Computer Organization and Assembly Languages Final Project

The comparison between 32-bit and 64-bit microprocessor architecture

B94902029 郭彥伶
B94902055 吳明瑾

References

1. Intel® 64 and IA-32 Architectures Software Developer's Manual
   Volume 1: Basic Architecture
2. AMD64 Architecture Programmer's Manual Volume 1: Application Programming
3. Wikipedia-X86-64: http://en.wikipedia.org/wiki/X86-64
4. Wikipedia-IA-32: http://en.wikipedia.org/wiki/IA-32

Preface

本學期在計算機組織與組合語言這門課中，老師介紹了 32-bit 的 architecture 還有 x86 assembly language，但卻甚少提及 64-bit 的 architecture，因此，在想要了解 64-bit 的 architecture 及其在 media programming 方面與 32-bit 的差異之情況下，我們決定了這個主題。

而這份 project 主要比較了 32-bit 和 64-bit 的 microprocessor architecture，內容則涵蓋了兩種 architecture 中 memory、registers、instructions 及 media programming 方面(即 MMX, SSE, SSE2, SSE3)的對照。

Modes of Operation

32-bit

(1) Protected mode

- Multitasking
- Protected memory：讓每一個程式不會使用到其他程式所用的 memory，避免兩個程式相衝。
- Virtual memory：讓程式可以 access 的 memory space 達到 4GB。

(2) Real-address mode

- 會在 DOS operating system 使用，程式可以任意存取 memory。
- 程式最多只能存取 1MB(20-bit)的 memory space。
- 在現在的 OS 中 (Ex. Windows, Linux)，只有開機的時候才會屬於 Real-address mode。

64-bit

(1) Compatibility mode

- 此模式下，64-bit 的 OS 能夠在不需要重新編譯的情況下執行 16-bit 和 32-bit 的 x86 應用程式。
- 與 32-bit 中的 Protected mode 相似，所以只能存取 linear-address space 中的前 4GB。

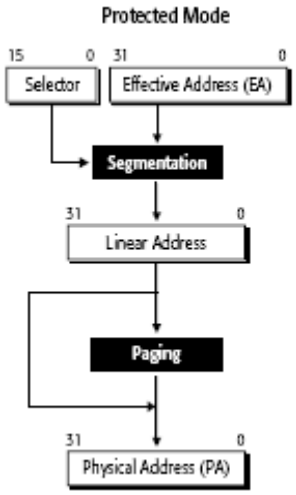
(2) 64-bit mode

- 只有在使用 64-bit OS 的前提下才會使用到 64-bit mode。
- 只有 64-bit 的 OS 在執行程式時，能夠存取最多到 64-bit 的 linear address space。
- 將 general purpose registers 和 SIMD extension registers，從 8 個增加到 16 個，而且 general purpose registers 擴充成 64-bit。
- 重新定義 opcode，使 operand size 可以支援到 64-bit 的 operand。

Memory Management

32-bit

(1) Protected Mode:

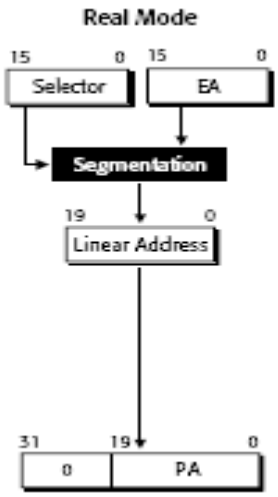


Protected mode 可支援 16-bit 跟 32-bit 的程式。

使用 table-based memory segmentation, paging 以及 privilege-checking，segmentation function 把 32-bit 的 effective address 和 16-bit 的 selector (包含 2-bit 的 Required Privilege Level, 1-bit 的 Table Indicator, 13-bit 的 index)，產生出一個 32-bit 的 linear address(爲 16K 個 memory segments 之一)，每一個空間皆可達到 4GB。

Paging 是非必須的，32-bit 的 physical address 可以靠 paging 產生，或者是 linear address 在沒有改變的情況下直接當成 physical address 使用。

