Introduction to Big Real Mode

Merck Hung <merckhung@gmail.com>, 洪豪謙

應朋友的要求，希望我花一點時間整理一下 x86 Big Real Mode 的文章.
另外也發現，身邊似乎有一些朋友也準備要開始從事 BIOS 方面之工作了.
感謝你們偶而會來逛一下我的 Blog.

雖然網路上已經有蠻多資料了，不過今天我打算從 Intel 64 and IA32 Architecture Software Developer's Manual (後面簡稱 SDM)，以及 x86 Processor 的角度，來解說如何打開 Big Real Mode.

我將會花時間解說 Intel SDM 內的資料，最後才丟一段 Code 給你.
例如說明什麼是 Big Real Mode，位址空間的差異, A20, Segment Descriptor 等細節.

而我會另外找時間再把 Protected Mode + Paging + PSE (存取大於 4GB Memory) 寫完.
感謝囉!

Big Real Mode 定義

Big Real Mode 是一個有趣的 x86 Processor Mode, 正好處在 32bit protected mode 與 16bit real mode 中間.

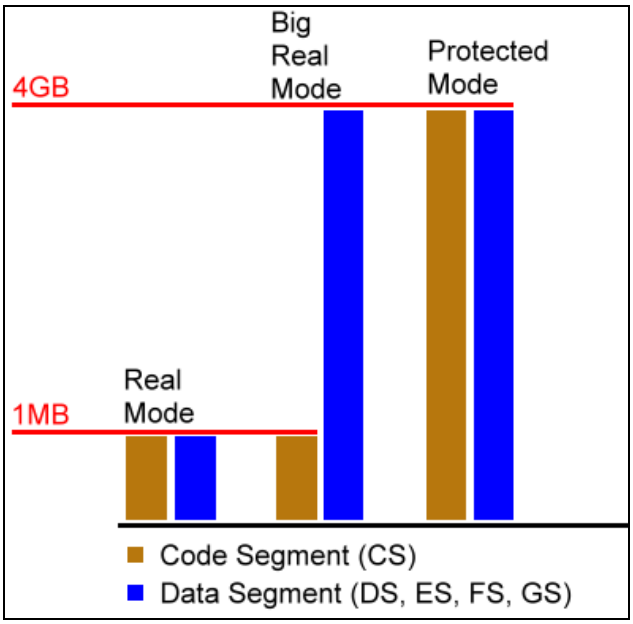

Fig.1, Real Mode, Big Real Mode, 與 Protected Mode 比較圖

以上圖來說, Real Mode 的 Code 與 Data Segment 的最大範圍均爲 1MB (16bit).

當 CPU 進入 Protected Mode 之後, Code 與 Data Segment 的最大範圍將可以達到 4GB.

而 Big Real Mode 很神奇的是, Code Segment 維持原來 1MB (16bit) 的限制, 但 Data Segment 卻可以存取到整個 4GB 的空間.

因爲一般 real mode 只有辦法存取到 1MB Memory, 然而以現在的電腦配備來說, 動輒 1GB, 2GB, 或 4GB 的記憶體大小. 如果 BIOS 單純僅運作在 real mode 內, 那樣根本完全無法存取到你所安裝的所有記憶體.

而一般 OS, 如 Windows, Linux 等, 皆是以打開 protected mode 來存取到整個 4GB 的 Memory Space.

但因爲 BIOS Code 大部分都是以 real mode 撰寫的, 所以當有那個需求要存取 1MB 以上的記憶體時, big real mode 就會是一個不錯的選擇.

雖然 protected mode 也可以達成存取 4GB 的目的, 但如果程式本身所執行的環境是 DOS 或 BIOS Code 的話, 打開 protected mode 反而會造成麻煩. 例如你會無法呼叫原來寫給 real mode 呼叫的一些 routines, 或是必須頻頻不斷的在兩個 mode 之間切換.

