# 64 位元處理器之記憶體模組測試方法

蕭啟維
中華大學資訊工程學系研究生
e-mail：e09902006@chu.edu.tw

張欽智
中華大學資訊工程學系助理教授
e-mail：changc@chu.edu.tw

## 摘要

動態記憶體(DRAM)在做成記憶體模組後，會進行功能測試與相容性測試。在現今軟體工程記憶體使用量越來越大，使得 DRAM 位元成長率(Bit Growth)逐年增加，記憶體模組測試若沒有變革以加強效能，相形之下成本、產出與品質便是相當大的壓力。

目前記憶體模組測試是在尚未進入作業系統或是在 DOS 下執行，隨著 x86 架構 CPU 之演進，由 32 位元轉換成 64 位元，進而採用多核心處理器方式，用以增進系統整體效能，若能妥善利用於記憶體測試，將能大幅減少測試時間成本並增加產出。

從本論文實驗數據分析，提出 64bit Mode 的測試方法，分析使用 64bit Mode 多核心測試在減少記憶體模組測試時間比傳統方式增加測試系統效能。

**關鍵詞**：動態隨機存取記憶體、多核心處理器、記憶體模組、記憶體測試。

## Abstract

After DRAM modules are produced, functional and compatibility tests will be conducted. Nowadays, because the memory requirement of software is growing, the DRAM size is increasing year by year. If testing methods do not change, the cost of production and quality insurance of DRAM is considerably high.

Memory module testing should be complete before the operating system is loading or under the DOS (Disk Operating System) mode. While traditional test programs are 32 bit mode on single-core, with the CPU architecture evolved from 32 bits to 64 bits the multi-core testing is more essential. If multi-core testing can be adopted, the testing cost will be reduced and productivity is increased.

In this paper we propose a method of 64 bit Mode memory testing. The experimental results show that the use of multi-core 64 bit test mode can reduce the test time. This test mode performs better than traditional methods.

**Keywords**: DRAM (Dynamic Random Access Memory), Multi-Core Processor, DIMM (Dual In-line Memory Module), Memory Testing.

## 1. 緒論

根據摩爾定律(Moore's Law)，積體電路(IC，Integrated Circuit)上可容納的電晶體數目，每隔 18 個月便會增加一倍 ，性能也增加一倍。DRAM 積體電路技術隨著電腦產業的蓬勃發展，容量也越來越大，以目前主流 DDR3 512Mx8 為例，一條 Dual Ranks Un-buffered Long DIMM 容量可高達 8GB。性能也從原來的 DDR3-1066 轉換成為 DDR3-1600 為主流規格。隨著容量倍增，測試時間也相對增長，對於模組產出與測試成本有著莫大影響。

有鑑於目前的記憶體模組測試程式均屬於 32 位元程式，並且單一處理器進行記憶體測試，測試時間冗長且不符記憶體模組測試成本。以 DDR3-1600 8GB 模組來看，有效測試時間將多達一小時，完全不符合記憶體模組生產業界量產標準，但記憶體模組測試卻又是不得不做的站點。目前個人電腦的 CPU 均屬於 64 位元並且為多核心架構[8]，若能妥善利用於記憶體模組測試將大幅提升產出，並且更貼近模擬出使用者的作業環境 (目前作業系統大多為 64 位元並且支援多核心架構)。

為了達到記憶體模組測試時間與增加錯誤覆蓋率(Fault Coverage)[1]之目的，本論文實驗透過 Intel x86 市面上主流之多核心處理器與本文理論上做結合用以比較出 32Bit 模式下單核心 與 64Bit Mode 多核心架構上於記憶體模組測試之差異。

## 2. 相關研究

### 2.1 真實模式(Real Mode)

此記憶體定址模式是為了 Intel 8086 提供舊有的程式提供向下兼容，在真實模式下實現程式與作業系統分段式線性定址空間，每區段空間共有 64Kbytes。最大線性定址空間為 1MB。

可以直接軟體存取 BIOS 常式(Routine Calls)及周邊硬體，沒有任何硬體等級的記憶體保護觀念或多工，x86 電腦會在啟動後，將 CPU 置於真實模式下。下圖為 Real-Address Model 下 Physical Address 計算方式。

