公告本

申請日期：88、3、26	案號：88104839
類別：	G06F 9/44

（以上各欄由本局填註）

發明專利說明書 432332

一、發明名稱	中文	用於物件導向記憶系統之裝置及方法
	英文	APPARATUS AND METHOD FOR OBJECT-ORIENTED MEMORY SYSTEM
二、發明人	姓名（中文）	1. 葛鈞瑞 K. 史勞特 2. 湯瑪士 艘波 3. 桑尼爾 K. 波帕迪克 4. 李靜惠
	姓名（英文）	1. GREGORY K. SLAUGHTER 2. THOMAS SAULPAUGH 3. SUNIL K. BOPARDIKAR 4. ZI-HUI LI
	國籍	1. 美國 2. 美國 3. 美國 4. 中國
	住、居所	1. 美國加州帕羅奧圖市艾默森街3326號 2. 美國加州聖荷西市伯瑞特哈特大道6938號 3. 美國加州陽光谷市櫻林大道537號 4. 美國加州山景市克利斯塔諾大道1929號
三、申請人	姓名（名稱）（中文）	1. 美商太陽微系統公司
	姓名（名稱）（英文）	1. SUN MICROSYSTEMS, INC.
	國籍	1. 美國
	住、居所（事務所）	1. 美國加州帕羅奧圖市聖安東尼歐路901號
	代表人姓名（中文）	1. 肯尼斯 歐森
	代表人姓名（英文）	1. KENNETH OLSEN

本案已向

國(地區)申請專利	申請日期	案號	主張優先權
美國 US	1998/03/26	09/048,333	有

有關微生物已寄存於	寄存日期	寄存號碼

無

五、發明說明 (1)

發明背景

發明領域

本發明大體上與電腦記憶相關，特別是，與平台獨立的記憶方法和系統。

相關的技術說明

為了供電腦系統與使用者溝通，系統通常包括例如顯示螢幕、列表機、和鍵盤等許多的週邊裝置。這些週邊裝置的每一個需要一特別的軟體元件，稱為裝置驅動程式，以在該週邊裝置和電腦系統的其餘部份之間提供有秩序的資料交換。

例如，一家公司發展新的彩色列表機，想要確保列表機對具有不同硬體配置的電腦("平台")可用，包括以最常見型態中央處理單元，例如昇陽微系統公司生產的史巴克(SPARC)、摩托羅拉公司生產的強力個人電腦(PowerPC)、和英代爾公司生產的奔騰(Pentium)等為基礎的系統。目前，這需要列表機製造業者為每個平台寫一單獨的裝置驅動程式，因而使列表機的發展成本上昇。而且，對每個平台特製裝置驅動程式的需求也表示當一新的平台介紹出來時，可能無法立刻取得最常見型態週邊裝置之裝置驅動程式。

對每個平台特製裝置驅動程式的需求密切地與每個平台的各種記憶特性有關。提供記憶系統和方法，可允許為一週邊裝置寫一單一裝置驅動程式，藉此允許週邊裝置在所有的平台，包括在未來將被介紹的新平台，上運作將會是

五、發明說明 (2)

令人期待的。

發明概要

本發明的特徵和優點將會在以下的說明中陳述，且部份可由說明中明瞭，或可由本發明的實現中習知。本發明的目的和其他優點將藉由書面說明、相關申請專利範圍、以及伴隨的圖式中特別指出的裝置、方法、和生產物件實現與穫得。

為了要達到所有這些優點，並在具體實施與概括地說明時與本發明的目的一致，本發明提出一種方法，以提供記憶功能給一具有中央處理單元之電腦系統的物件導向用戶端軟體元件。該方法包含提供一第一組記憶體類別的步驟，第一組的每個類別是平台獨立的；提供一第二組記憶體類別，第二組的每個類別是第一組類別的一子類別，且是平台相關的；並藉由僅存取第一組類別的物件執行用戶端元件記憶函數。

在另一個態樣中，本發明包括在一物件導向電腦系統中，用以執行記憶函數的裝置，包含一第一組記憶體類別，第一組的每個類別是平台獨立的；一第二組記憶體類別，第二組的每個類別是第一組類別的一子類別，且是平台相關的；一裝置驅動程式，僅存取第一組類別的物件；及一匯流排管理器，存取第二組的類別。

此外在另一態樣中，本發明包括含有指令以提供具有中央處理單元之電腦系統的物件導向用戶端軟體元件記憶功能之電腦可讀媒體。該等指令包含一第一組記憶體類別，

五、發明說明 (3)

