUpdate.exe



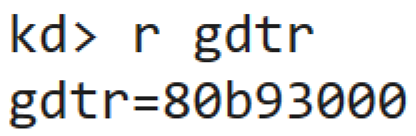
```
kd> !process GoogleUpdate.exe
PROCESS 83db49d0  SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 00185000  ObjectTable: 87801b08  HandleCount: 506.
    Image: System
    VadRoot 84ba0410 Vads 3 Clone 0 Private 4. Modified 5328. Locked 0.
    DeviceMap 87808880
    Token                             87801240
    ElapsedTime                       00:00:35.228
    UserTime                          00:00:00.000
    KernelTime                        00:00:00.210
    QuotaPoolUsage[PagedPool]         0
    QuotaPoolUsage[NonPagedPool]      0
    Working Set Sizes (now,min,max)  (58, 0, 0) (232KB, 0KB, 0KB)
    PeakWorkingSetSize                907
    VirtualSize                       1 Mb
    PeakVirtualSize                   5 Mb
    PageFaultCount                    8123
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      12

        THREAD 83e17020  Cid 0004.0008  Teb: 00000000 Win32Thread: 00000000 WAIT:
```

https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/-process

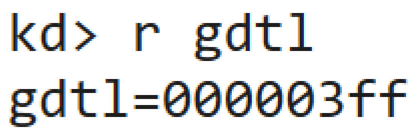
- Show GDT base address, length and raw contents

r gdtr



```
kd> r gdtr
gdtr=80b93000
```

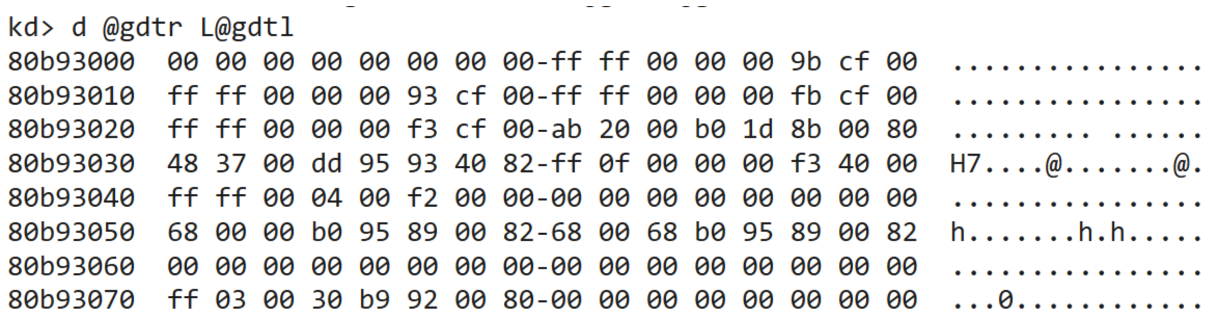
r gdtl



```
kd> r gdtl
gdtl=000003ff
```

WinDbg lists the upper 32 bits of the GDTR as “gdtr” but the lower 16 bits as “gdtl” (Intel has no reference about gdtl).

d <gdt_base_address> L<length>

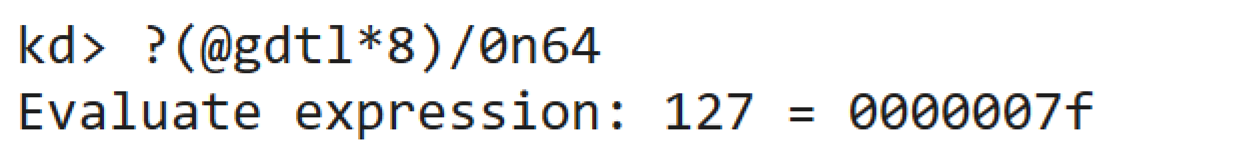


```
kd> d @gdtr L@gdtl
80b93000  00 00 00 00 00 00 00 00-ff ff 00 00 00 9b cf 00  ................
80b93010  ff ff 00 00 00 93 cf 00-ff ff 00 00 00 fb cf 00  ................
80b93020  ff ff 00 00 00 f3 cf 00-ab 20 00 b0 1d 8b 00 80  ......... ......
80b93030  48 37 00 dd 95 93 40 82-ff 0f 00 00 00 f3 40 00  H7....@.......@.
80b93040  ff ff 00 04 00 f2 00 00-00 00 00 00 00 00 00 00  ................
80b93050  68 00 00 b0 95 89 00 82-68 00 68 b0 95 89 00 82  h.......h.h.....
80b93060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
80b93070  ff 03 00 30 b9 92 00 80-00 00 00 00 00 00 00 00  ...0............
```

https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/d--da--db--dc--dd--dd--df--dp--dq--du--dw--dw--dyb--dyd--display-memor
https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/address-and-address-range-syntax

- How many segment descriptors do we have?

Each segment descriptor is 64 bits.
The GDT size is 0x3FF **bytes** (r gdtl), so it is 0x3FF * 8 = 8184 bits.



```
kd> ?(@gdtl*8)/0n64
Evaluate expression: 127 = 0000007f
```

The **0n** prefix expresses that the number is in decimal notation.

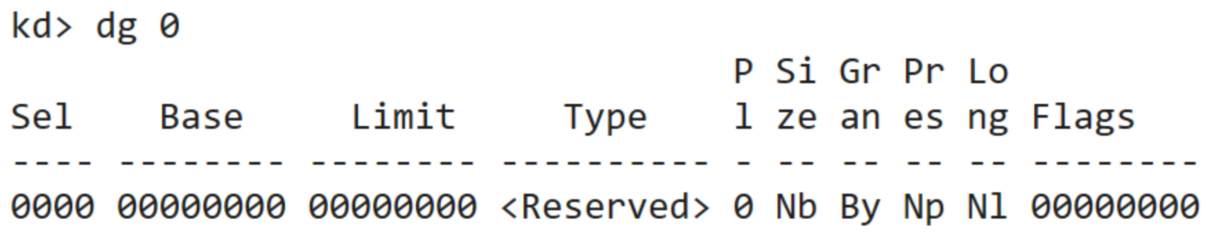
There are 127 segment descriptors.