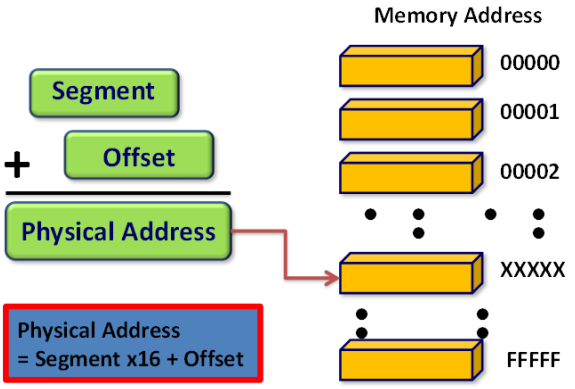


**圖 1 Real-Mode 下實體位址計算**

## 2.2 保護模式(Protected Mode)

在保護模式之下，記憶體管理分為 Segmentation 與 Paging Mechanism。Segmentation 提供在獨立的 Code、Data 和 Stack 的機制讓多個程式(Program)或者任務(Task)能夠在相同的處理器下工作而不會互相的干擾。Paging 則是提供程式(Program)的執行環境一個記憶體頁面與虛擬記憶體系統(Virtual-Memory System)。在保護模式下，Segmentation 需要數個，但是 Paging 是可以選擇的。這兩種機制(Segmentation and Paging) 可以在共享記憶體架構下在單一程式或任務、多工系統、多處理器系統運作 [9]。 如下圖 2 Segmentation 提供一個可以區分 CPU 可定址的記憶體空間(Liner Address Space)，如果有一個或一個以上的程式或是任務在一個處理器上執行，程式會被安排在屬於自己的 Segment 內。程式與程式之間並不會相互影響。

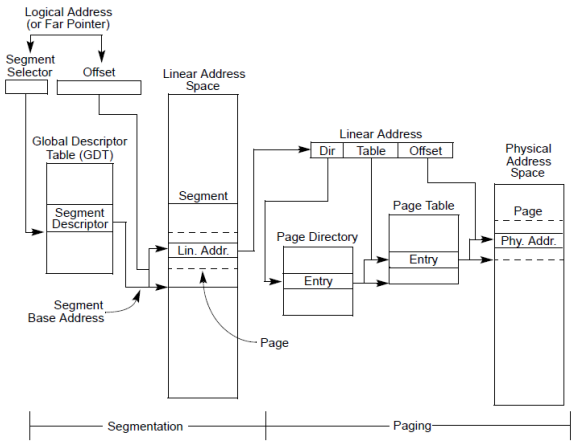


**圖 2 Segmentation 與 Paging [9]**

## 2.3 IA-32e Mode

IA-32e (AMD 稱為 Long Mode [3]) 為傳統 Protected Mode 的延伸。在 IA-32e 模式中分成兩個子模式 64Bit Mode 與相容模式 (Compatibility Mode) [9]。 下圖 3 為 Long Mode 下 64Bit Mode 與 Compatibility Mode 記憶體管理示意圖。

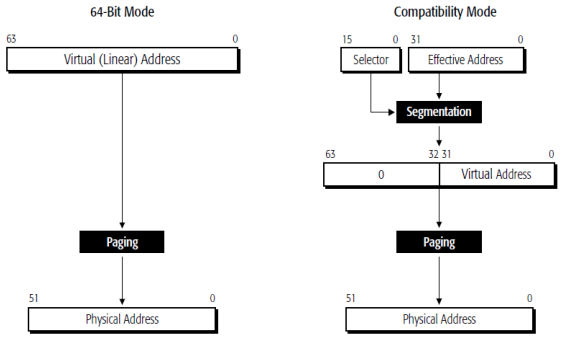


**圖 3 Long Mode 記憶體管理示意圖 [3]**

在 64-Bit 模式下，可以完全使用 64 位元定址並且使用 Register-Extension。 Instruction Pointer 也由原來的 32bit 延伸至 64bit (RIP)。Register-Extensions 是使用 REX Prefixes 來達成 64bit 暫存器延伸，此外並增加了八個 GPRs(General Purpose Register) R8~R15 與八個 128-bits XMM 暫存器(XMM8~XMM15) [2]。REX prefixes 也提供十六個 GPRs 進行 Byte、Word、 Double Word、 Quad Word Register 運算 [2]。

相容模式允許 64Bit 作業系統下，執行原有 16 位元或是 32 位元的應用程式，而不需要重新編譯。應用程式在此模式下執行，使用 32 位元或 16 位元的定址與存取 [4]。

## 2.4 記憶體管理機制

X86 記憶體架構支援直接實體記憶體定址或者虛擬記憶體(透過 Paging)。 當使用實體記憶體時，線性位址等於實體上的記憶體位址。當 Paging 使用時，只有最近被存取到的頁面才會放置於實體記憶體中。 Page 在實體記憶體內存放著兩個資料結構 Page Directories 和 Page Table。控制暫存器 CR3 於 Page 機制中放置 Page Directory base 。一個 Page Directory 都包含放置 Page Table Base 的實體記憶體位址，存取權和記憶體管理資訊。一個 Page Table 進入都包含 Page Frames 的實體位址，存取權與記憶體管理資訊。當使用 Page Mechanism 時，Liner Address 會被分成 Page Directory、Page Table、 Page Frames，如圖 4 所示 [9]。一個系統可以是單一 directory 或是多個。例如