Address Space 差異

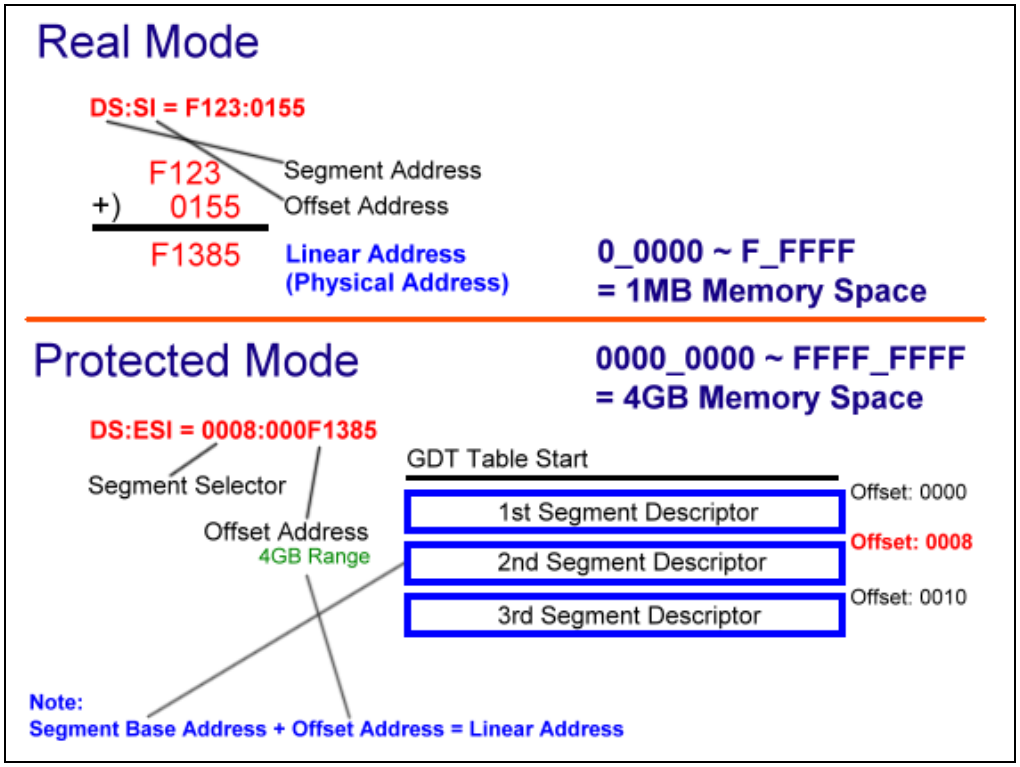

Fig.2, Real Mode 與 Protected Mode 位址空間之差異圖

在 real mode 中, CPU 最後實際存取 Memory 的位置 (Physical Address 或 Linear Address), 是透過 Segment Address 向左 Shift 4 bits, 然後加上 Offset Address 來計算.

故 real mode 的 address space range 是從 0_0000 ~ F_FFFF, 也就是只能組合出 1MB 的空間.

而在 protected mode (或 big real mode) 中, 原 real mode 中的 “segment register” 換了個名稱與定義, 改稱作 “segment selector”. 而這個 “segment selector” 將不是直接填寫 Address, 而改爲載入 Segment Descriptor 在 GDT Table 裡面的 Offset Address.

而 Segment Descriptor 內將會含有 Base Address 這個欄位, 用以說明 Segment 起始位址. 故將 Segment Descriptor 內之 Base Address, 加上 Offset Address 後, 便是實際 CPU 所將存取的 Memory 位置 (Physical Address 或 Linear Address).