第一組的每個類別是平台獨立的；及一第二組記憶體類別，第二組的每個類別是第一組類別的一子類別，且是平台相關的。

應該要了解的是前面的一般性描述和以下詳細描述都是做為範例和說明的，且是為了提供本發明所申請專利之進一步解釋。

所包含的隨附圖式提供本發明的進一步瞭解，且合併在本說明書中並構成其一部份，說明本發明的一個/數個具體實施例，並和說明一起，用來解釋本發明的原理。

圖式概述

隨附的圖式，合併在本說明書中並構成其一部份，說明本發明的具體實施例，並和說明一起，用來解釋本發明的目的、優點和原理。

在該等圖式中：

√圖1是實施本發明之電腦系統的一個硬體方塊圖；

√圖2是圖1電腦系統的物件導向軟體的圖解；

√圖3是顯示圖2所顯示軟體的一部份之記憶系統的類別架構的圖解；

較佳實施例詳述

構成本發明的方法與裝置包含在一物件導向作業系統中。所揭露的具體實施例以加州山景市昇陽微系統公司所提供的爪哇程式環境實施。然而，本發明並非以此為限，而是可以合併到習知該項技藝人士知道的其他電腦系統、其他作業系統、和其他程式編寫環境中。爪哇、爪哇有關

五、發明說明 (4)

的商標、昇陽、昇陽微系統與昇陽圖案是加州山景市昇陽微系統公司的商標或註冊商標。

現在將詳細參考依照本發明的一種實施，如圖式中所說明。只要可能，相同的參考數字將會在所有圖式和以下說明中參照相同的或相似的部份。

圖1顯示包括本發明的一電腦系統10。電腦系統10包括一中央處理單元(CPU)12，它可以是例如一昇陽公司的SPARC、摩托羅拉公司的Power PC或英代爾公司的Pentium。電腦系統10可代表各種的計算裝置。舉例來說，系統10可代表一廣泛地使用在家庭中和辦公室的標準個人電腦。或者，系統10可包含一更特殊化的"智慧型"系統，例如用在接收高解析度電視的隨選視訊盒，或多功能蜂巢式電話。

中央處理單元12連接到記憶體14，它可以包括各種不同型態的記憶體，例如隨機存取記憶體(RAM)和唯讀記憶體(ROM)。中央處理單元12也連接到一擴充匯流排16，它可以是，例如一周邊元件互連(PCI)匯流排。各種不同類型的輸入和輸出裝置19、20和22連接到匯流排16。舉例來說，輸入裝置18可以是數據機或網路介面卡，將系統10連接到一電話線路或區域網路。輸入裝置20可以是，例如，一鍵盤而輸出裝置22可以是，例如，列表機。系統10可選擇性地包括一大量儲存裝置24，例如由輸入／輸出匯流排26，例如，小型電腦系統介面(SCSI)匯流排所連接的硬式磁碟磁碟機。系統10也包括一，在中央處理單元12控制之

五、發明說明 (5)

下的，直接記憶體存取(DMA)控制器23，它可在記憶體14和PCI匯流排16之間提供直接的資料傳送。

現在參考圖2，其中顯示儲存在圖1記憶體14中的軟體之圖解。圖2顯示邏輯上安排在一連串階層的軟體。較上面的階層30是"平台獨立"的。"平台"一辭通常指中央處理單元、實體記憶體、和永久地附加的裝置和匯流排。因此，包含在平台獨立層32的軟體是，根據任何中央處理單元、不論現存的或未來將發展的，以能使用在任何平台而不用修改的方式撰寫。第二層34是平台相關的。因此，階層34的軟體必須依電腦系統10的特定平台而修改。

平台獨立層32包括一應用程式層36，它為使用者執行特定的運作，例如桌上排版、管理電話呼叫、或資料庫管理。應用層36接口包含有一稱為"Java™虛擬機器"(JVM)元件的運行時間系統38。JVM是一軟體元件，它以機器獨立位元組碼的形式接收應用程式所產生的指令，並藉由動態地轉換與執行來解譯那些指令。JVM接口特定的平台以執行所要的功能。所揭露的具體實施例使用一爪哇虛擬機器，但可使用其他型態的虛擬機器，如習知該項技藝人士所熟知的。JVM的運作為習知該項技藝人士所熟知，而且在例如印第安那州印第安那玻里斯市新騎士出版其公司所出版、提姆·律察所著的Java!一書，和愛迪生-魏力斯理(1997)公司林德漢和葉爾林所著The Java Virtual Machine Specification中討論過。

運行時間系統38也包括一裝置介面部分42，它支援裝置

五、發明說明（6）

的運作，例如匯流排16和26與裝置18、20、22（圖1）。特別地，裝置介面42包括裝置管理器46和雜項管理器48。裝置介面42也包括裝置驅動程式50，它是為每一個裝置18、20和22寫的物件導向程式。注意裝置驅動程式50包含在平台獨立層32，也因此，一旦寫好，可用來支援任何平台上，現存的和未來將發展，的裝置18、20和22。同樣地，裝置介面42包括平台獨立匯流排管理器53，它是為匯流排，如圖1的PCI匯流排16和SCSI匯流排26，寫的物件導向程式。裝置介面42也包括記憶體類別52，它組成本發明的一部份且將在以下做更詳細討論。

