處理器中的記憶體分段 (Memory Segmentation)

cousin007 · Follow
13 min read · Jul 7, 2020

前言

記憶體與處理器(CPU)的關係相當密切，CPU會直接存取記憶體中的程式碼與資源工作。這裡所指的記憶體為Main memory，也就是我們常見的RAM(Random Access Memory)，它就像CPU的工作桌，所有程序都要先載到記憶體才能執行，因此如何有效地定址記憶體成為了很重要的課題。

Intel由8086處理器開始引入了記憶體分段(Memory Segmentation)。由CPU中的段寄存器(Segment register)指示記憶體段落的基礎地址(Base address)，加上偏移寄存器(Offset register)的邏輯地址(Logical address)，計算後就可以得出目標的實體地址(Physical address)。透過Segmentation可以獲得以下好處：

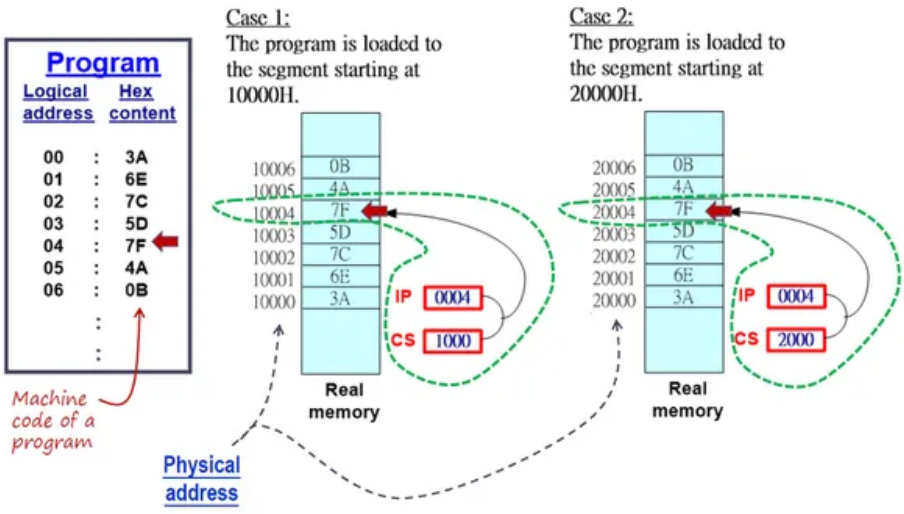
1. 定址比寄存器大的記憶體容量
2. 程式重新定位 (Program Relocation)

寄存器(Register)是記憶體架構中的頂端，讀取速度非常快，然而容量很少只能以位元來計算，擴充上也面對著高昂成本的問題。以8086為例， 8086的register size只有16 bits，直接儲存實體地址的話，只能定位2^{16} = 64 KB的記憶體。若透過Segment:Offset的方式生成地址的話，就能夠讀寫倍數數量的記憶體，用有限的資源取得更大的效益。

> *撇除成本之外，亦有不少研究說明，不斷擴充register size並不會帶來相應的效能增長，因此現在的處理器都會傾向增加register的數量多於提升register的容量。所以64位元的處理器很大可能會繼續陪伴我們一段長的時間*

另一方面，我們現在用的電腦叫作通用計算機，意即我們可以依自身需要，在電腦上運行不同的程式。面對這種不確定性與多程序的環境，編寫這些應用程式時，不能夠以physical address作為依據，只能夠以logical address編寫，留待執行時由系統負責地址間的轉換。利用segmentation之類的記憶體管理方

法，每次執行程式時，系統都會劃出一塊記憶體予要執行的程序，起始地址記錄為segment address，當中的程式碼就會以偏移量(Offset)讀取。所以，無論程式區塊移動到任何位置都可以用同樣的offset存取，令程式碼具備可移植性。



以0x4 offset存取相同的程式碼

接下來將會探討一下x86架構下的兩種Segmentation mode：

1. 真實模式(Real mode)
2. 與保護模式(Protected mode)

Protected mode是Real mode的升級版，然而基於向後兼容，兩者會視情況互相切換，所以並不屬於取代的關係。

. . .

