Descriptor Tables: GDT, IDT and LDT

Content by: johnfine@erols.com
HTML formatting by: volunteer_needed
Cleanup of HTML by: K.J.

Descriptor Format

There are two formats for descriptors:

- Gates
- nonGates

A gate descriptor contains

- Offset
- Selector
- Attributes

7	6	5	4	3	2	1	0
63............ 48		47.............32		31.............16		15..............0	
Offset 31.............16		Attributes		Selector		Offset 15..............0	

A descriptor which is not a gate contains

- Base
- Limit
- Attributes

7	6		5	4	3	2	1	0
63........56	55..52	51..48	47........40	39...............................16			15....................0	
Base 31........24	Attr	Limit 19..16	Attr	Base 23................................0			Limit 15....................0	

Because the 386 was designed with backwards compatibility for the 286, descriptors ended up with fragmented internal structure

- Base is a 32 bit value split between bits 16-39 and bits 56-63 of the descriptor.
- Limit is a 20 bit value split between bits 0-15 and bits 48-51 of the descriptor.
- Offset is a 32 bit values split between bits 0-15 and bits 48-63 of the descriptor.
- Selector is a 16 bit value in bits 16-31 of the descriptor.
- Attributes are a collection of values in bits 32-47 of a gate or in bits 40-47 plus 52-55 of a nongate.

Attributes

Unlike base, limit or offset, the attribute bits have no inherent bit numbers outside their use in the descriptor.

You can say that bit 0 of the base is in bit 16 of the descriptor; But there is no real basis for saying something like "bit ? of the attributes is in bit 40 of the descriptor.

Despite that, I have found it very useful to invent a bit numbering for the attributes:

- Bits 8 to 15 of the attributes are in descriptor bits 40 to 47
- Bits 0 to 7 of the attributes of a gate are in descriptor bits 32 to 39
- Bits 4 to 7 of the attributes of a nongate are in descriptor bits 52 to 55
- Bits 0 to 3 of the attributes of a nongate don't exist (a nongate descriptor has room for only 12 attribute bits and I chose to number them 4 through 15)

Descriptor byte	6		5	4
Descriptor bits	55..52	51..48	47.........40	39.........32
Gate Attribute bits			15.........................0	
nonGate Attribute bits	7....4		15.........8	

This numbering gives me a consistent way to pass all the attributes together as a 16-bit value to any subroutine or macro that builds descriptors. It also gives me a consistent way to define symbolic constants for attribute values that can be combined in source code without regard to which bytes of the descriptor the bits end up in.

Descriptor bit number		55	54	53	52	47	46	45	44	43	42	41	40	39..37	36..32
Attribute bit number		7	6	5	4	15	14	13	12	11	10	9	8	7....5	4....0
Gate	Task Gate					Present	DPL	DPL	0	Z	1	0	1	?	?
	Call Gate					Present	DPL	DPL	0	32Bit	1	0	0	Z	count
	Interrupt Gate					Present	DPL	DPL	0	32Bit	1	1	0	?	?
	Trap Gate					Present	DPL	DPL	0	32Bit	1	1	1	?	?
nonGate	TSS	Gran	?	Z	Avail	Present	DPL	DPL	0	32Bit	0	B	1		
	LDT	Gran	?	Z	Avail	Present	DPL	DPL	0	Z	0	1	0		
	Code Segment	Gran	Big	Z	Avail	Present	DPL	DPL	1	1	C	R	Acc		
	Data Segment	Gran	Big	Z	Avail	Present	DPL	DPL	1	0	E	W	Acc		

Bits marked "Z", "?", or "Avail" in the above table have no documented meanings. The descriptor will work correctly if all these bits are set to zero. The descriptor MAY work correctly if some of them are not zero. The documentation I have seen seems to indicate that those marked "Z" must be zero. Often this means Intel plans to assign a meaning to them later or has already assigned a meaning not covered by the documentation I read. The ones marked "Avail" are explicitly documented as being ignored by the CPU, so the programmer can use them for some other purpose. The ones marked "?" aren't mentioned one way or the other.