每個 Task 可以是擁有屬於自己的 Page directory。

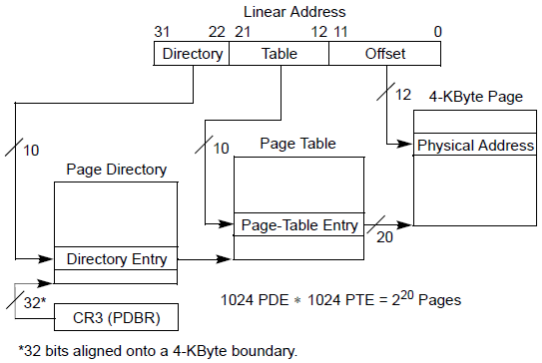


**圖 4 Liner Address Translation [9]**

### 2.5 多處理器介紹與管理

Intel64 與 IA-32 架構提供多處理器管理機制與提升效能基於多處理器連接共同系統匯流排。下列機制將分別有助於對稱式多處理器(Symmetric-Mulitprocessing ,SMP) [9]：

- 鎖定匯流排 (Bus Locking) 與執行Atomic Operation 時Cache Coherency 管理機制。
- 於 Process Chip 內會有一個 Advance Programmable Interrupt Controller (APIC)
- Pentium 4 ， Intel Xeon and P6 family 於處理器內連結 Second level Cache(Level 2，L2)
- Intel Xeon 則於處理器內連結 Third Level Cache(Level 3，L3)
- 執行緒(Hyper-Threading)，於 Intel 64 與 IA-32 架構內開起單一處理器同時執行兩個或是更多的線程。

這些多處理架構之機制有下列特性：

- 維持系統記憶體的一致，當有兩個或是更多處理器試圖同時去存取相同位址之系統記憶體時會有通訊的機制或者是記憶體存取協定來使資料是一致的並且在某些例子中會允許一個處理器暫時把記憶體位址鎖住。
- 維持快取記憶體中的一致性，當一個處理器再另一個處理器中的快取記憶體存取資料時，它不能有錯誤資料的發生，若更改資料內容時，其它的處理器也必需要接收到修改過後的資料。
- 可預測之記憶體寫入順序，在某些情形下，在相同的程序之程式記憶體寫入準確被監控很相當重要的。
- 中斷(Interrupt) 在於一組處理器之間的處理，當數個處理器平行運作於作業系統中，整個系統非常需要一個中央式機制用來接收與發佈中斷至相對應之處理器。
- 為增加系統整體效能，會利用多執行緒(Multi-Threaded) 與多處理器運行(Multi-Process) 於先進作業系統與應用程式。

## 3. 基於 64Bit Mode 下之系統架構

### 3.1 Page 建置

當系統運行於保護模式時，IA-32 架構允許線性位址直接 mapped 到較大的實體記憶體中(e.g.,4 GB of RAM) 或者間接(使用 Paging) mapped 到較小的實體記憶體和磁碟儲存裝置中。後者的 mapped 方式是由線性位址參照虛擬記憶體或者隨選分頁(Demand-Paged)虛擬記憶體中。當使用分頁，處理器會把線性位址空間區分為固定 size 之頁面(4KB、2MB or 4M B) mapped 到實體記憶體或者是磁碟儲存裝置。當程式(or task) 參照到邏輯位址時，處理器會轉換成線性位址，然後依照分頁機制把線性位址轉換成為相對應之實體記憶體位址。若 page 包含線性位址不在實體記憶體中，處理器會產生一個 Page-Fault Exception (#PF)。這個例外處理會交由作業系統或是執行從磁碟儲存裝置將 Page 載入實體記憶體中，當 Page 被載入實體記憶體中後，回到例外處理(Exception Handler) 並使 Exception 重新啟動。這些資訊讓處理器對映線性位址與實體位址空間並產生 Page-Fault-Exception 是包含 Page-Directories 與 Page Table 儲存於記憶體中。位址轉換最小 bus cycle ，最近存取的分頁目錄(Page-Directory) 和分頁表(Page table) 進入點被快取至處理器內稱做 TLBs(Translation Lookaside Buffers) 。在 TLBs 滿足大多數讀取分頁目錄與分頁表而不需要匯流排週期(Bus Cycle)。額外的匯流排週期只有發生在分頁表不在 TLBs 內，通常是某個頁面很久都沒有被存取過。下圖 5 所示在 IA-32e 中對映現性位址 2-Mbyte Page 時 之 PML4，PDP，PDE，PTE 之轉換階層。此分頁模式可以定址到 $2^{27}$ Pages 並且拓展線性位址空間至 $2^{48}$ Bytes。2M-Byte Page Size 設定在於 PDE.PS Flag [9]。此時線性位址分成四個部份。

- PML4 Table Entry ： Bit 47：39 提供 PDP Table 之基底實體位址。
- Page Directory Table Entry： Bit 38：30 提供 PDE Table 之基底實體位址。
- Page Directory Entry： Bit 29：21 提供 2M-Byte Page 之基底實體位址。
- Page Offset ： Bit 20：0 提供該 Page 之實體位址。