(2) Real Mode



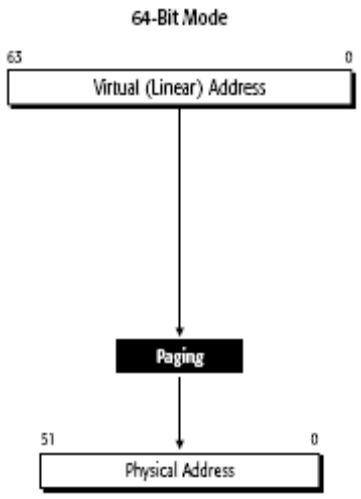
Real mode 支援 16-bit 的程式使用 register-based shift-and-add segmentation，但是不支援 paging。

對 16-bit 的 effective addresses 做 zero-extended 加上往左 shift 4 bit 的 16-bit segment-base address (內含特定實體記憶體位址的 address)，產生一個 20-bit 的 linear address。

此 linear address 再做 zero-extension extend 成 32-bit 的 physical address，但其最多只能存取 1MB 的 memory space。

64-bit

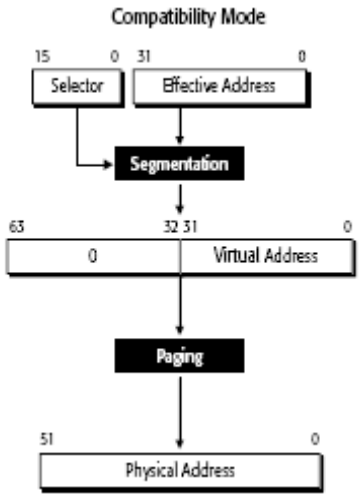
(1) 64-bit Mode



Segments以及real address在64-bit mode中是無法使用的。

在64-bit mode中，程式產生一個可以達到64 bits大小的virtual addresses，然後透過PAE(Page Address Extensions) paging轉換到physical address，但真正能執行的空間並沒有到達64-bit。

(2) Compatibility Mode



由程式所產生的 16-bit 或 32-bit 的 effective addresses 跟此程式的 segments 結合，產生一個 32-bit(最大可以 zero-extended 到 64-bit)的 virtual(linear) address，之後 paging 的方式與 64-bit mode 相同。

註 1：paging 將 virtual addresses 轉換爲 physical addresses。

註 2：segmentation selector 加上 effective address 也可以稱做是 logical address 或者是 far pointer。

註 3：virtual address 又稱做 linear address。

Basic Program Execution Registers

1. General purpose registers

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L - R15L
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

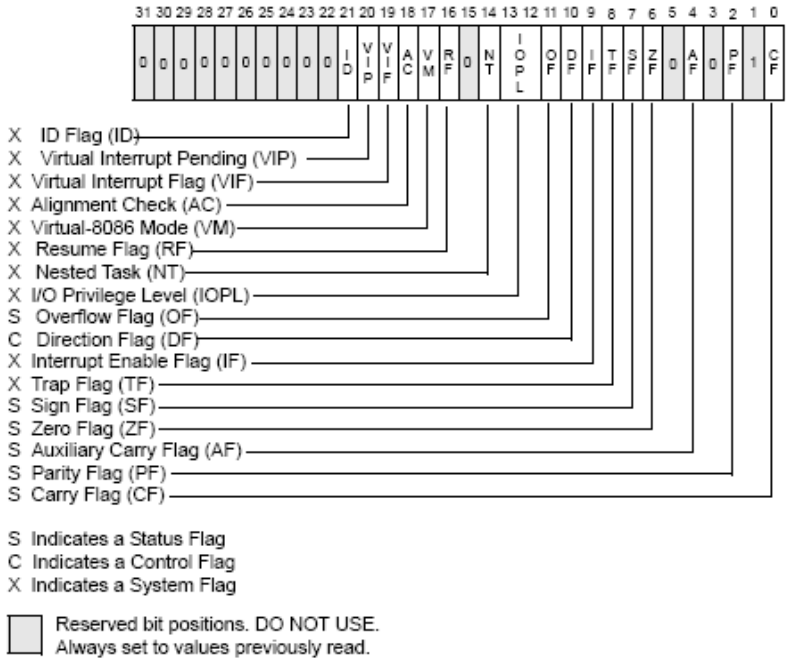
以下是這些register特別的用途:

- EAX — Accumulator for operands and results data
- EBX — Pointer to data in the DS segment
- ECX — Counter for string and loop operations
- EDX — I/O pointer
- ESI — Pointer to data in the segment pointed to by the DS register; source pointer for string operations
- EDI — Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations
- ESP — Stack pointer (in the SS segment)
- EBP — Pointer to data on the stack (in the SS segment)

2. **Segment registers**

   CS(code segment), DS, ES, FS, GS(data segments), SS(stack segment)內含 16-bit 的 segment selector，在 64-bit mode 中，CS, DS, ES, SS 被視爲 0，這樣使 code, data, stack 成爲 flat address space，FS, GS 則爲 exceptions。

3. **EFLAGS (program status and control) register**



   在 64-bit mode 中，EFLAGS 被擴充到 64 bits 稱爲 RFLAGS，前面的 32-bit 被保留，後面的 32-bit 就是原來的 EFLAGS。

4. **EIP (instruction pointer) register**

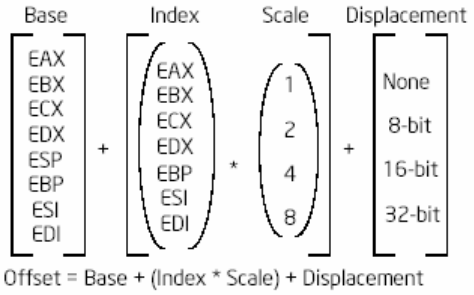
   - 此 register 包含了在現在這個 code segment 中下一個要執行的 instruction 的 offset。
   - 在 64-bit mode 中，EIP 被擴充到 64 bits 稱爲 RIP。

Offset

32-bit

- **Displacement** — An 8-bit, 16-bit, or 32-bit value.
- **Base** — The value in a general-purpose register.
- **Index** — The value in a general-purpose register.
- **Scale factor** — A value of 2, 4, or 8 that is multiplied by the index value.

包含以上幾個元素的 offset 被稱爲 effective address，除了 Scale factor 以外，每一個元素可以是正的值也可以是負的值(2s complement)。



64-bit

- **Displacement** — An 8-bit, 16-bit, or 32-bit value.
- **Base** — The value in a 32-bit (or 64-bit if REX.W is set) general-purpose register.
- **Index** — The value in a 32-bit (or 64-bit if REX.W is set) general-purpose register.
- **Scale factor** — A value of 2, 4, or 8 that is multiplied by the index value.
- **RIP + Displacement** — In 64-bit mode, RIP-relative addressing uses a signed 32-bit displacement to calculate the effective address of the next instruction by sign-extend the 32-bit value and add to the 64-bit value in RIP.

Media Programming

MMX

它定義了一種簡單且有彈性的SIMD execution model用來操作64-bit packed integer data，但是會使用到x87 FPU，故MMX與floating point的運算不能同時使用，而MMX在32-bit與64-bit的表現並無顯著的不同。

32-bit

- 含有 8 個 64-bit data register 稱作 MMX register。
- 三種的 data type
  - — 64-bit packed byte integers (signed and unsigned)
  - — 64-bit packed word integers (signed and unsigned)

— 64-bit packed doubleword integers (signed and unsigned)

64-bit

- 在 compatibility mode 與 64-bit mode，MMX 的指令就和它們在 protection mode 相同，只是 memory operand 要以上面所述之 offset computation 方法來 specify。

SSE

藉由增加對packed和scalar single-precision floating-point的操作，SSE延伸且擴充了SIMD execution model，未使用x87 FPU是它與MMX另外的不同點。