- Show segment descriptors

The dg command shows the segment descriptor for the specified selector.

dg <first_selector> [last_selector]

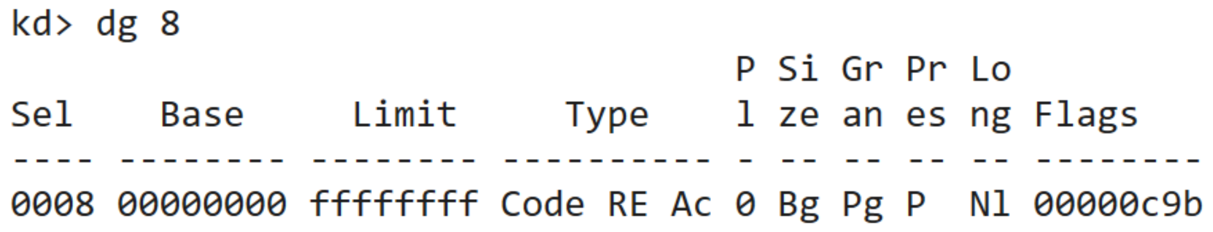
dg 0



```
kd> dg 0
                                         P Si Gr Pr Lo
Sel    Base     Limit       Type         l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0000 00000000 00000000 <Reserved> 0 Nb By Np Nl 00000000
```

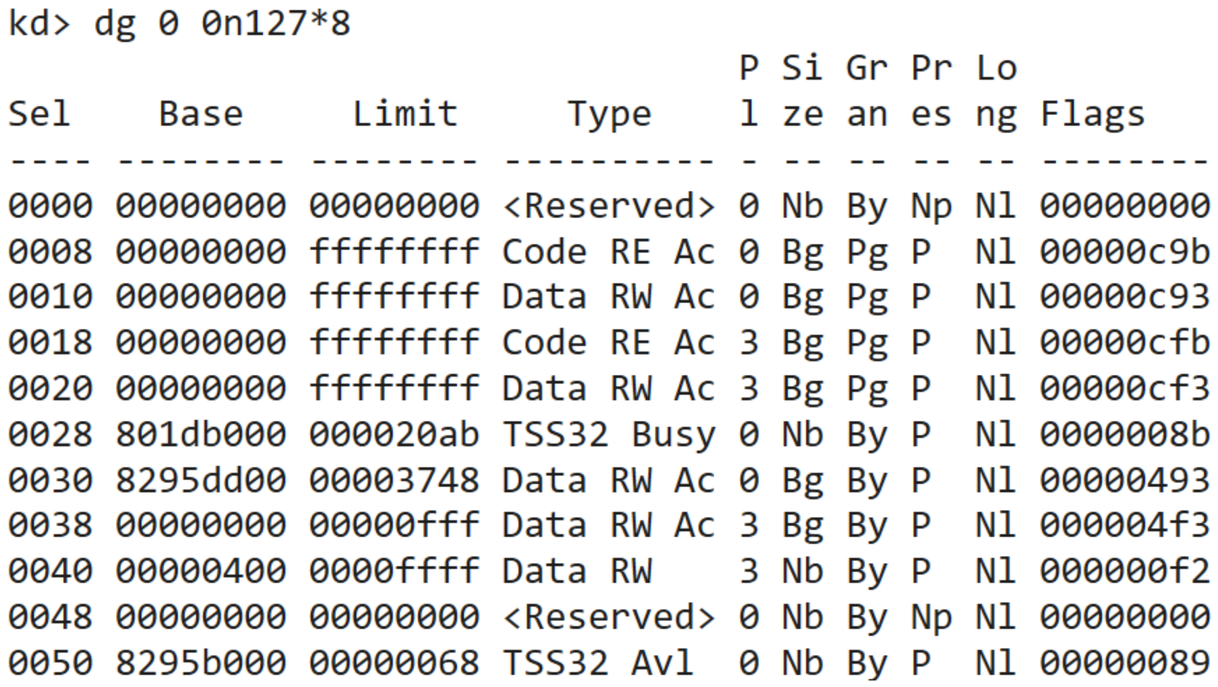
Get information about the second segment selector

dg 8



```
kd> dg 8
                                  P Si Gr Pr Lo
Sel    Base     Limit     Type    l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0008 00000000 ffffffff Code RE Ac 0 Bg Pg P  Nl 00000c9b
```

dg 0 0n127*8 (or dg 0 0X3F8) (or dg 0 0x3FF-8)



```
kd> dg 0 0n127*8
                                  P Si Gr Pr Lo
Sel    Base     Limit     Type    l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0000 00000000 00000000 <Reserved> 0 Nb By Np Nl 00000000
0008 00000000 ffffffff Code RE Ac 0 Bg Pg P  Nl 00000c9b
0010 00000000 ffffffff Data RW Ac 0 Bg Pg P  Nl 00000c93
0018 00000000 ffffffff Code RE Ac 3 Bg Pg P  Nl 00000cfb
0020 00000000 ffffffff Data RW Ac 3 Bg Pg P  Nl 00000cf3
0028 801db000 000020ab TSS32 Busy 0 Nb By P  Nl 0000008b
0030 8295dd00 00003748 Data RW Ac 0 Bg By P  Nl 00000493
0038 00000000 00000fff Data RW Ac 3 Bg By P  Nl 000004f3
0040 00000400 0000ffff Data RW    3 Nb By P  Nl 000000f2
0048 00000000 00000000 <Reserved> 0 Nb By Np Nl 00000000
0050 8295b000 00000068 TSS32 Avl  0 Nb By P  Nl 00000089
```

IMPORTANT

The first segment descriptor is null.