Enable 與 Disable 流程

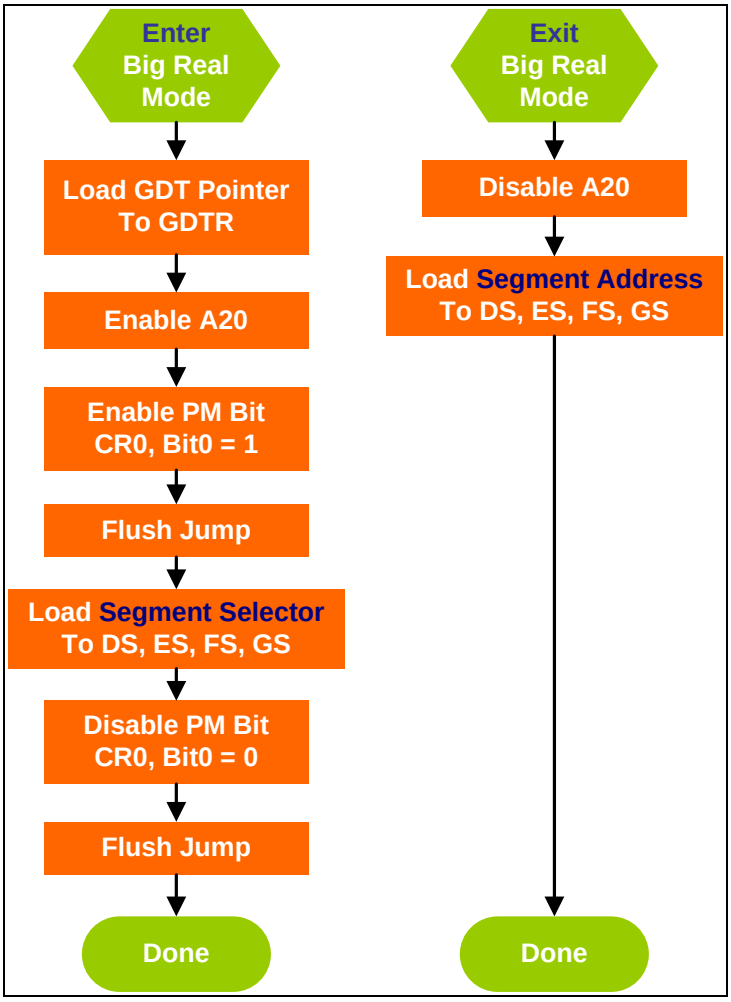

Fig.3, Enable 與 Disable Big Real Mode 的流程圖

打開 Big Real Mode 的程序:

1. 設定 Global Descriptor Table (GDT)
2. 載入 GDT Pointer 的 Physical Address 到 GDTR Register
3. 打開 A20
4. Enable Protected Mode Bit, CR0 Bit 0 = 1, 並作一個 Flush Jump
5. 將 4GB 的 Data Segment Selector 載入 DS, ES, FS, GS
6. Disable Protected Mode Bit, CR0 Bit 0 = 0
7. 作一個 Far Jump Flush

關閉 Big Real Mode 的程序:

1. 關閉 A20
2. 將 DS, ES, FS, GS 的內容設定 (或從 Stack pop 回來) 爲 Real Mode Segment Offset 即可