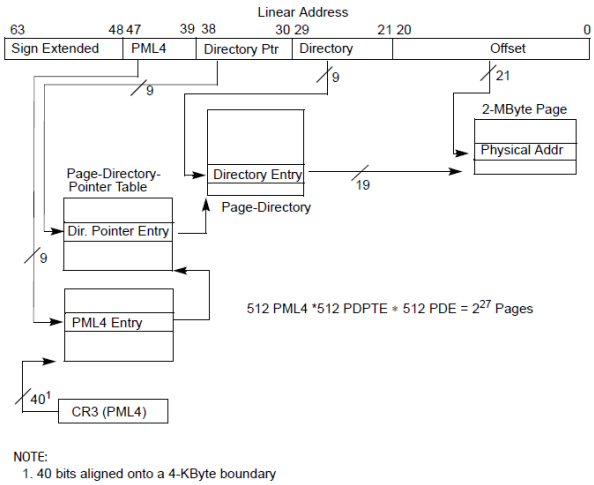


**圖 5 Liner Address Translation with PAE (2-Mbyte Pages) [9]**

## 3.2 多核心初始化

在多處理器或多核心架構中，BIOS 必須要多附加的責任需要去執行。

- 讓所有的 APs(Application Processor) 進入睡眠狀態，它們不需要像 BSP(Bootstrap Processor) 一樣去執行相同的 BIOS code 並且 BIOS 也不是多執行緒之多工處理程序。
- 初始化 APICs (Advanced Programmable Interrupt Controller) 與 Multi-Processor。
- 建立 MP Configuration Table 讓 OS ，APICs 與 APs 相互溝通。

當按下 Reset 按鈕時供應系統之電源時，硬體電路會發出"RESET"的訊號給所有的系統硬體進入 Initial state[10]。此時所有 Active Processor 會進入 POST (Power-on Self Test) 程序。為了預防在 POST 之後所有的 APs 去執行到 BIOS Code。 BIOS 會選擇某一個處理器當成 BSP 然後再依序讓所有的 APs 進入睡眠之中[10]。BIOS 可以使用 APIC ID 去辨識 AP ，讓它去執行適合之程序。只有 BSP 在 POST 程序之後可以繼續載入 OS。 在多核心兼容系統之中，BSP 的功能和 AP 是沒有任何差別的。

### 3.2.1 Application Processors Startup

由於 APs 於 BIOS 階段時便進入睡眠狀態(HALT state) ，同時當第一道指令執行時，AP 的狀態如下：

- APs 從 OS 執行程式時被受約束的(BIOS 或是硬體)
- APs 是處於 HALT State 之中並且中斷 (interrupts) 是被 disable ，APs Local APIC 會去 monitor APIC Bus 並且只會接受 INIT or Startup Inter-processor Interrupts (IPIs)。

任何一個 AP 都可以被 BSP 或是其它的 Active AP 所喚醒。OS 使用下面的演算法去導引 APs 去執行在 OS 之中的 Initial Task。下列所示演算法包含 Inter processor Interrupts 和短暫的 Delay 讓 AP 去反應 wakeup Command。

```
BSP sends AP an INIT IPI
BSP DELAYs (10mSec)
If (APIC_VERSION is not an 82489DX) {
      BSP sends AP a STARTUP IPI
      BSP DELAYs (200μSEC)
      BSP sends AP a STARTUP IPI
      BSP DELAYs (200μSEC)
}BSP verifies synchronization with executing AP
```

當發出 Startup IPI 時，AP 會從 Real Mode 下之 Address 000VV000h 開始去執行，VV 是一個 8 bit 的向量是 IPI Messages 的一部份。Startup IPI 是不可遮罩之中斷，它並不會改變 APs 的狀態(除了改變 Instruction Pointer) [11]。當 OS 發出 Startup IPI 去 wakeup APs 時，AP 會立即跳至 000VV000h 去執行 OS 所安排之初始程序。

### 3.2.2 在多核心系統中辨識處理器

在 BIOS 完成 MP Initialization Protocol 後，每一個邏輯處理器會有一個獨特的 APIC ID，軟體可以從下列方法讀取[9]：

- From local APIC：當 code 執行邏輯處理器時可以讀出 APIC ID。若 Local APIC 運作於 x2APIC Mode 時，可透過 RDMSR 指令去讀出，使用此方法和 CPUID leaf 0Bh 是相等的。若 Local APIC 運作於 xAPIC Mode 時，使用 MOV 指令 去讀出 APIC ID 暫存器的值。
- ACPI or MP Table[11]：在 BIOS MP Initialization Protocol 之中 BIOS 會建立 ACPI 與 MP table。這些表格會標示出處理器列表以及它們各自再系統端之 Local APIC ID。
- Read Initial APIC ID： (如果處理器不支援 CPUID leaf 0Bh 這項功能)當 Power up 時，APIC ID 會被賦予於邏輯處理器。這是 Initial APIC ID 可以從 CPUID.1：EBX[31：