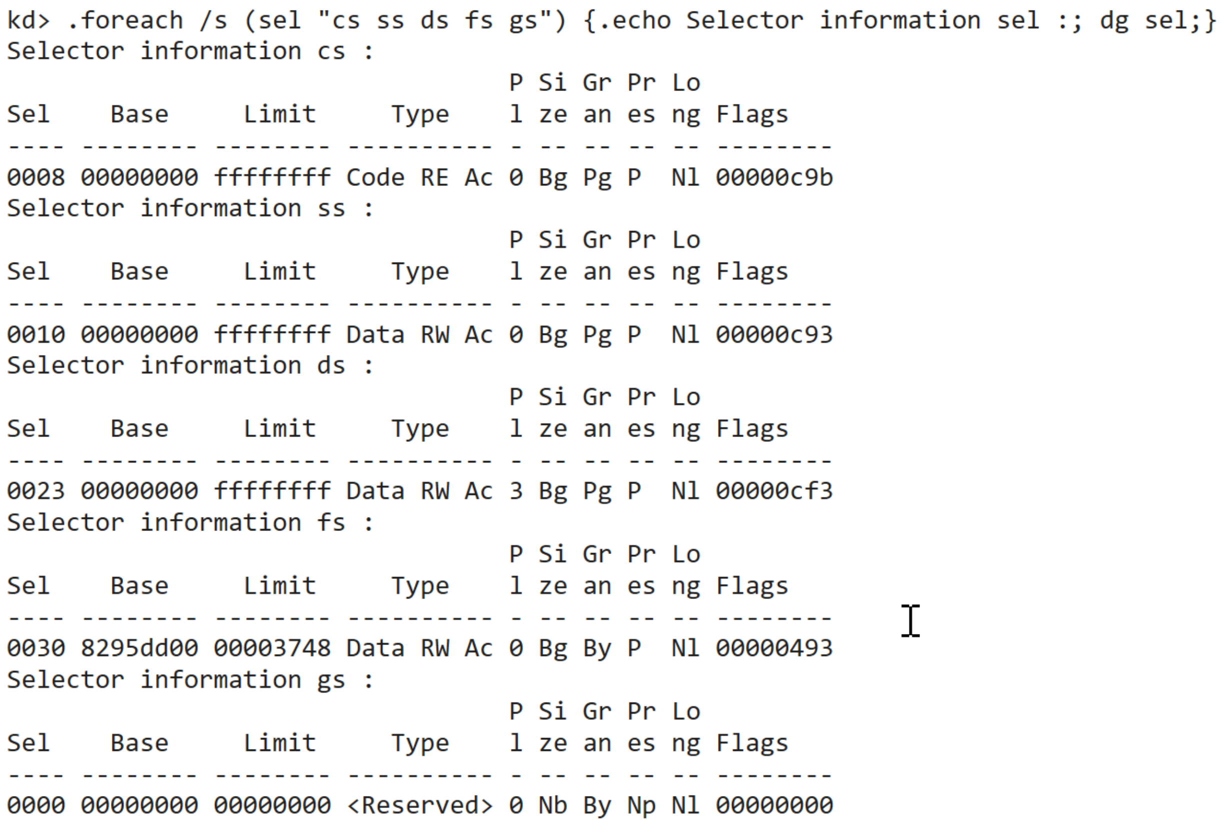
The following 4 segment descriptors is how the operating system renders segmentation to its minimal use.

Each of those segments cover the whole memory space. (from 0x00000000 to 0xffffffff).

The first two segments are kernel segments given that their DPL field is 0 (highest privilege). The other two are user segments given that they DPL field is 3 (lowest privilege).

For each privilege there are one Code segment and one Data segment.

.foreach /s (sel "cs ss ds fs gs") {.echo Selector information sel :; dg sel;}



```
kd> .foreach /s (sel "cs ss ds fs gs") {.echo Selector information sel :; dg sel;}
Selector information cs :
                                  P Si Gr Pr Lo
Sel    Base     Limit      Type   l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0008 00000000 ffffffff Code RE Ac 0 Bg Pg P  Nl 00000c9b
Selector information ss :
                                  P Si Gr Pr Lo
Sel    Base     Limit      Type   l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0010 00000000 ffffffff Data RW Ac 0 Bg Pg P  Nl 00000c93
Selector information ds :
                                  P Si Gr Pr Lo
Sel    Base     Limit      Type   l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0023 00000000 ffffffff Data RW Ac 3 Bg Pg P  Nl 00000cf3
Selector information fs :
                                  P Si Gr Pr Lo
Sel    Base     Limit      Type   l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0030 8295dd00 00003748 Data RW Ac 0 Bg By P  Nl 00000493
Selector information gs :
                                  P Si Gr Pr Lo
Sel    Base     Limit      Type   l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0000 00000000 00000000 <Reserved> 0 Nb By Np Nl 00000000
```

/s specifies that the variables should be retrieved from the string that follows.

Address Translation using WinDBG (Hero Mode)

- ## Choosing Linear/Virtual Address

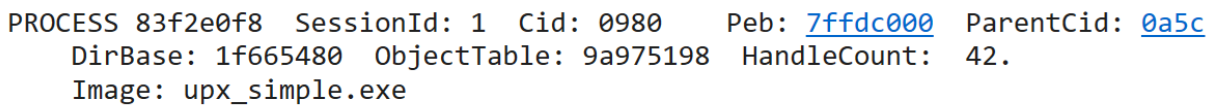
From a process you know, choose an address that contains a string. For example:

1. Execute upx_simple.exe

You can see that a modal window shows up with a title containing “trying to unpack me”.

2. List all processes running in debugee

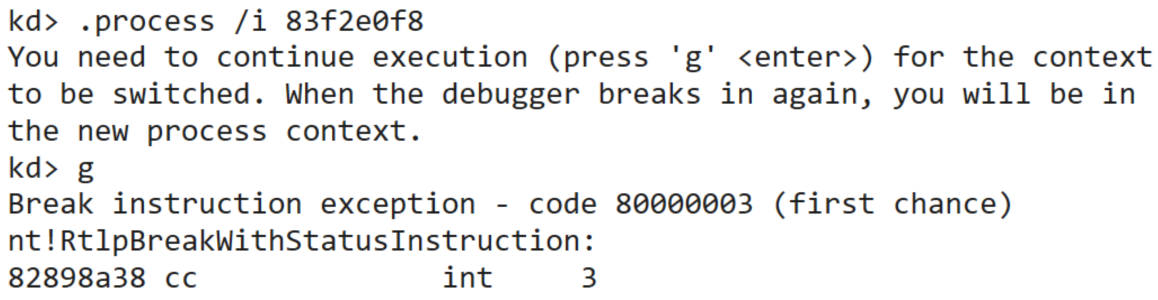
!process 0 0



```
PROCESS 83f2e0f8  SessionId: 1  Cid: 0980    Peb: 7ffdc000  ParentCid: 0a5c
    DirBase: 1f665480  ObjectTable: 9a975198  HandleCount:  42.
    Image: upx_simple.exe
```