32-bit

- 含有 8 個 128-bit data register 稱作 XMM register。
- 一個提供XMM上operation的control、status bit的32-bit register，稱作 MXCSR register。
- 一種新的data type
  —128-bit packed single-precision floating-point data type
- 除了single-precision floating-point operation的指令外，開始有支援explicit prefetching of data, control of the cacheability of data, and control the ordering of store operations的指令。

64-bit

- 在compatibility mode，SSE的指令就和它們在protection mode相同。
- 在64-bit mode，才可以存取新增的8個XMM register(XMM8-XMM15) 。

SSE2

它多支援了packed double-precision floating-point與128-bit packed integers。

32-bit

- 六種新的data type
  — 128-bit packed double-precision floating-point
  — 128-bit packed byte integers
  — 128-bit packed word integers
  — 128-bit packed doubleword integers
  — 128-bit packed quadword integers
- 因爲支援pairs of packed double-precision floating-point的操作，所以允許在XMM registers上做更精確的運算，增進了在科學、工程應用上與3D幾何技術上processor的表現。
- 因爲提供了128-bit packed integers的操作，所以對SIMD的operation提供了更穩定、更大的處理能力，這樣的能力對於RSA authentication和RC5 encryption這類的應用是非常有幫助的。

64-bit

- 在compatibility mode，SSE2的指令就和它們在protection mode相同。
- 在64-bit mode，才可以存取新增的8個XMM register(XMM8-XMM15)
- 有些SSE2的instruction會直接操作到GPR，故要以上面所述之offset computation方法來specify。

SSE3

其 programming environment 與 SSE2 無任何差別也沒有新增任何 data type，它是針對 64-bit 的 architecture 來發展。

64-bit

- 在compatibility mode，SSE2的指令就和它們在protection mode相同。
- 在64-bit mode，才可以存取新增的8個XMM register(XMM8-XMM15)
- 有些SSE3的instruction會直接操作到GPR，故要以上面所述之offset computation方法來specify。
- 大部分的SIMD instruction是做垂直的運算，SSE3最大的不同點是增加了12個水平的加減運算指令。