Attribute bit 15, "Present" must be set for a descritor to be used. Otherwise a fault will result. Clearing it is useful when implementing segment based virtual memory.

Attribute bits 13 and 14, "dpl" are the descriptor privilege level (see Intel documentation).

Attribute bit 11 in a gate or a TSS, "32bit" is set to indicate the 32 bit version. In an interrupt or trap gate IDT this bit decides whether the eip,cs,flags pushed on the stack are 16 bit or 32 bits each.

Attribute bits 0 to 4 in a call gate are the parameter count.

Attribute bit 7 in a nongate, "gran" controls the granularity of the limit field. When this bit is clear the limit is equal to the value in the limit field. When this bit is set:
```
limit = ( limit_field SHL 12 ) + 0xFFF
```

Attribute bit 9 in a TSS, "B" is set to indicate that the TSS is busy.

Attribute bit 6 in a code or data segment, "big" indicates the default size. In CS it controls the default for operand size and address size. In SS it controls the choice of sp vs. esp for push, pop (etc.). In DS, ES, FS, GS it has no effect.

Attribute bit 10 in a code segment, "C" is set for "conforming" code segments. See Intel documentation. If you don't understand this feature, don't set this bit.

Attribute bit 10 in a data segment, "E" is set for "expand down" data segments. See Intel documentation. If you don't understand this feature, don't set this bit.

Attribute bit 9 in a code segment, "R" is set to make a segment you can execute or read. It is cleared for execute-only. A code segment never permits write.

Attribute bit 9 in a data segment, "W" is set to make a segment you can read or write. It is cleared for read-only. A data segment never permits execute.

Attribute bit 8 in a code or data segment, "Acc" is set by the CPU to indicate that the segment has been accessed. If you do not want the CPU to do that extra write on the first access, you can preset the bit yourself.