3. Switch context interactively to that process (press “g” to continue process execution)

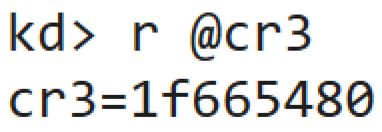
.process /i <addr>



```
kd> .process /i 83f2e0f8
You need to continue execution (press 'g' <enter>) for the context
to be switched. When the debugger breaks in again, you will be in
the new process context.
kd> g
Break instruction exception - code 80000003 (first chance)
nt!RtlpBreakWithStatusInstruction:
82898a38 cc              int     3
```

4. Check the context switch was correct

“r @cr3” value and process DirBase must match:

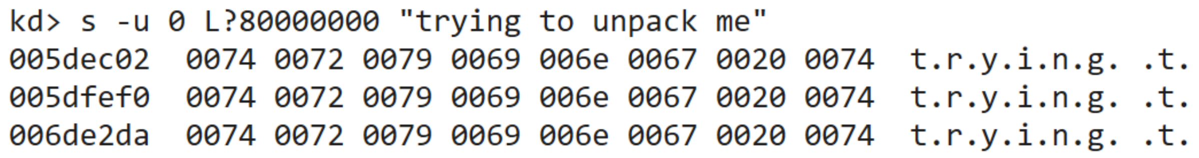


```
kd> r @cr3
cr3=1f665480
```

5. Search that string in memory to find its virtual/linear address

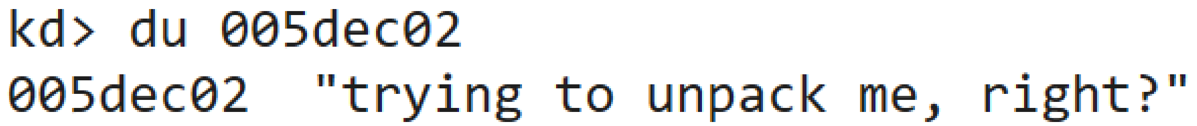
Low 2GB range (0x00000000 through 0x7fffffff) → Used by the process

s -u 0 L?80000000 “trying to unpack me” # search for unicode strings in specific range



```
kd> s -u 0 L?80000000 "trying to unpack me"
005dec02  0074 0072 0079 0069 006e 0067 0020 0074  t.r.y.i.n.g. .t.
005dfef0  0074 0072 0079 0069 006e 0067 0020 0074  t.r.y.i.n.g. .t.
006de2da  0074 0072 0079 0069 006e 0067 0020 0074  t.r.y.i.n.g. .t.
```

Selected virtual address: 0x005dec02



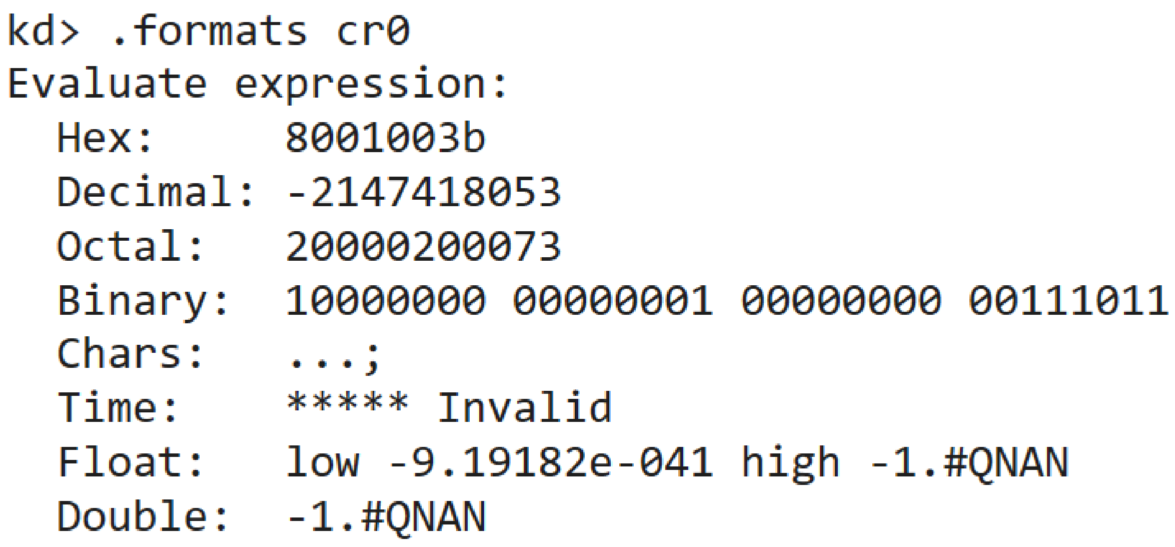
```
kd> du 005dec02
005dec02  "trying to unpack me, right?"
```

- **Are Protected Mode and Paging enabled?**

NOTE: bit count starting at 0!

Register CR0, bit 31 (MSB), defines if paging is enabled (bit set).
Register CR0, bit 0 (LSB), defines if protected mode is enabled (bit set).