A20 開關

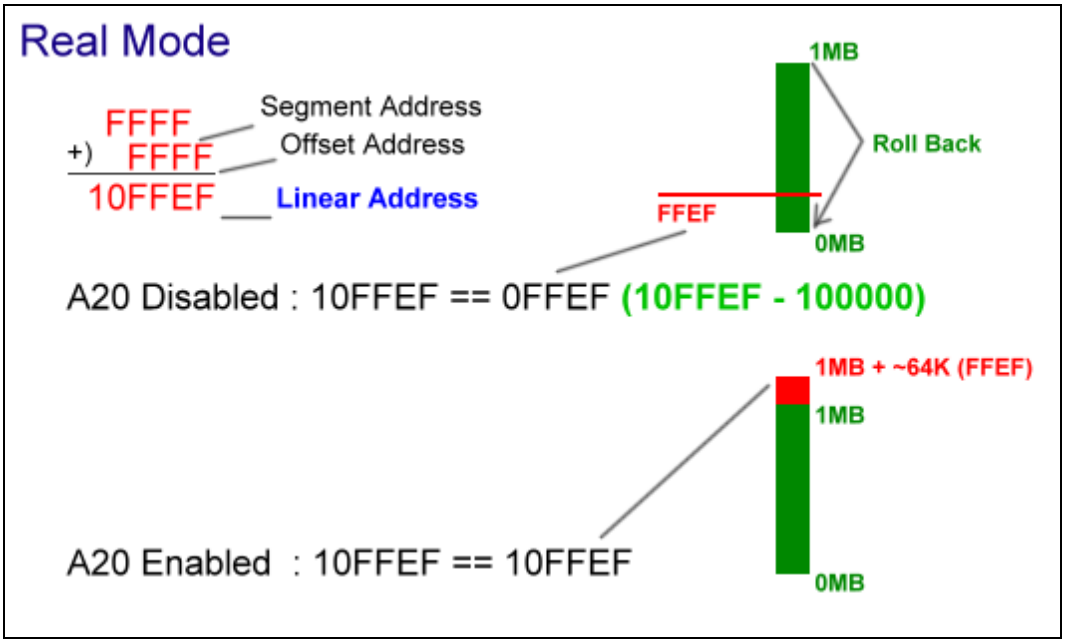

Fig.4, 在 Real Mode 中, A20 Enabled 所多出之約 64k Address Space

在早期 8086 CPU 位址線只有 20 條, 所以造就了 Segment:Offset 這樣的存取方式. 但其實大家可以注意到, FFFF:000F = FFFFF (1MB), 其中 Offset Address 還沒填到最大值 FFFF. 但因爲 8086 只有 20 隻腳位, 所以超過 FFFFF (1MB) 的搭配, CPU 將會回繞回 0MB 起算.

而後來的 CPU 的位址線增加, 如 286 有 24 隻腳位, 386 以上有 32 隻腳位. 但爲了維持 PC 架構的相容性, 所以增加了 A20 這樣的開關, 在 A20 關閉時讓超過 1MB 的定址做回繞到 1MB 的動作. 但只要 A20 打開後, 超過 1MB 將不會有回繞動作.

而在 real mode 中, 因爲使用 Segment:Offset 的存取方式, 所以打開了 A20 後, 大約只能增加 64k 左右的空間 (10000 ~ 10FFEF).

但也因爲 A20 開關牽涉到位址線回繞的問題, 所以當我們打算進入 protected mode (使用全部 32 隻位址線) 之前, 打開 A20 開關也是一個重要的課題.

GDT 與 Segment Descriptor

有別於 16bit real mode 將 **Segment Address** 直接載入 DS, ES, FS, GS (Data Segment Registers) 的方式.

在 protected mode 內, DS, ES, FS, GS 轉換了一個名稱, **Segment Selector**.

因爲 DS, ES, FS, GS 還是維持原來 16bit 的大小, 並非像 AX, BX,……等 16bit 暫存器, 推出 32bit 的版本, 如: EAX, EBX,……等.

但因爲 32bit protected mode 存取 Data 的方法還是維持 DS:XX, ES:XX, FS:XX, GS:XX 的方式, 所以 Intel 提供存取 4GB 的新方法, 而這個方式就是利用 Segment Descriptor 與 Segment Selector.