Real Mode

Real mode由8086 CPU開始引入，是80286前的唯一模式，時至今天，x86架構CPU在開機時都會先使用real mode讀取BIOS等開機程式然後才切換至其他定址模式。

Real mode採用Segment:Offset的方式計算出實體地址，每次定址都會用到一組對應的Segment register和Offset register，而某些register是有預設功能的，例如：

1. 程式碼會用到Code Segment(CS)和Instruction Pointer(IP)
2. Stack會用到Stack Segment(SS)和Stack Pointer(SP)

> *在80286或之前的CPU只能夠有4個Active segment (CS, DS, SS, ES)。在80386之後就增至6個 (新增FS, GS，這兩個就沒有特別預設用途)。*

現在我們知道real mode的組成部分是一組對應的register，接下來要知道它們的大小。8086是顆16位元的處理器，所以寄存器大小就是16 bits。

- Segment register: 16 bits
- Offset register: 16 bits
- Address bus: 20 bits

這裡看到8086的地址總線(Address bus)是有20 bits，意味著我們需要做一些計算才能得到實體地址。先回憶一下前文關於segment與offset的定義：

1. Segment為記憶體區段，Segment register儲存的是這個區段的初始地址
2. Offset為目標偏移量，是與初始地址的相對位置

計算方法

簡單來說就是「Segment + Offset = Physical address」然而，我們現在需要的是20 bits的地址，直接相加的話，是沒辦法把16 bits變成20 bits的。而Real mode的做法簡單粗暴，將Segment address左移4 位元就可以了。這裡來看一個例子會更容易理解：

```
假設CS = 0x1000, IP = 0xFFFF:
CS:  10000 <-1位Hex代表4位Bin，所以只補一個0
IP: + FFFF
    ------
     1FFFF
```

一般來說，二進制的計算都會化成十六進制，因為這樣比較容易閱讀。Real mode的機制相對簡單，動的只有Segment address，比較學術的描述是把segment address乘以16。但以應用來說，只需要記住補一個0就可以了。例子中的答案`1FFFF`就是目標的physical address。

相關補充

1. Real mode最終會計算出20位元長的實體地址。因此Real mode下的記憶體最大定址數為2^{20} = 1 MB。而因為Real mode的關係，記憶體中的頭1 MB會被特別稱為Real memory / Conventional memory。

2. 由於Segment address固定左移4 bits，因此起始地址必定為16的倍數(eg. 0x00010, 0x00020...)
   - 這個區間稱為paragraph
   - 每個能夠被16整除的地址叫做paragraph boundary
   - Paragraph size = 16。

3. Real mode並沒有明確表示每個Segment的大小，不過從offset register的物理限制來看，16位元最大數值為0xFFFF，得出每個segment的上限為2^{16} = 64KB。

4. Segment之間沒有屬地概念，只要有有效的segment address + offset就能夠讀寫記憶體，所以不同segment之間會互相重疊。而且Real mode在設計上並沒有硬體上的讀寫保護機制。在現代的角度看來，是不理想的設計。

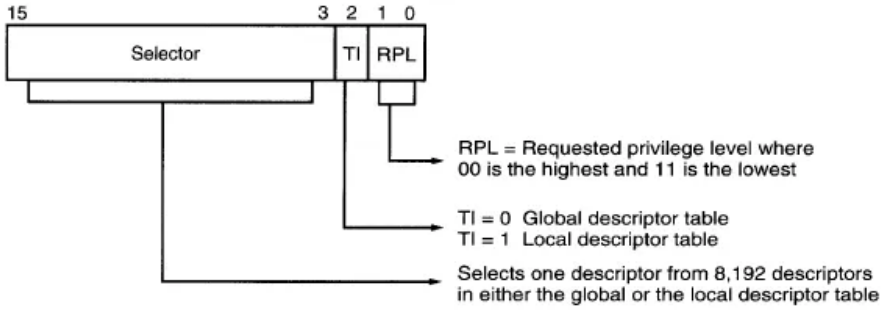
. . .

Protected mode

Protected mode由80286開始引入，貫通整個x86架構。顧名思義，它對記憶體是有硬體上的讀寫保護。與Real mode不同，Protected mode的Segment register不再直接儲存實體地址，取而代之是儲存一個選擇器(Selector)，透過查表找到對應的描述符(Descriptor)，而descriptor則是記錄了相關區段的資訊。

然而整體Segment:Offset的定址方向沒有改變，只是在Segment的查找上需要多花功夫。Protected mode的組成部分有點多，接下來會先介紹一下Protected mode的各個組成部分。

1. Selector



Selector Format (16 bits)

一個16位元的Selector分成三部分：

1. 13 bits的選擇索引
2. 1 bit的Table Indicator (TI)
3. 2 bits的權限等級(RPL)