.formats @cr0



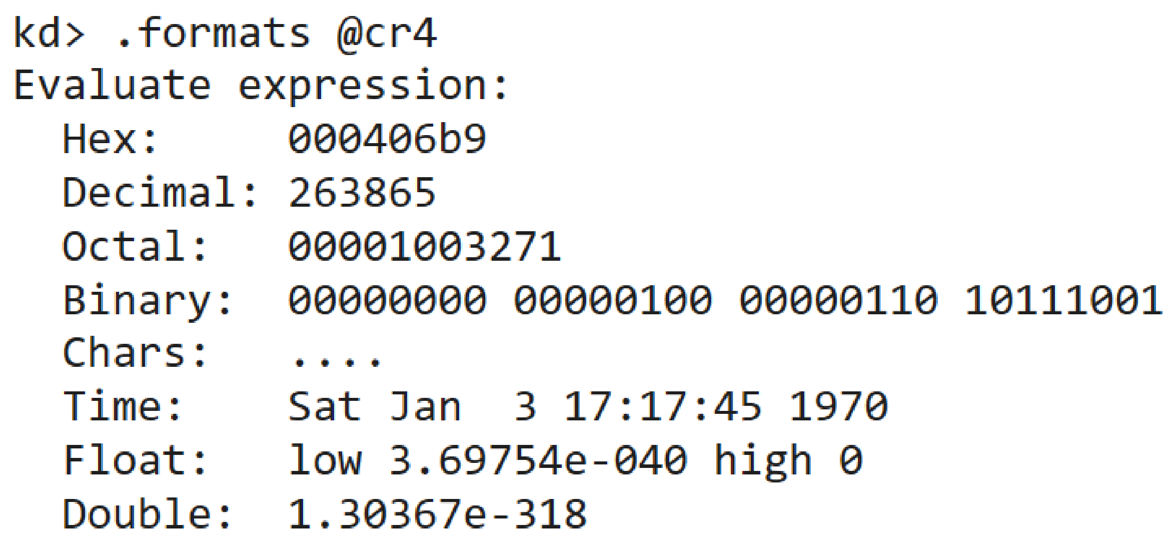
```
kd> .formats cr0
Evaluate expression:
  Hex:     8001003b
  Decimal: -2147418053
  Octal:   20000200073
  Binary:  10000000 00000001 00000000 00111011
  Chars:   ...;
  Time:    ***** Invalid
  Float:   low -9.19182e-041 high -1.#QNAN
  Double:  -1.#QNAN
```

Bit 31 set. Paging is enabled!
Bit 0 set. Protected mode is enabled!

Is PAE enabled?

Register CR4, bit 5, defines if PAE is enabled (bit set).
Register CR4, bit 4 (PSE bit), defines if pages of 2MB can be used.
If CR4.PSE = 0 → 4KB pages

.formats @cr4



```
kd> .formats @cr4
Evaluate expression:
  Hex:     000406b9
  Decimal: 263865
  Octal:   00001003271
  Binary:  00000000 00000100 00000110 10111001
  Chars:   ....
  Time:    Sat Jan  3 17:17:45 1970
  Float:   low 3.69754e-040 high 0
  Double:  1.30367e-318
```

Bit 5 set. PAE is enabled!
Bit 4 set. PSE is enabled, 2MB pages[1] can be enabled.
Only 2MB pages will be used if bit PDE (Page Directory Entry) PS bit 7 is set[2]. 4KB otherwise.

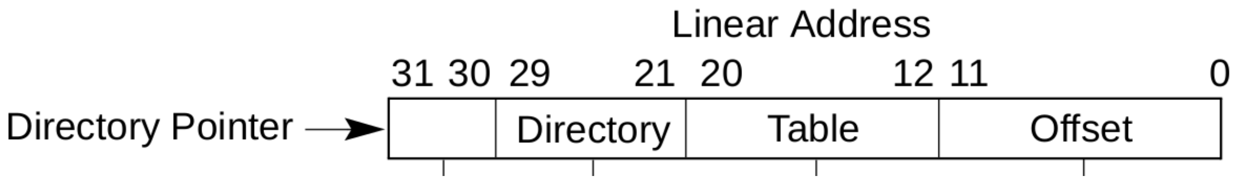
[1] 2MB with PAE, 4MB without PAE

[2] Intel® 64 and IA-32 Architectures, Software Developer’s Manual, Volume 3A, System Programming Guide, Part 1, 4-8 Vol. 3A

- **Linear/Virtual Address Decomposed**

- **Assuming pages of 4KB:**

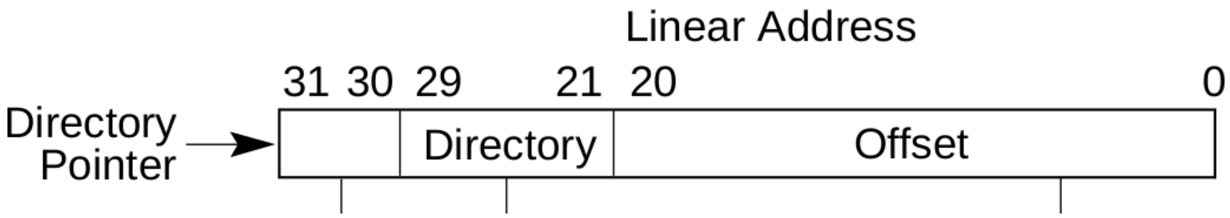
Linear Address = 0x005dec02 = 0b00-000000010-111011110-110000000010



PDPT Index = 0b00 = 0
PD Index = 0b000000010 = 2
PT Index = 0b111011110 = 0x1DE = 478
Physical Page Offset = 0b110000000010 = 0xC02

- **Assuming pages of 2MB:**

Linear Address = 0x005dec02 = 0b00-000000010-111011110110000000010



PDPT Index = 0b00 = 0
PD Index = 0b000000010 = 2
PT Index = 0b111011110 = 0x1DE = 478
Physical Page Offset = 0b110000000010 = 0xC02

NOTE: Take into account that PDPT Index and PD Index are the same with both approaches.

4KB or 2MB pages?

We know CR4.PSE (bit 4) is set. So pages can be of 2MB.

If PDE PS flag (bit 7) is 1 → Pages will be 2MB.
If PDE PS flag (bit 7) is 0 → Pages will be 4KB.

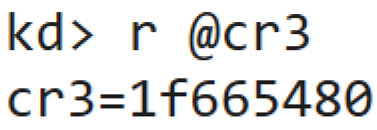
NOTE: Depending on this we will know what's the real format of the virtual/linear address, thus we will know if Page Tables are used in the address translation process or no.