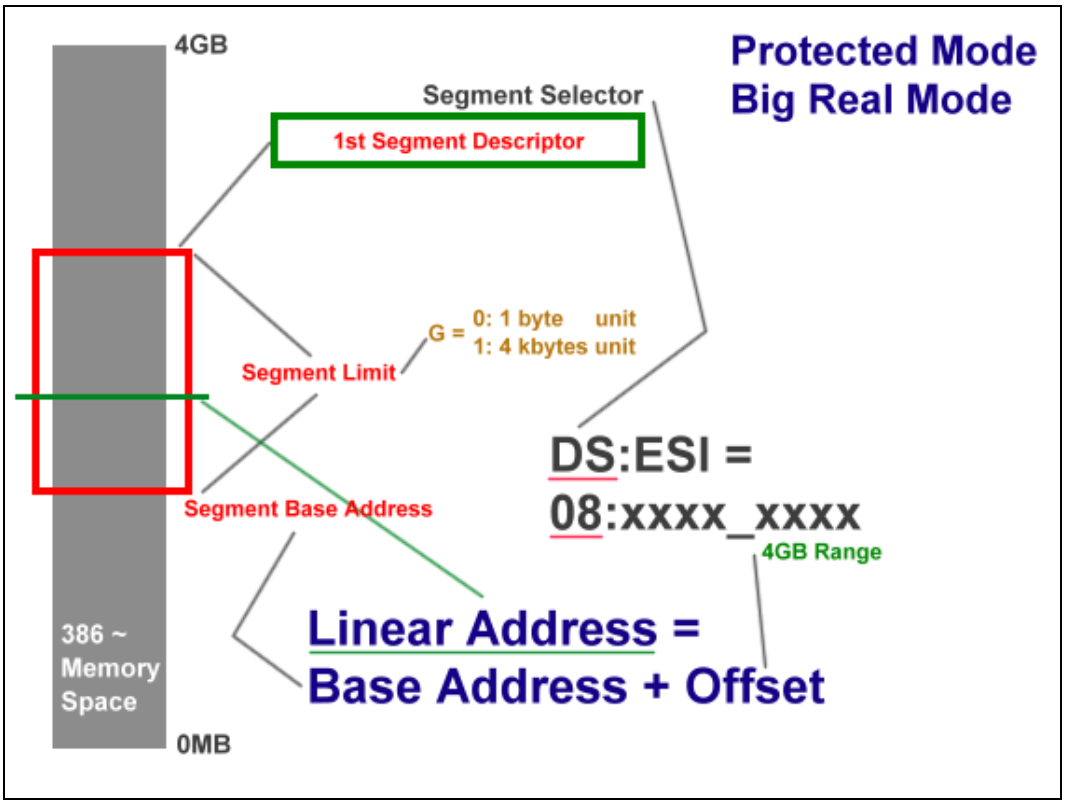

Fig.5, Segment Descriptor 定義 Segment 的與位址的對應關係

一般爲了方便使用, Segment Base Address 通常會被設爲 0, 而 Segment Limit 會設爲 4GB. 這不是我所創造的慣例, 而是現在的作業系統都是這樣使用. 甚至現在支援 64bit 的 CPU, 在進入 64bit long mode 後, Segment 這樣的 feature 已經乾脆被捨棄了 (因爲過去大家都直接開 4GB, 等同於不使用 Segment 這個特性).