24]之中取得，它可能會和 local APIC 所讀取到的會不相同。這個 Initial APIC ID 可以於多核心或多處理器系統中辨識邏輯處理器。

- Read 32-bit APIC ID From CPUID leaf 0Bh：若處理器有支援，便可以從 CPUID.0B：EDX[31:0]中取得，可以從獨特的 APIC ID 去在多核心或多處理器系統之中辨識出邏輯處理器。

從下圖 6 所示兩個例子來說明 APIC ID bit 欄位。Bit 2：1 是 Package Identifier，若是系統屬於多個處理器叢集，Bit 4：3 是 Cluster ID。Bit 0 是使用於辨識兩個邏輯處理器於一個封裝之內。若處理器不支援 Hyper-Threading 技術則 Bit 0 恆為 0。

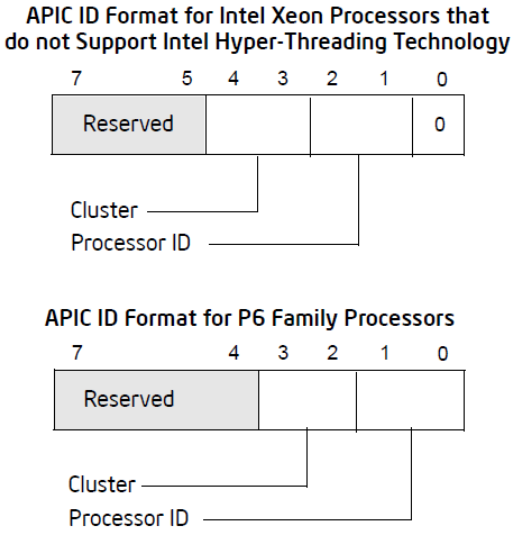


**圖 6 APIC ID 格式 [9]**

### 3.3 進入64Bit Mode

Processor 在 BIOS 系統設定之後處於 Real Mode 下。一些基本的資料結構和程式碼必須先載入到記憶體之中用以讓 Processor 得以進入保護模式之中。在進入 64 bit Mode 時，Processor 必須先置於保護模式並且 Paging Enable。IA-32e Operation 也需要 PAE (Physical-Address-Extend)和 4 階層加強 Paging 結構，如同上個章節所談到的。 Initial IA-32e 需要的步驟如下 [9]：

- 從保護模式之中進入，使用 CR0.PG = 0 來關閉 Paging。(指令必須來自 Identity-mapped page)
- 使用 CR4.PAE 致能 PAE。當 init.IA-32e 時若致能失敗則會產生#GP Fault。
- 將 PML4 Table 之實體位址載入 CR3。
- 設定 IA32_EFER.LME =1 進入 IA-32e Mode。
- 設定 CR0.PG = 1 去致能 Enable 64 Bit mode 之 Paging Tables 在打開 IA-32e 之前必須在實體記憶體位址中之第一個 4GB 之內。這是必要的，因為 MOV CR3 指令在 Protected Mode 下執行，只有寫入低 32 位元之暫存器，限制了 Tables 於記憶體位址中 4GB 之內。依照之前章節去設定 Paging 然後開啟 IA-32e Mode (AMD 為 Long Mode [5])。如此便進入 64 Bit Mode 。在喚醒多顆處理器後，也按照相同的步驟，便可使所有的處理器都處於 64Bit Mode 之中，接下來便可以撰寫後面之記憶體測試 Pattern。下圖 7 為讓所有處理器進入 64 Bit Mode 簡單示意流程圖。

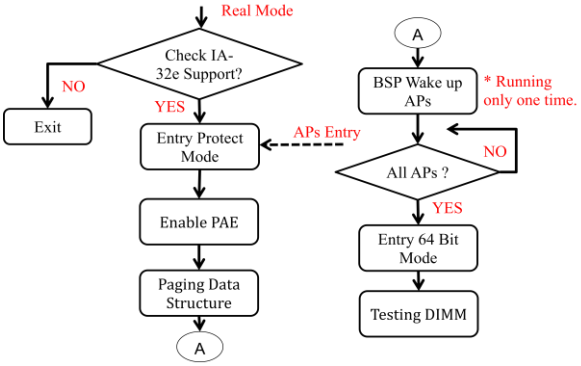


**圖 7 進入 64Bit Mode 流程圖**

## 4. 系統實驗與分析

當所有處理器都進入64Bit mode ，此時可以開始進行本論文之重點。利用64Bit 多核心處理器 來進行記憶體模組之系統測試。首先簡單介紹測試Pattern 演算法，接下來便是使用測試Pattern 來比較測試時間與DRAM IC Fault Coverage，用以判斷此方法是否優於目前於模組測試廠使用方法。