GDT.INC

This file defines constants and macros to help you create GDT, LDT and IDT descriptors. By using these constants and macros, you can make your source code more readable and maintainable than with the usual methods of creating descriptors.

```
; gdtn.inc   symbols and macros for building descriptors
;
;  This is an HTML version of a subset of my GDT.INC file.
;  This GDTN.INC subset does not require my linker JLOC, it does require
;  NASM version 0.98-J4 or later.
;
;  You can find (nonHTML) GDTN.INC here and you can find other
;  versions of GDT.INC in various .ZIP files at
;  http://www.execpc.com/~geezer/johnfine/index.htm
;  The versions that require JLOC support some features that the nonJLOC
;  versions can't support.
;_____________________________________________________________________________
;
; The start_gdt macro marks the beginning of a GDT and uses the first 8 bytes
; of the GDT as a psuedo-descriptor for the LGDT instruction.
;_____________________________________________________________________________
;
; The end_gdt macro marks the end of a GDT and computes the limit value.
;_____________________________________________________________________________
;
; The desc macro pieces together a segment descriptor.
;
; SLCTR  desc  OFFSET, SELECTOR, ATTRIB      ;For gate descriptors
; SLCTR  desc  BASE, LIMIT, ATTRIB           ;For all other descriptors
;
;  SLCTR    (optional) is a label.  Unlike an ordinary label, it will not be
;           defined as the address of the descriptor.  Instead it will be
;           defined as the selector used to access the descriptor.
;  OFFSET   is the offset of the routine pointed to by a gate.
;  SELECTOR is the selector of the routine pointed to by a gate.
;  BASE     is the full 32 bit base address of the segment
;  LIMIT    is one less than the segment length in 1 or 4K byte units
;  ATTRIB   the sum of all the "D_" equates which apply (for call gates, you
;           also add the "parameter dword count" to ATTRIB).
;_____________________________________________________________________________

;Each descriptor should have exactly one of next 8 codes to define the type of
;descriptor (see attribute bit numbering)
D_LDT           EQU      200h   ;LDT segment
D_TASK          EQU      500h   ;Task gate
D_TSS           EQU      900h   ;TSS
D_CALL          EQU     0C00h   ;386 call gate
D_INT           EQU     0E00h   ;386 interrupt gate
D_TRAP          EQU     0F00h   ;386 trap gate
D_DATA          EQU     1000h   ;Data segment
D_CODE          EQU     1800h   ;Code segment

;Descriptors may include the following as appropriate:
D_DPL3          EQU     6000h   ;DPL3 or mask for DPL
D_DPL2          EQU     4000h
D_DPL1          EQU     2000h
D_DPL0          EQU     0000h
D_PRESENT       EQU     8000h   ;Present
D_NOT_PRESENT   EQU     8000h   ;Not Present
                                ;Note, the PRESENT bit is set by default
                                ;Include NOT_PRESENT to turn it off
                                ;Do not specify D_PRESENT

;Segment descriptors (not gates) may include:
D_ACC           EQU      100h   ;Accessed (Data or Code)

D_WRITE         EQU      200h   ;Writable (Data segments only)
D_READ          EQU      200h   ;Readable (Code segments only)
D_BUSY          EQU      200h   ;Busy (TSS only)

D_EXDOWN        EQU      400h   ;Expand down (Data segments only)
D_CONFORM       EQU      400h   ;Conforming (Code segments only)

D_BIG           EQU       40h   ;Default to 32 bit mode (USE32)
D_BIG_LIM       EQU       80h   ;Limit is in 4K units

%macro start_gdt 0
%push table
%$startoftable:
        dw      %$limitoftable
        dd      %$startoftable
        dw      0
%endmacro

%macro end_gdt 0
%$limitoftable  equ     $-%$startoftable-1
%pop
%endmacro

%macro desc 3
%ifid %00
%00 equ $-%$startoftable        ;Define selector
%endif
%if (%3) & (~(%3)>>2) & 0x400   ;Gate
        dw      %1
        dw      %2
        dw      (%3)+D_PRESENT
        dw      (%1) >> 16
%else                           ;Not a gate
        dw      %2
        dw      %1
        db      (%1) >> 16
        db      ((%3)+D_PRESENT) >> 8
        db      (%3) + ((%2) >> 16)
        db      (%1) >> 24
%endif
%endmacro

;-----------------------------------------------------------------------------
;
;  A gate is identified as any descriptor whose attributes has bit 10 set and
;  bit 12 clear.
;
;  For a gate, the following rearrangement occurs:
;
;  subField                Final location
;  ------------------      --------------
;  Selector[0..15]             16..31
;  Minor attribute bits        32..39
;  Major attribute bits        40..47
;  Offset[0..15]                0..15
;  Offset[16..31]              48..63
;
;  For non-gates the following rearrangement occurs:
;
;  subField                Final location
;  ------------------      --------------
;  Limit[0..15]                 0..15
;  Limit[16..19]               48..51
;  Minor attribute bits        52..55
;  Major attribute bits        40..47
;  Base[0..23]                 16..39
;  Base[24..31]                56..63
;
;  The last parameter to the desc macro contains all the attribute bits
;  combined.  It is generated by adding together the appropriate
;  D_ constants.  For all descriptors, it has the major attribute bits in D_
;  bits 8 to 15.  The minor attribute bits are in either D_ bits 0 to 7 or
;  bits 4 to 7 depending on the type of descriptor.
;_____________________________________________________________________________
```

First 8 bytes of the GDT

The CPU does not use the first 8 bytes of the GDT. I always use those bytes to hold the psuedo-descriptor required by the LGDT instruction. That means that the address of the psuedo-descriptor is the same as the address of the GDT itself.

My start_gdt macro creates the psuedo-descriptor in the first 8 bytes of the GDT.

The psuedo descriptor consists of a word giving the limit (length in bytes minus one) of the GDT, followed by a dword giving the GDT's linear address. (The fourth word is not used).