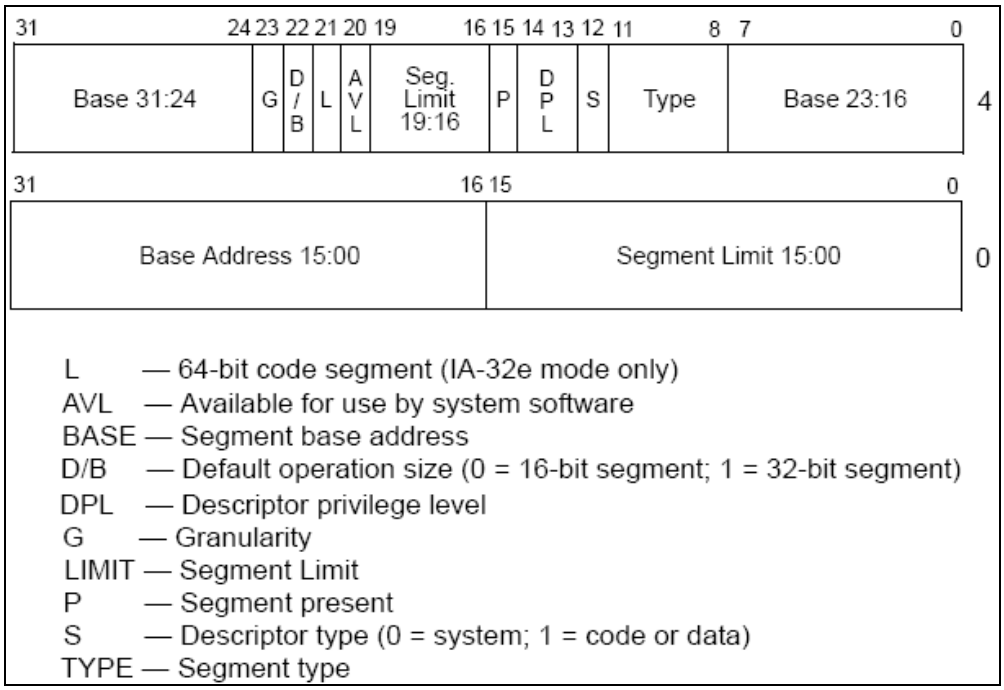

Fig.6, Intel SDM Vol.3 3-13, Segment Descriptor 各欄位說明

由上圖所知, Segment Descriptor 是一個 8 bytes 的資料結構.

其中 Base Address (31:00, 32bit), 將會用來 “說明”, 這個 Segment 的起始位址.

而 Segment Limit (19:00, 20bit), 將會用來 “說明”, 從起始位址開始算起的 “長度”, 是屬於這個 Segment.

而 G 欄位用來決定 Segment Limit 的單位, 0 爲 1 bytes, 1 爲 4k. 因爲 Segment Limit 只有 20bit, 所以當 G=0 (1 bytes) 時, 最大只能涵蓋到 1MB. 但當 G=1 (4 kbytes) 時, 最大就能涵蓋到 4GB.

Type 是一個 4bit 欄位, 總共有 16 種 Type 可供填寫, 主要的分類是 Code 或 Data, 細項分類將於 protected mode 一文中說明.

P 欄位讓 OS 用來表示這個 Segment 是否被 Swap Out 到硬碟上, 而沒有實際在記憶體上.

S 欄位, 爲 0 時表示 System Segment, 爲 1 時表示 Code 或 Data Segment.

```
;###############################################################################
; Globel Descriptor Table (GDT)
;
align 16
GDT_TABLE:

    ; NULL segment
    DW   0, 0, 0, 0                    ; 四個 WORD 等於 8 bytes

    ; Data segment, read/write
FLAT_DATA_SEG          EQU       $ - GDT_TABLE
    DW   0FFFFh
    DW   0
    DW   9200h
    DW   00CFh

GDT_SIZE               EQU       $ - GDT_TABLE

;
; GDT Pointer
;
GDT_POINTER:
    DW   GDT_SIZE - 1
    DW   0                  ;
    DW   0                  ; GDT base address
```

我們來看一個實際的範例, 設定 GDT 的主要重點是:

1. **GDT 起頭要對齊 16 bytes (align 16)**
2. **第一個 Segment Descriptor 務必要爲 NULL Segment**
3. **最後需要填寫 GDT Pointer, Pointer 的 Linear Address 將用於載入 CPU 的 GDTR 暫存器.**
4. **整個 GDT 的 Size 及 GDT 的起始 Linear Address 需填入 GDT Pointer.**

而以 Big Real Mode 來說, 我們只需設定一個 **Data Segment Descriptor**, 而 **FLAT_DATA_SEG** 就是所謂的 **Segment Selector (此例中爲 08h)**, 將會用於載入 DS, ES, GS, FS 暫存器.