裝置介面42也包括額外的元件，例如一系統資料庫56和一支援電腦系統10運作的系統載入程式54。同樣地，運行時間層38包括額外的函數58，支援例如輸入/輸出、網路運作、圖形、列印、多媒體等運作。

平台相關層34包括一平台介面60、一作業系統原生方法層61、一微核心62和一啟動介面64。平台介面60包括虛擬機器系統函數程式館處置器66，它可用爪哇程式語言撰寫，以供支援由虛擬機器40所接收到的系統函數呼叫。平台介面60進一步包括中斷類別68，也是用爪哇程式語言撰寫，它支援電腦系統10和匯流排管理器44的中斷運作。最後，平台介面60包括直接記憶存取（DMA）類別70和記憶體類別72，每一個都是用爪哇程式語言撰寫。DMA類別70和記憶體類別72是本發明的一個特徵，且將在以下做更詳細討論。

五、發明說明 (7)

作業系統原生方法層61包括類別68、70、72和44的作業系統方法，其係以中央處理單元12特定的語言("本機"語言)撰寫，且和微核心62的較低階運作接口。這些方法包括中斷原生方法78、DMA原生方法80和記憶原生方法82。方法80和82將在以下做更詳細討論。

微核心62由支援中央處理單元12必要低階硬體功能的本機語言軟體元件所組成，例如資源配置、中斷處理、保全等。特別地，微核心62包含多個核心函數74，包括線管理、例外、時序、實體記憶管理、硬體中斷處理、平台控制、程序管理、程式館管理、輸入／輸出支援和監控函數。微核心62進一步包括除錯函數76。函數74和76可由昇陽微系統公司市場上可買到的合唱團(Chorus)微核心執行。但，也可以用其他的微核心，如習知該項技藝人士所熟知。

平台相關層34最後的元件是啟動介面64。啟動介面64用以供電腦系統10最初打開電源時，將軟體載入到記憶體14(圖1)之內並設定初值。啟動介面64可載入儲存在，例如軟式磁碟、大量儲存24(圖1)之上的軟體，或在網路上經由輸入裝置18以電腦資料載波形式所接收到的軟體。啟動介面64不構成本發明的一部份，將不進一步討論。

現在參考圖3，在其中顯示說明記憶和DMA類別52、70和72的架構之圖解。記憶和DMA類別52、70和72有下列設計目標：

- 用戶端軟體元件如驅動程式的平台獨立類別，以存

五、發明說明 (8)

取(讀或寫)記憶體：雖然能夠支援多種不同類型的平台，驅動程式將總是使用這些類別的相同平台獨立方法，且只用這些方法，來讀和寫記憶體。發展以爪哇程式語言撰寫，完全與特定平台無關，的裝置驅動程式，是本發明的一個主要目標。

• 建構記憶體物件的匯流排管理器方法：本發明將裝置平台細節壓縮在一撮匯流排管理器中，其建構裝置驅動程式的巨大陣列之記憶體物件，以用來存取記憶體。記憶體類別允許匯流排管理器建構一總稱記憶體物件，它最終地提供對三種基本型態記憶體中之一(如匯流排管理器所描述)的存取：

- 一般的記憶體，藉由來自中央處理單元的標準載入和儲存運作存取。此記憶體可由中央處理單元直接地定址(也就是實體記憶體)，或可經過一記憶體管理單元(MMU)間接地存取(也就是虛擬記憶體)。

- 輸入／輸出記憶體，藉由來自中央處理單元的標準載入和儲存運作存取，但可能也需要完成其他的運作以維持輸入／輸出記憶體與主記憶體一致。再一次，此記憶體可由中央處理單元直接地定址(也就是實體記憶體)，或可經過一MMU間接地存取(也就是虛擬記憶體)。

- 輸入／輸出埠記憶體，經由特別的輸入／輸出指令存取(例如在以英代爾x86為基礎之平台的in和out指令)。

• 匯流排管理器的方法，以設定DMA代表用戶端驅動

五、發明說明 (9)

程式。

- 記憶體和DMA類別本身之爪哇程式語言部分的平台獨立性。
- 支援虛擬記憶體和多重位址空間。此外，所支援的虛擬記憶體必須足夠普遍以包括沒有MMU的平台。在此情況下，虛擬位址將與實體位址相同，且平台將只提供一個虛擬位址空間。即便正直接地存取實體記憶體，裝置驅動程式將總是以為正在存取虛擬記憶體，即使在沒有MMU且只有一個虛擬位址空間的系統上。
- 足夠豐富的記憶體模型。
- 藉由原生方法，對微核心通用的介面，如圖2所示。新的記憶體和DMA類別設計成與一通用的微核心接口。