PDPT Physical Base Address

PDPT Physical Base Address → CR3[5:31] (with PAE enabled)[3]

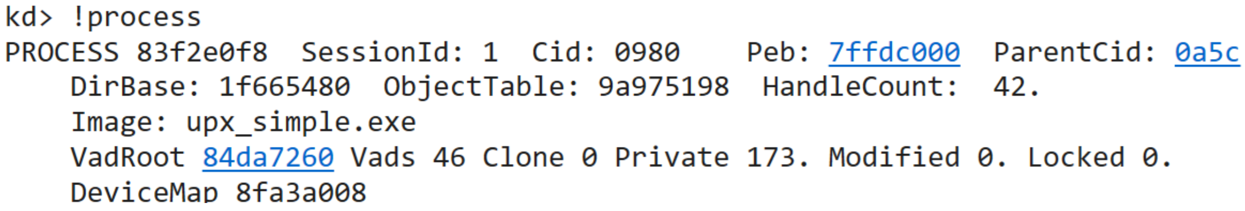
NOTE: When Intel documentation says CR3 bits 0:4 are ignored, it means that they are part of the PDPT physical base address but they are set to 0.[4]

r @cr3



```
kd> r @cr3
cr3=1f665480
```

This should match *DirBase* value of current process (after interactive process context switch - check that *Image* value is *upx_simple.exe*):



```
kd> !process
PROCESS 83f2e0f8  SessionId: 1  Cid: 0980    Peb: 7ffdc000  ParentCid: 0a5c
    DirBase: 1f665480  ObjectTable: 9a975198  HandleCount:  42.
    Image: upx_simple.exe
    VadRoot 84da7260 Vads 46 Clone 0 Private 173. Modified 0. Locked 0.
    DeviceMap 8fa3a008
```

PDPT Physical Base Address = 0x1f665480

[3] Intel® 64 and IA-32 Architectures, Software Developer's Manual, Volume 3A, System Programming Guide, Part 1, Vol. 3A 4-13, Section 4.4.1

[4] "If MOV to CR3 is executed while the logical processor is using PAE paging, the PDPTEs are loaded from the address being loaded into CR3." - Intel System Programming Guide, Vol. 3A 4-13, Section 4.4.1

- **PDPTE Physical Address**

PDPT Physical Base Address = 0x1f665480 (from previous section)
PDPT Index = 0 (from previous section)
PDPT Entry Length = 64 bits = 8 bytes (from Annex)

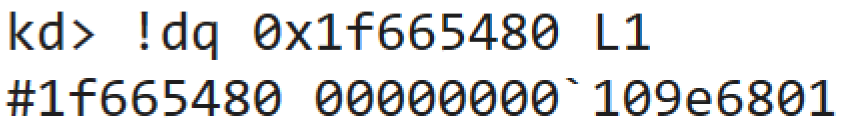
PDPTE address = 0x1f665480 + 0 * 8 = 0x1f665480

- **PDPTE Value**

PDPTE address = 0x1f665480 (from previous section)

NOTE: PDPT Base Address is a **PHYSICAL** address. We need to read the contents of 0x1f665480 **PHYSICAL** address. We cannot use regular d* commands, but use !d* commands.

!d* # read physical address contents (dq for quad-word → 64 bits)



```
kd> !dq 0x1f665480 L1
#1f665480 00000000`109e6801
```

PDPTE value = 0x00000000`109e6801

- **PD Physical Base Address**

PDPTE value = 0x00000000`109e6801 = 0b10000100111100110-100000000001
PD Physical Base address = PDPTE[12:M-1][5] = 0b10000100111100110-***000000000000*** = 0x109E6000

NOTE: Again, the 12 LSB are ignored, so we set them to 0.

- **PDE Physical Address**

PD Physical Base Address = 0x109E6000 (from previous section)
PDE Index = 2 (from previous section)
PDE length = 64 bits = 8 bytes (from Annex)

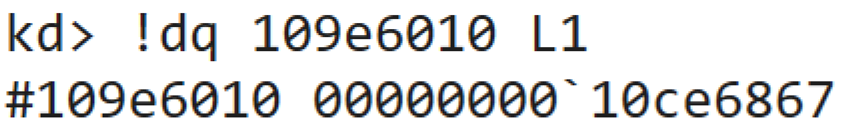
PDE Address = 0x109E6000 + 2*8 = 0x109E6000 + 0x10 = **0x109E6010**

[5] M → MAXPHYADDR → In 32-bit processors with PAE enabled this value is usually 36. It does not matter in our example because starting at bit 29, all remaining MSB are 0.

- **PDE Value. Is PDE.PS bit enabled? (2MB or 4KB pages)**

PDE Address = 0x109E6010

!dq <addr> L<length> # print 64 btis value honoring little endianess



```
kd> !dq 109e6010 L1
#109e6010 00000000`10ce6867
```