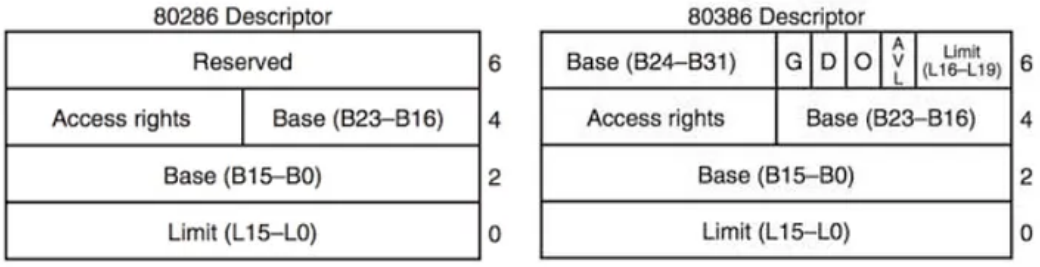
Selector其中的13 bits會被用作在描述符表(Descriptor table)中選擇單個Descriptor之用。例如顯示為1，則代表選擇第一個，2代表第二個，如此類推。因此最大選取值為2^{13} = 8192個。

Protected mode的運作模式下會有兩款descriptor table，分別是全域表(GDT)與局域表(LDT)，若TI bit顯示為0則選擇GDT、1則為LDT。

RPL是標示存取權限，00為最高權限等級，11為最低。

> *注意：Descriptor 0稱為Null Descriptor，若Selector存取Descriptor 0會觸發exception*

2. Descriptor



Descriptor format (64 bits)

Descriptor主要分成兩種格式，一種是286用，另一種是386跟往後的CPU用。兩種格式幾乎是一樣的，主要分別在於基址(Base)的擴充與Segment大小(Limit)的變化。

Descriptor定義了segment的各種資訊，我們主要看的是Base跟Limit。Base與real mode中的segment register一樣，儲存了Segment的實體起始地址，分別是Base的地址不需要再作後期加工。

與real mode不同，protected mode會明確定義一個segment的大小。286的Limit比較簡單，16 bits的Limit標示一個segment的大小可以從1 Byte至64 KB。

但如果是386的Descriptor的話，就必須要特別留意，除了Limit延長至20 bits以外，還有一個因素會影響segment size。386 Descriptor有一支叫「G bit」的Flag，G為Granularity，中文翻譯為粒度，大概取其粒子粗幼度的意思。若G bit = 1時，Limit就會倍大4 KB倍（實際應用時可直接在Limit後邊補上0xFFF）。換言之，386的Limit是浮動的，可以是1 Byte~1 MB也可以是4 KB~4 GB。

	80286	80386
Base	24bit	32bit
Limit	16bit	20bit
Segment size	1B~64KB	1B~1MB / 4K~4GB
Maximum memory	16MB	4GB

Descriptor的基礎資料

3. GDT / LDT

Global / Local Descriptor Table，是一張集合了descriptor供selector選擇的表，分為全域(Global)與局域(Local)兩種。GDT是一張全域表，整個系統只會有一張，會在Protected mode起始時建立。而LDT是局域表，在系統中可以存在多張，由不同任務或程序建立。值得注意的是，LDT本身同樣是一個memory segment，所以也會有代表這個segment的descriptor，它會儲存在GDT之中。

> *配合selector的選取上限，每張表的descriptor數量最多為8192個。*

4. GDTR / LDTR

最後要介紹的是GDT與LDT的register，GDTR會直接儲存GDT所在的實體地址與Limit。如前段所說，GDT會在Protected mode初始時分配位置，因此GDTR就會在這個時間寫入。GDTR是Protected mode查找流程的入口。