00CF9200h																															
Base (31:24)								G	D	L	A	Lim (19:16)				P	DPL		S	Type				Base (23:16)							
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
0				0				C				F				9				2				0				0			
0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0

0000FFFFh																															
Base Address (15:00)																Segment Limit (15:00)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fig.7, 範例中各欄位 Bit 標示

Segment Base = 0000_0000h

G = 1, 單位 = 4 kbytes
Segment Limit (00:19) = F_FFFFh
Segment Limit = 4GB (FFFF_FFFFh)

D/B = 1, 32bit Segment (爲存取 4GB, 故爲 32bit 區段)
DPL = 0, Kernel Segment, Ring 0 (屬於 Kernel Ring 0, 最大權限)
Type = 2, Data Read/Write Segment (屬於資料, 可讀可寫區段)
AVL = 0, Reserved (保留)
L = 0, Reserved (保留)
P = 1, Present in Physical Memory (預設 1)
S = 1, Code or Data Segment (程式或資料區段)

Sample Code – Enter and Leave Big Real Mode

```
LIBFLAT SEGMENT USE16 'CODE'


;################################################################################
; __enter_flat_mode – Enter Big Real Mode, 進入 Big Real Mode
;
; Input:
;     None
;
; Output:
;     None
;
; Modified:
;     All possible
;
__enter_flat_mode PROC FAR PUBLIC


    ; Convert GDT base physical address to linear one
    ; 將 GDT 的 Base Address, 由 Segment:Offset 轉爲 Linear Address
    xor       eax, eax                          ; 另 EAX = 0 (32bit)
    mov       ax, cs                            ; 將 Code Segment Address 放入 AX (16bit)
    shl       eax, 4                            ; EAX (32bit) 向左位移 4 的 bits
    add       eax, OFFSET GDT_TABLE             ; 加法演算, 加上 GDT 在 Code Segment 內的 Offset
    mov       dword ptr GDT_POINTER+2, eax      ; 將 EAX (32bit) 的結果值, 寫入 GDT Pointer (本來爲 0)


    ; Save original GDT and Load new one
    ; 利用 LGDT 指令, 將 GDT Pointer 載入 CPU 的 GDTR 暫存器
    lgdt      fword ptr GDT_POINTER


    ; Disable interrupt
    ; 關閉 CPU 中斷
    cli


    ; Enable A20
    ; 打開 A20
    in        al, 92h                           ; 從 Keyboard Controller (IO Port 92h) 讀入
    or        al, 02h                           ; 將 Bit 1, A20 設爲 1
    out       92h, al                           ; 寫回 Keyboard Controller


    ; Enable protected mode
    ; 打開 CR0 Bit 0, 也就是 protected mode
    mov       eax, cr0                          ; 先將 CR0 讀入 EAX
    or        eax, 01h                          ; 將 Bit 0, PM Flag 設定爲 1
    mov       cr0, eax                          ; 將 EAX 寫入 CR0
    jmp       @f                                ; Flush Jump, 讓 CPU 更新狀態

@@:                                             ; Flush Jump 到這裡
```

```
    ; Setup Segments
    ; 進入 protected mode 後, 開始將 Segment Selector 載入 DS 與 ES
    mov     ax, FLAT_DATA_SEG               ; 此例中 AX = 08h
    mov     ds, ax                          ; 將 DS 設爲 4GB Segment
    mov     es, ax                          ; 將 ES 設爲 4GB Segment


    ; Disbale protected mode
    ; 設定完 DS, ES 後, 就可以關閉 protected mode 了
    mov     eax, cr0                        ; 先將 CR0 讀入 EAX
    and     al, 0feh                        ; 將 Bit 0, PM Flag 設爲 0
    mov     cr0, eax                        ; 將 EAX 寫入 CR0


    ; Far jump to 16 bit FLAT mode
    ; 因爲 MASM 沒有 Far Jump 的指令, 所以我們直接利用 DB, DW 等 Macro 來寫 CPU 指令
    ; OpCode 部分是 EAh, 後面第一個 WORD 是 Offset Address, 第二個是 Segment Address
    DB      0eah                            ; Far Jump 指令的 OpCode
    DW      OFFSET @f                       ; 請組譯器幫我們填 Offset Address
    DW      SEG @f                          ; 請組譯器幫我們填 Segment Address


@@:                                         ; Far Jump 到這裡


    ret