PDE Value: 0x00000000`10ce6867 = 0b10000110011100110100001100111

PDE.PS (bit N) = 0 → Pages are 4KB → **This means we have Page Tables!**

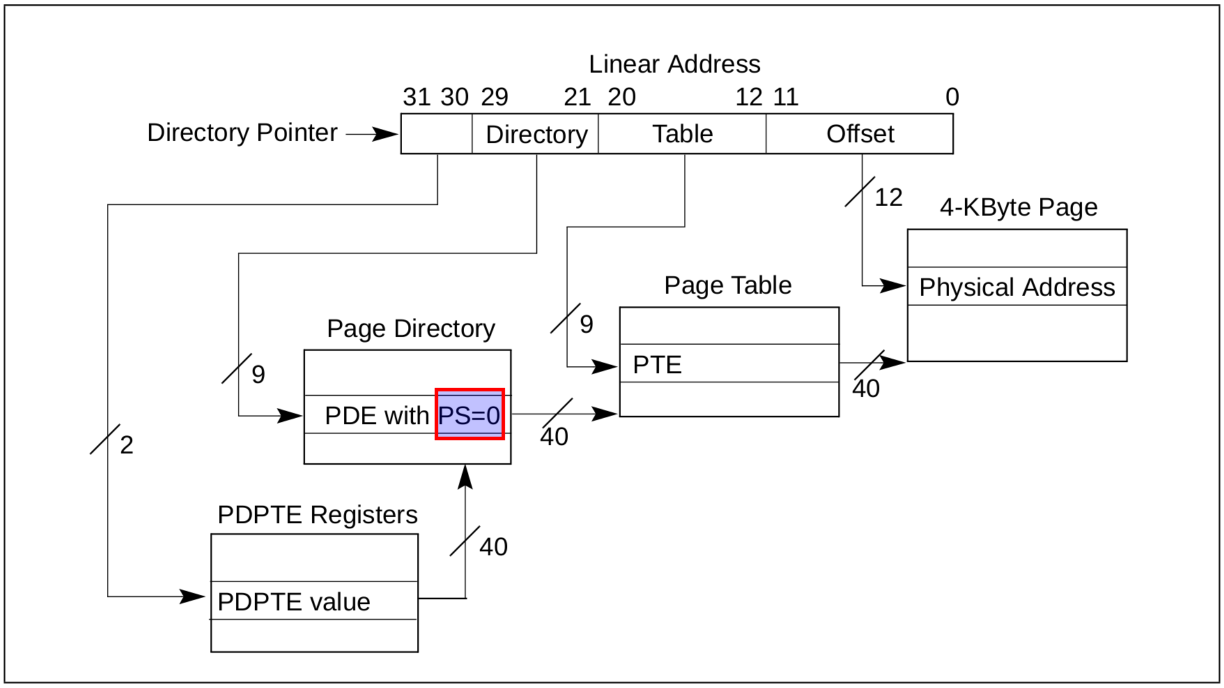


Figure 4-5. Linear-Address Translation to a 4-KByte Page using PAE Paging

● PT Base Physical Address

PDE value = 0x10ce6867 = 0b10000110011100110-100001100111

PT Base Physical Address = PDE[12:M-1] = 0b10000110011100110-**000000000000** = 0x10CE6000

PT Base Physical Address = 0x10CE6000

● PT Base Entry Physical Address

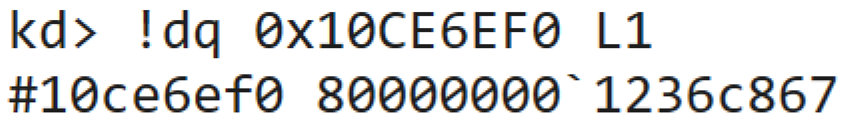
PT Base Physical Address = 0x10CE6000 (from previous section)
PTE Index = 0x1DE = 478 (from previous section)
PTE length = 64 bits = 8 bytes (from Annex)

PTE Physical Address = 0x10CE6000 + 0x1DE*8 = 0x10CE6EF0

● PT Entry Value

PTE Address = 0x10CE6EF0 (from previous section)

```
!dq <addr> L<length>  # print 64 btis value honoring little endianess
```



```
kd> !dq 0x10CE6EF0 L1
#10ce6ef0 80000000`1236c867
```

PTE Value = 0x80000000`1236C867

● Page Frame Base Physical Address

PTE value = 0x80000000`1236C867 = 0b(...)0b10010001101101100-100001100111

PF Base Physical Address = PTE[12:M-1] = 0b10010001101101100-**000000000000** = 0x1236C000

PF Base Physical Address = 0x1236C000

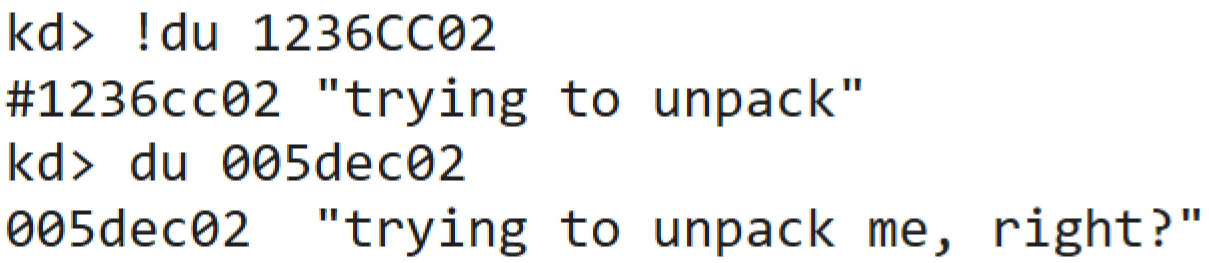
- **Physical Address**

PF Base Physical Address = 0x1236C000 (from previous section)
PF Offset = 0xC02 (from previous section)

Physical Address = 0x1236C000 + 0xC02 = 0x1236CC02

- **Physical Address Value**

Read contents of both physical and virtual address and check they match:



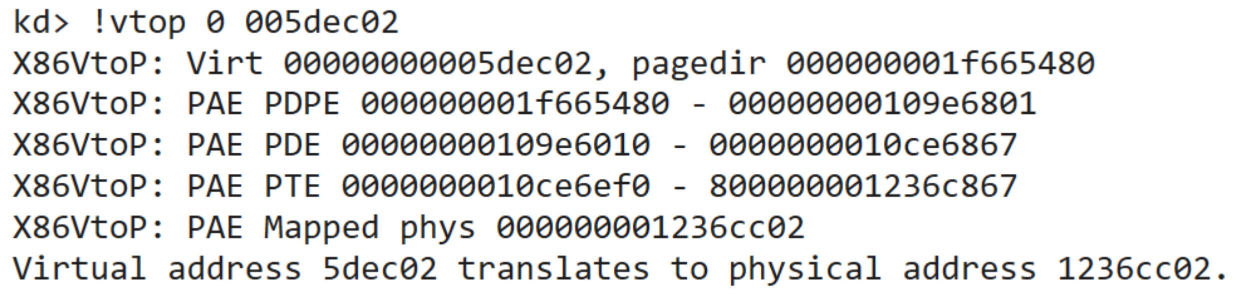
```
kd> !du 1236CC02
#1236cc02 "trying to unpack"
kd> du 005dec02
005dec02  "trying to unpack me, right?"
```

Address Translation using WinDBG (Kiddy Mode)

!vtop 0 <virtual_addr>

This extension converts a virtual address to the corresponding physical address, and displays other page table and page directory information.

The 0 means that the context of the current process is used. So you have to change the context using “.process /i <addr>” (seen in previous sections)



```
kd> !vtop 0 005dec02
X86VtoP: Virt 00000000005dec02, pagedir 000000001f665480
X86VtoP: PAE PDPE 000000001f665480 - 00000000109e6801
X86VtoP: PAE PDE 00000000109e6010 - 0000000010ce6867
X86VtoP: PAE PTE 0000000010ce6ef0 - 800000001236c867
X86VtoP: PAE Mapped phys 000000001236cc02
Virtual address 5dec02 translates to physical address 1236cc02.
```

Annex: [Reminder] Paging structure under PAE

Depending on CR4.PSE (bit 4) and PDE PS flag (bit 7), pages will be 2MB or 4KB.