### 4.1 記憶體模組測試 Pattern 之介紹

#### 4.1.1 March Pattern[1]

以非常簡單的步驟與數值變化來測試(all 0 and all 1)，優點為速度快， 若記憶體很容易發生 Fail 的狀況，則可由此快速的 Pattern 刷出 Fail 的顆粒。March Algorithm 可細分為下列 6 個 Steps，如圖 8 所示：

1. 位址由低往高，逐一寫入 0x00000000。
2. 位址由低往高，逐一讀出資料， 比較是否為 0x00000000 並反向寫回，即為 0xFFFFFFFF。

3. 位址由低往高，逐一讀出資料，比較是否為0xFFFFFFFF，並反向寫回。
4. 位址由高往低，逐一讀出資料，比較是否為 0x00000000，並反向寫回。
5. 位址由高往低，逐一讀出資料，比較是否為 0xFFFFFFFF，並反向寫回。
6. 位址由高往低，逐一讀出資料，比對是否為 0x00000000。

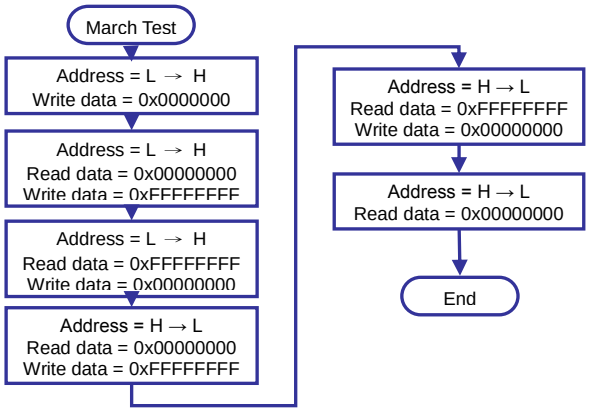


**圖 8 March Pattern 流程圖**

### 4.1.2 Moving Inversion Pattern[6]

以位移的方式將 0 與 1 由最低位元向最高位元搬移，執行順序為：

1. Write 0：將Pattern由低位址寫至高位址
2. Read 0 Write 1：由低位址往高位址讀取，並寫回相反的值。
3. Read 1 Write 0：由高位址往低位址讀取，並寫回相反的值。
4. 重複地2與第3步驟後即完成一個Step。

整個流程如圖 9 所示。

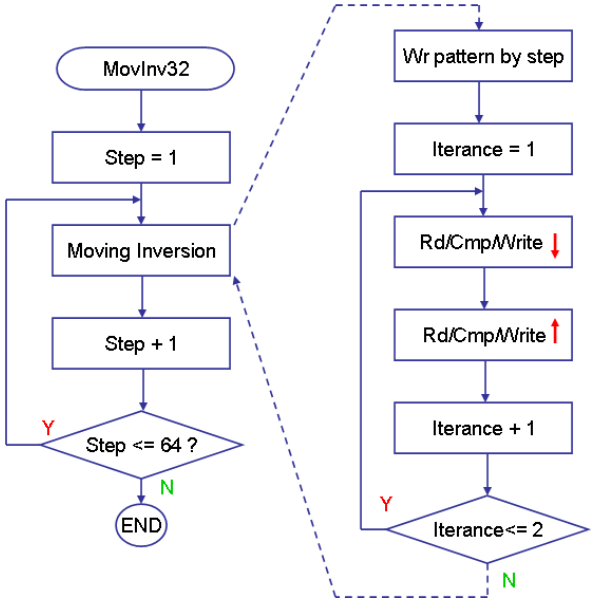


**圖 9 Moving Inversion Pattern 流程圖**

### 4.1.3 Random Number Sequence Pattern [6]

將記憶體填滿非規律的亂數，因為有非常多種組合所以可有效的測試出有問題的記憶體。 Random Number Algorithm 可細分為下列 5 個 Steps [6]，流程如圖10所示：

1. 位址由低往高，逐一寫入亂數
2. 位址由低往高，逐一讀出資料，比較是否正確並反向寫回
3. 位址由低往高，逐一讀出資料，比較是否正確並反向寫回
4. 位址由低往高，逐一讀出資料，比較是否正確並反向寫回
5. 位址由低往高，逐一讀出資料，比較是否正確並反向寫回