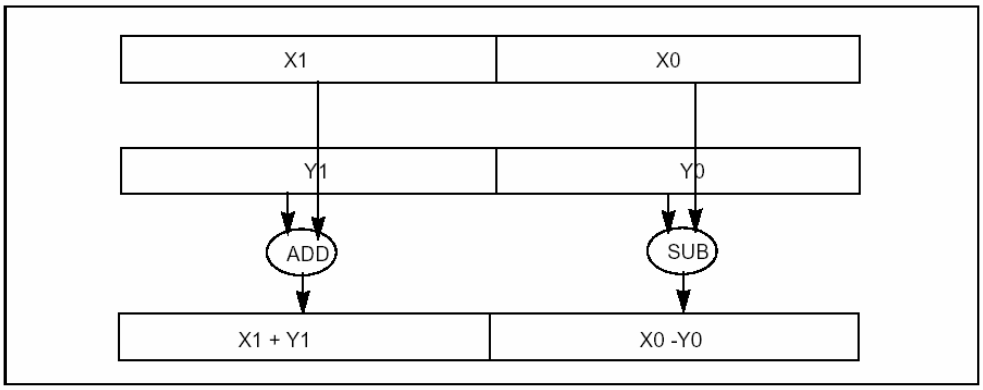


Figure 12-1. Asymmetric Processing in ADDSUBPD

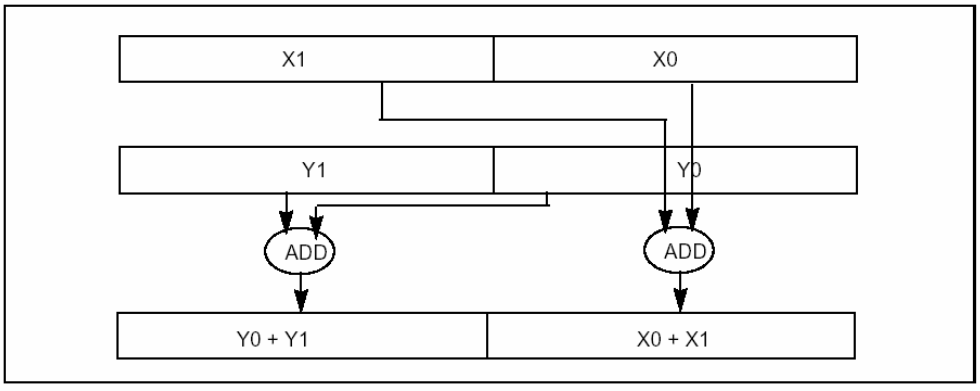


Figure 12-2. Horizontal Data Movement in HADDPD

Conclusion

64-bit 的 architecture 早已行之有年，各大 OS 也都有對應的平台，但是應用軟體產品卻很少有 64-bit 的產品而且缺乏 64-bit 的驅動程式，所以目前個人電腦平台在 64-bit 的應用並不廣泛。

其實，64-bit 的運算，在個人電腦上實際應用不多，在 32-bit 的 architecture 上，已經可以存取高達 4GB 的 memory，對於一邊個人電腦使用戶來說，記憶體的使用率距離 4GB 還有很大的距離；而以下是在 64-bit 的個人電腦中，對於檔案壓縮以及影像壓縮的測試，但由於軟體環境尚未成熟，應該還有很大的進步空間，至於要如何降低 64-bit 對記憶體使用量的影響，則是程式設計者的一大挑戰。

MiniGZip with ZLIB 1.1.4 檔案解壓縮測試				
	測試項目 （數字越低越好）	64 位元	32 位元	成長幅度
Athlon 64 3800+	壓縮檔案	3.625 秒	8.796 秒	+145%
	解壓檔案	1.203 秒	1.250 秒	+3.9%
Pentium 4F 3.2GHz	壓縮檔案	4.640 秒	7.828 秒	+69%
	解壓檔案	0.656 秒	0.719 秒	+9.6%
※註：ZLIB 採用針對 AMD 最佳化的版本。				

DivX 影像壓縮測試				
	測試項目 （數字越高越好）	64 位元	32 位元	成長幅度
Athlon 64 3800+	FPS	3.74 FP/s	3.72 FP/s	+0.5%
	En.FPS	35.71 FP/s	33.80 FP/s	+5.7%
Pentium 4F 3.2GHz	FPS	7.28 FP/s	5.97 FP/s	+22%
	En.FPS	27.13 FP/s	21.09 FP/s	+28.6%
※註：DivX encoder 採用針對 AMD 最佳化的 5.03 版本，我們將一個 770kB 的 wmv 檔案先轉爲 YUV 格式，再進行影像壓縮的工作。				

(以上資料引用自http://www.poweruser.com.tw/)

對於資料庫、資料解壓縮、資料加密、影像處理的特殊應用來說，64-bit 運算便佔了很大的優勢，因爲擁有 64-bit 的 flat memory 的存取能力，相較於 4GB 的 virtual memory 的空間限制，甚至是 PAE 延伸到相當於 64GB 的 36-bit 的 memory，對於大型的資料庫或者是伺服器，都將不會發生不規則存取大範圍記

憶體的程式行爲，而使記憶體存取效率不彰；因此，現在的高階運算及伺服器市場幾乎都已經被眾多 64-bit 的 RISC 處理器產品所主宰，並不令人感到意外。

有報導指出，微軟 Longhorn 核心的 OS(也就是已上市的 Vista) 推出時，應該就是推出 64-bit 產品的最佳時機，雖然 64-bit 系統的普遍化依然言之過早，但是 64-bit 的系統取代 32-bit 的系統似乎是長期趨勢，也許未來的 Longhorn 和市場持續成長的個人用 Linux 平臺，才是 64-bit 系統最後的答案。