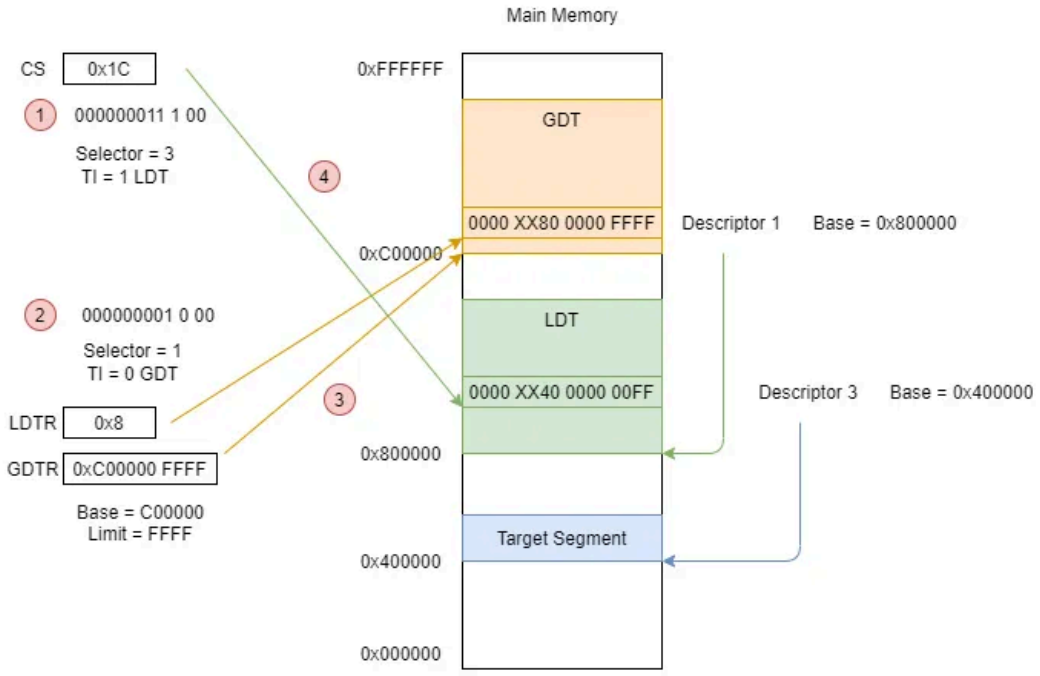
	Base	Limit	Total
80286	24bits	16bits	40bits
80386	32bits	16bits	48bits

GDTR Format

而LDTR就需要留意，它不是像GDTR一樣儲存LDT的實體地址，它儲存的是一個Selector，正如前文所述，LDT也是一個Memory segment。如果需要訪問LDT的話，系統會拿著這個selector查找GDT，才能找到LDT的實際位置。

訪問流程

Protected mode的訪問流程比較複雜，我們來直接看例子。下面是80286的訪問例子，請注意descriptor的格式與GDTR的長度。



Protected mode的LDT查詢流程

1. 假設系統現在需要存取Code segment的記憶體區段，我們知道這是一個16 bits的Selector，先把`0x1C`拆成二進制看看，看到selector = 3而且TI bit = 1，所以現在要求的是「**LDT中的第3個descriptor**」，因為目前還不知道LDT的實際地址，因此我們要先找到LDT的所在地。
2. 來到LDTR，我們一樣先把它拆開，對照selector format，知道`0x8`指向「**GDT的第1個descriptor**」，進一步我們要去找GDT的地址。

3. 透過GDTR知道GDT的實體地址為`0xC00000`，並找到第1個descriptor。在這個例子，我們只需要descriptor當中的Base就夠了 (8~32 bits)。所以`0x800000`就是LDT的實際地址了。

4. 現在我們知道LDT的physical address，回到CS的Selector，它要求的是「LDT中的第3個descriptor」。同理，我們擷取descriptor當中的Base，得出目標segment是在`0x400000`，大功告成！

> *為了簡化概念 (其實是懶得畫圖)，例子直接給出對應的descriptor。現實是可能需要大家算出來。首先每個descriptor長度為8 Bytes，由descriptor table的基址開始數每+8為一個descriptor。如上述例子，GDT第1個descriptor address為* `0xC00008` *、LDT第3個descriptor address為* `0x800018` *。*

相關補充

1. Segment size

計算Segment size的時候要小心，雖然在解釋descriptor中的Limit時會說它是定義segment的大小，但limit儲存的其實是「Segment的最終offset address」因此，Segment size應該是`Limit + 1` (把0也算進去)。

例如，Limit是0xFF，Segment size就等於`0xFF + 1 = 0x100`

這個情況就好像Class C IP有256個，但最終address是255一樣。

2. Segment boundary

Protected mode的Segment address能夠開始在任何地方，不像Real mode局限在地址的16倍數，因此Protected mode沒有了paragraph的概念。

3. 權限

我在這邊沒有太多著墨在Protected mode的Access right上，這是因為我也不太懂(汗，而且在descriptor中也有相當多的特別參數會影響系統的運作。但由於我上課程只是初級理論課，或許未來了解到的時候再作補充吧。

. . .

後記

Memory segmentation其實只是其中一種管理記憶體的方法，現實上不同架構的處理器亦會採用不同的記憶體管理方式。我們現在的主流x86_64處理器基於

向後兼容的關係，是有保留作兼容模式使用，但在64位元模式下，segmentation的重要性幾乎可以忽略了。

接下來，下一篇應該會講一下由80386開始引入，更強大的記憶體管理模式**分頁(Paging)**。

最後，其實我也是在學習途中，如果筆記中有錯誤的地方，希望各位能夠幫忙指出，謝謝大家！

Cpu 真實模式 保護模式 記憶體 分段