__enter_flat_mode ENDP



;################################################################################
; __exit_flat_mode – Leave Big Real Mode, 離開 Big Real Mode
;
; Input:
;     None
;
; Output:
;     None
;
; Modified:
;     All possible
;
__exit_flat_mode PROC FAR PUBLIC

    ; Disable A20
    ; 關閉 A20
    in      al, 92h                         ; 從 Keyboard Controller 讀入
    and     al, not 02h                     ; 將 Bit 1, A20 設爲 0
    out     92h, al                         ; 寫回 Keyboard Controller

    ; Reenable interrupt
    ; 重新打開中斷
    sti

    ret

__exit_flat_mode ENDP
```

```
;#############################################################################
; Globel Descriptor Table (GDT)
;
align 16
GDT_TABLE:

      ; NULL segment
      DW   0, 0, 0, 0

      ; Data segment, read/write
FLAT_DATA_SEG          EQU       $ - GDT_TABLE
      DW   0ffffh
      DW   0
      DW   9200h
      DW   00cfh

GDT_SIZE                EQU       $ - GDT_TABLE

;
; GDT Pointer
;
GDT_POINTER:
      DW   GDT_SIZE - 1
      DW   0                    ;
      DW   0                    ; GDT base address


LIBFLAT ENDS
```

```
;###############################################################################
; enter_flat_mode -- Enter Big Real Mode, 進入 Big Real Mode
;
; Input:
;     None
;
; Output:
;     None
;
; Modified:
;     All possible
;
enter_flat_mode MACRO

      ; Save segment for FLAT exit
      ; 因爲我們會將 DS, ES 設定爲 4GB, 所以一般應用可以在進入前, 將原 DS, ES 推入 Stack
      push      ds
      push      es
      call      __enter_flat_mode

ENDM



;###############################################################################
; exit_flat_mode -- Leave Big Real Mode, 離開 Big Real Mode
;
; Input:
;     None
;
; Output:
;     None
;
; Modified:
;     All possible
;
exit_flat_mode MACRO


      ; Restore original segments
      ; 回存進入時所推入的 ES, DS, Segment Address 值, 來回復 real mode 64k.
      pop       es
      pop       ds
      call      __exit_flat_mode                    ; 關閉 A20, 打開中斷

ENDM
```

```
;-------------------------------------------------------------------------
; Code segment
;
_TEXT SEGMENT PARA USE16 'CODE'


libflat SEGMENT USE16 PUBLIC

      EXTERN __enter_flat_mode:FAR
      EXTERN __exit_flat_mode:FAR
@CurSeg ENDS


;#########################################################################################
; MAIN procedure
;
MAIN PROC FAR PRIVATE


      ; Save for DOS return
      push      ds
      push      ax


      ASSUME    SS:STACK, DS:_DATA, CS:_TEXT, ES:_DATA
      mov       ax, _DATA
      mov       ds, ax
      mov       es, ax


;-------------------------------------------------------------------------
; Enter FLAT mode
;-------------------------------------------------------------------------
enter_flat_mode                         ; 進入 Big Real Mode
;-------------------------------------------------------------------------
; FLAT mode code start
;-------------------------------------------------------------------------


      ; 存取 Linear Address 12345678h, 讀入 Double Word 到 EAX
      mov       esi, 12345678h          ; 將 ESI 設爲 12345678h
      mov       eax, ds:esi             ; 將 Linear Address 12345678h 的內容, 讀入 EAX


;-------------------------------------------------------------------------
; Exit FLAT mode
;-------------------------------------------------------------------------
exit_flat_mode                          ; 離開 Big Real Mode
;-------------------------------------------------------------------------
; REAL mode code
;-------------------------------------------------------------------------


      ; Return to DOS
      ret


MAIN ENDP

_TEXT ENDS
```