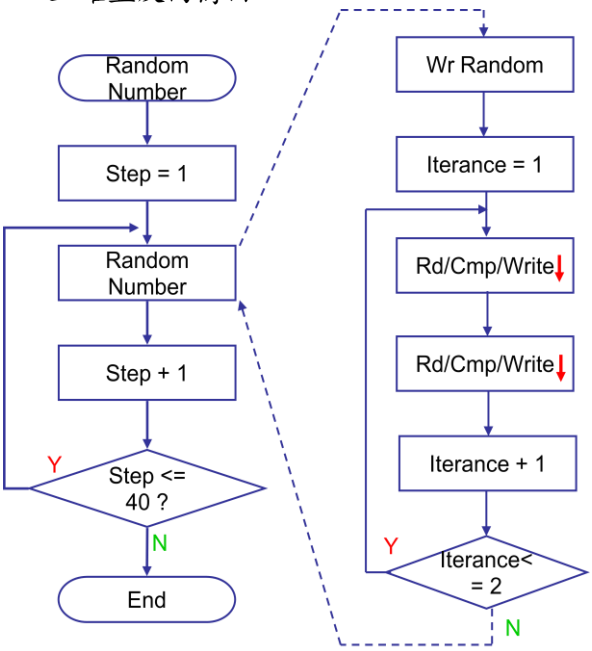


**圖 10 Random Number Sequence Pattern 流程圖**

## 4.2 測試時間比較

分別使用上述之Pattern 來攥寫32Bit、64Bit與64Bit開啟多核心測試來比較測試記憶體實際上運行之時間。藉以比較三種模式之系統效能並且分析實驗結果，用來判定是否適合使用測試記憶體。在此使用了三種Intel 主流之硬體平台，其資訊如下表1所示，平台A 為Intel H77使用CPU為I5-3550(Ivy Bridge)共有兩個記憶體通道。平台B 為Intel Z87 使用CPU 為I5-4670k(Haswell)共有兩個記憶體通道。平台C 為Intel X79 使用CPU 為I7-4820k (Ivy Bridge-E)，記憶體通道共有四個。

表 1 測試平台資訊表

| Platform | 主機板 | PCH | CPU | 時脈 | 腳位 | 核心 / 執行緒 | L2 快取 | L3 快取 |
|---|---|---|---|---|---|---|---|---|
| A | Gigabyte GA-H77 D53H | Intel H77 | Intel i5-3550 | 3.3 G | 1155 | 4C / 4T | 256K x4 | 6MB |
| B | ASUS Z87-PRO | Intel Z87 | Intel i5-4670 k | 3.8 G | 1150 | 4C / 4T | 256K x4 | 6MB |
| C | ASUS P9X79 | Intel X79 | Intel i7-4820 k | 3.7 G | 2011 | 4C / 8T | 256K x4 | 10M B |

以Platform A，B 與C 分別比較三種模式之測試時，縱軸為以March(32)，Movin(32)，Ranodm(32) 為時間基準之倍數。橫軸為記憶體容量與記憶體通道數。數值越大，代表所花費的時間越少代表其效率越高。Memory 時脈設定為：DDR3-1600 11-11-11-28 (tCL-tRCD-tRP-tRAS);電壓設定為1.5v。測試結果如圖11、12與13所示。