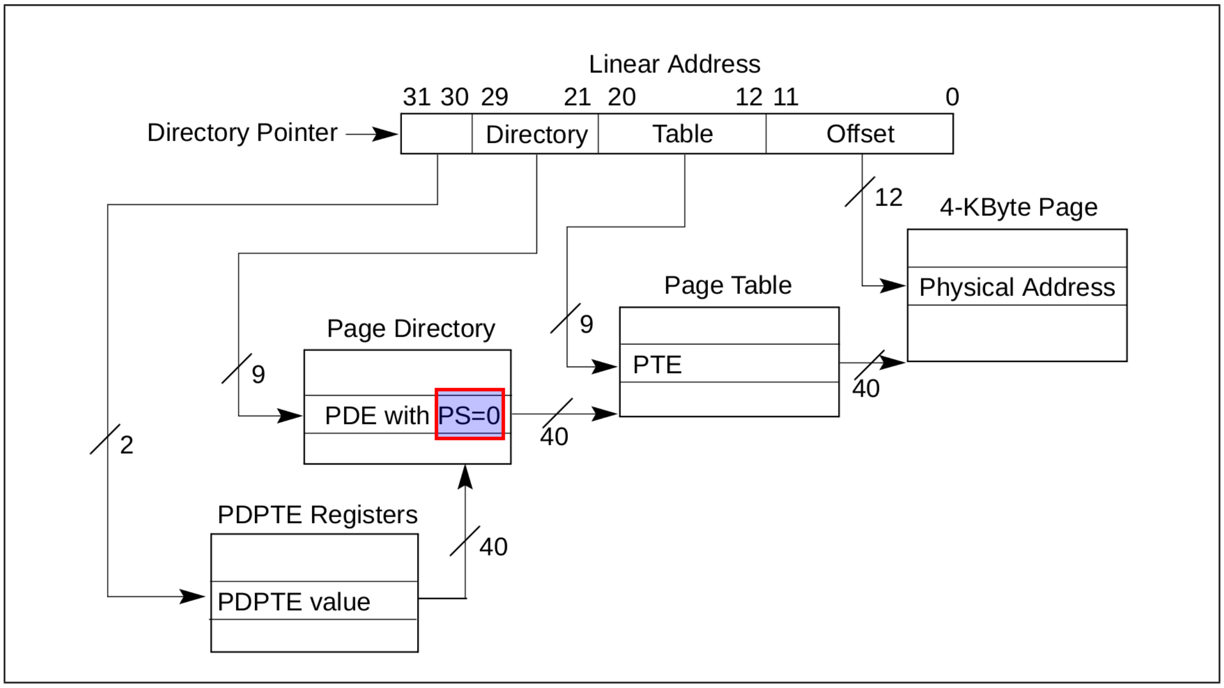


Figure 4-5. Linear-Address Translation to a 4-KByte Page using PAE Paging

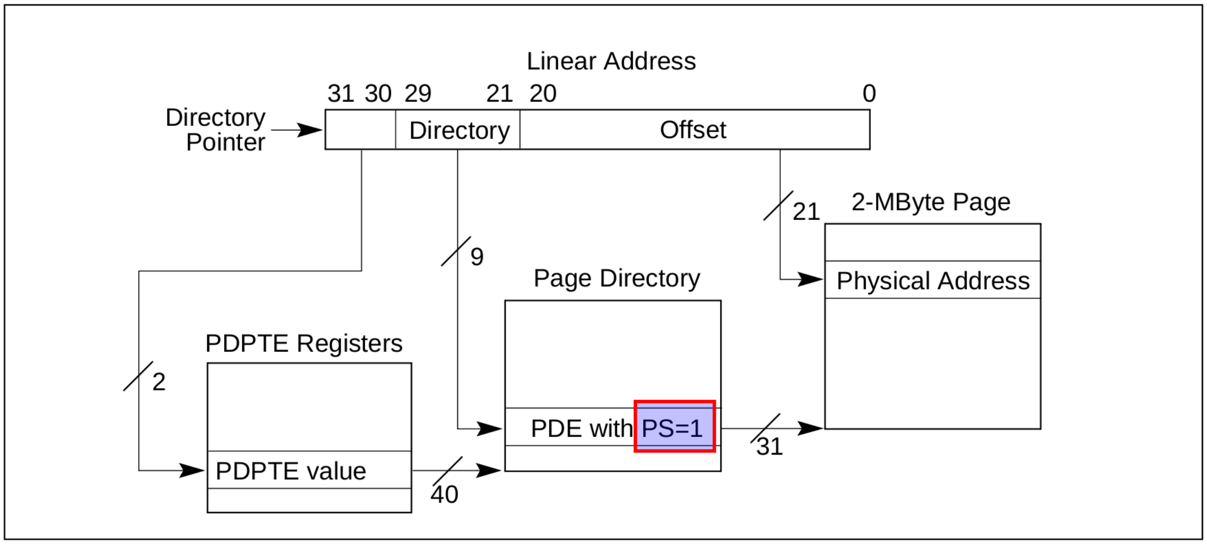


Figure 4-6. Linear-Address Translation to a 2-MByte Page using PAE Paging

	Bit 31 … 5	4 … 0
CR3	Physical Base Address of the PDPT	

32-bit Linear Address (under paging with PAE)

Bit 31, 30	29 … 21	20 … 12	11 … Bit 0
Index into PDPT	9-bit index into page directory	9-bit index into page table	12-bit offset into physical page

64-Bit PDPTE

Bit 63 … M	M-1 … 12	11 … 9	8 … 5	4	3	2	1	0
Reserved (i.e., zero)	Base Physical Address of Page Directory	Ignored	Reserved (zero)	PCD	PWT	0	0	P

64-Bit Page Directory Entry (PDE)

Bit 63 … M	M-1 … 12	11 … 8	7	6	5	4	3	2	1	0
Reserved (i.e., zero)	Base Physical Address of Page Table	Ignored	PS	0	A	PCD	PWT	U/S	W	P

64-Bit Page Table Entry (PTE)

Bit 63 … M	M-1 … 12	11 … 9	8	7	6	5	4	3	2	1	0
Reserved (i.e., zero)	Base Physical Address of Page	Avail	G	PAT	D	A	PCD	PWT	U/S	W	P