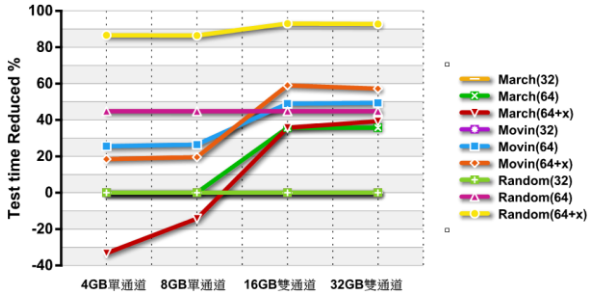


圖 11 Platform A 減少測試時間比率

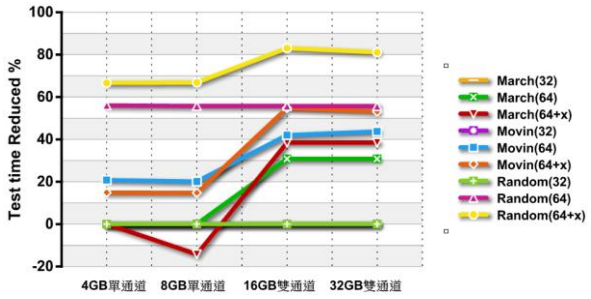


圖 12 Platform B 減少測試時間比率

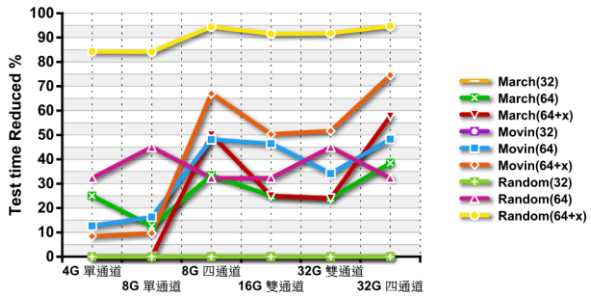


圖 13 Platform C 減少測試時間比率

透過上面實驗數據，我們可以得到以下幾點結論：

- 在多記憶體通道效能比 32bit < 64bit < 64bit+x (多核心)。以Intel I7-4280k (4C/8T) 使用32GB 四通道來看效能可以增加最高至94.73%。
- 若以單通道March，Movin 64bit 多核心的效能會比64bit 還要差。主要的原因是由於當多顆處理器同時搶著共同的記憶體資源並且記憶體頻寬不足以應付時會發生處理器在等待另一顆處理器 [8]。同樣容量8GB單通道與四通道可以看出記憶體頻寬於多處理器系統之重要性[15]。
- March 與 Moving Inversion 之 Processing Type 為Memory 而Random 則為Compute(產生亂數) + Memory。在多核心系統之下，加強了Compute 與Memory 之能力，故使得效能得以增進[15]。
- 此三組平台實驗數據有些差異但基本的趨勢是相同的，主要取決於CPU 時脈，核心數，執行緒(Hyper-thread)，記憶體通道數與 Memory Controller 對於記憶體之 Policy [12, 13, 14]。

## 5. 結論與未來展望

從以上之實驗來分析數據，64Bit Mode 由於暫存器是 32Bit Mode 的兩倍，一次存取記憶體之內容也是兩倍[2, 5]，加上多核心架構使記憶體通道頻寬加大更能彰顯系統整體之效能。以 Intel 4820K(Ivy Bridge - E) 為例，擁有 4 Cores 與多執行緒(Hyper-thread)，再加上記憶體四通道，於 64bit Mode 多核心架構下比原本 32Bit Mode 單核心最多可以節省測試時間多達 94.73%，這大大幫助了記憶體模組測試廠減少測試成本與增加測試產出。

在後續的研究，計畫探討錯誤覆蓋率(Fault Coverage)的問題，在持續降低測試時間並能提昇錯誤覆蓋率，用來降低記憶體模組測試的成本與加強記憶體模組之產出。在初步的實驗中，得知在 64Bit Mode 多核心單通道之中，記憶體之系統效能反而不如 32Bit Mode 單核心單通道，其原因在於記憶體頻寬之不足所影響。但對於核心數、Hyper-Thread、記憶體通道頻寬，三者如何權衡(Trade-off )確實是一個很有趣的議題[15]，若有此方面資料，便能在往後如何加快對於需要大量記憶體資料之應用程式提出建議。

## 參考文獻

[1] 黃寶興(民96)，動態隨機存取記憶體測試

實務與測試時間最佳化技術之研究，輔仁大學電子工程學系碩士論文。
[2] Advanced Micro Devices, “AMD 64-Bit Technology The AMD x86-64 Architecture Programmers Overview”, Advanced Micro Devices, Jan. 2001.
[3] Advanced Micro Devices, “AMD64 Technology AMD64 Architecture Programmer’s Manual Volume 1：Application Programming”, Advanced Micro Devices, Sep. 2007.
[4] Advanced Micro Devices, “AMD64 Technology AMD64 Architecture Programmer’s Manual Volume 2： System Programming”, Advanced Micro Devices, Sep. 2007.
[5] Advanced Micro Devices, “AMD64 Technology AMD64 Architecture Programmer’s Manual Volume 3：General-Purpose and System Instructions”, Advanced Micro Devices, Sep. 2007.
[6] Brady, Chris and Demeulemeester, Samuel, “Memtest86+ Source Code”, http：//www.memtest.org.
[7] Grysztar, Tomasz,” Entering long mode - simple examples”, http：//flatassembler.net/examples.php.
[8] Hennessy, John L. and Patterson, David A., Computer Architecture - A Quantitative Approach (5. ed.), Morgan Kaufmann, 2012.
[9] Intel, Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 3A：System Programming Guide, Part 1, Intel, Sep. 2013.
[10] Intel, Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 3B：System Programming Guide, Part 2, Intel, Sep. 2013.
[11] Intel, Multi-Processor Specification Version 1.4, Intel, May 1977.
[12] Mutlu, O. and Moscibroda, T., “Parallelism-Aware Batch Scheduling：Enabling High-Performance And Fair Shared Memory Controllers”, IEEE Micro, vol. 29, no.1, Jan./Feb. 2009, pp. 22-32 .
[13] Mutlu, O. and Moscibroda, T., “Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors”, 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2007, Dec.2007, pp. 146-160.
[14] Mutlu, O. and Moscibroda, T., “Parallelism-Aware Batch Scheduling：Enhancing both Performance and Fairness of Shared DRAM Systems “, 35th International Symposium on Computer Architecture, ISCA 2008, Jun. 2008, pp. 63-74.
[15] Sancho, J.C. and Lang, M. and Kerbyson, D.K., “Analyzing the Trade-off between Multiple Memory Controllers and Memory Channels on Multi-core Processor Performance”, 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW), April 2010, pp.1-7.