很重要的是特別注意記憶體和DMA類別暫時不提供另外一個作業系統記憶體管理子系統的實施。相反地，管理虛擬記憶體、實體記憶體、輸入／輸出記憶體和DMA的功能是由微核心提供。記憶體和DMA類別透過原生方法在此種微核心上進行呼叫，但這些類別影響該功能，而非重新創造它。

類別更詳細的討論顯示在圖3，在以下陳述。

Memory(記憶體)抽象類別

在最高階層，記憶和DMA類別的每一個最終都是Memory抽象類別的子類別，它只有如基底位址、長度、和限制(存取和配置兩者)等一般屬性的抽象。在其下面，最後記

五、發明說明 (10)

憶體不是可藉中央處理單元存取(也許並非總是直接地，如在實體記憶體的情形下)，由此而是主-記憶體(MainMemory)，或經由DMA在中央處理單元外部存取，由此而是DMAMemory。

記憶體-描述子(MemoryDescriptor)類別

具有一非抽象且立即的Memory子類別，成為在真正的記憶體物件被建構之前，對代表記憶體有用的基本特性，例如基底位址和長度。除了是例示的，MemoryDescriptor類別並不提供Memory所提供之外的任何額外功能。這對匯流排管理器在抽象記憶體實際配置之前代表它是有用的。

MainMemory抽象類別

MainMemory抽象類別只描述對管理快取的抽象方法，它是對其子類別唯一共通的功能。最後在架構中(在實體-記憶體[PhysicalMemory]、虛擬-記憶體[VirtualMemory]和埠-輸出入-記憶體[PortIOMemory])，實現這些抽象快取管理方法。重要的是注意快取管理高度平台相關的，且因此這些實施的每一個最終靠啟動微核心來清除快取、設定快取模式等。因此，裝置驅動程式使用MainMemory的平台獨立方法對全部快取函數有完全的存取。

MainMemory不是抽象類別可存取-記憶體(AccessibleMemory)，代表能直接地由中央處理單元存取的所有記憶體，就是PhysicalMemary類別，代表一般總是不能直接地由中央處理單元存取的實體記憶體。

DMAMemory類別

五、發明說明 (11)

DMAMemory類別提供設定DMA對映和進行解除對映到實體記憶體上(最後由微核心在這些類別下執行來完成)的方法。最終會決定讓微核心設定DMA對映，因為所要支援DMA硬體的完整範圍將不會適合在DMAMemory類別階層去加以摘要出來，因為這樣做將會再創造微核心必然已經支援的功能，且因為這很可能無法發展出一足夠一般化的方案，處理DMA硬體所有想像得到的樣式。

特別注意DMAMemory物件與微核心替它設定的DMA位址是否為透過某些IOMMU對映到實體DMA位址的虛擬DMA位址無關，也與DMA位址是否為實體DMA位址無關。這些位址單純只是DMA位址，且這些對映對DMAMemory是透明的，且完全由微核心維護。

裝置驅動程式能因此使用以下詳細描述DMAMemory的方法獲得一基底位址，它可傳遞到DMA控制器，藉此讓裝置驅動程式以完全平台獨立的方式提供完整的DMA功能。

AccessibleMemory抽象類別

AccessibleMemory是由匯流排管理器給驅動程式的，所以它們總是能以平台獨立的方式存取記憶體，因此允許它們可攜帶穿越所有的平台。AccessibleMemory如此定義平台獨立的抽象方法，例如setByte()。驅動程式應該只使用摘錄在AccessibleMemory中的方法，或由AccessibleMemory的上層類別，即是MainMemory、和Memory，所摘錄或實施的方法。在AccessibleMemory下面的類別，PhysicalMemory和DMAMemory也一樣，應該只由

五、發明說明 (12)

了解平台細節的匯流排管理器存取。這樣子，平台細節將不會進入驅動程式碼之內。AccessibleMemory不是由中央處理單元發出標準載入和儲存指令存取，由此而VirtualMemory，就是它由中央處理單元發出特別的輸入／輸出埠指令存取，由此而是PortIOMemory。

PhysicalMemory類別

在此記憶體模式中，PhysicalMemory是不可存取的，因為即使在沒有MMUs的系統上，中央處理單元實際上存取正好一對一對映到實體記憶體的虛擬記憶體(不是實體記憶體)而維護了抽象性。此抽象性因而提供了處理具有或沒有MMUs系統一種相同的一般性方法。由於PhysicalMemory類別是MainMemory抽象類別的子類別，PhysicalMemory提供了在MainMemory中摘錄出快取管理方法的實施。

PhysicalMemory類別允許匯流排管理器指出實體位址的範圍，也許是對裝置驅動程式的暫存器。雖然，靠它本身，由於不能存取PhysicalMemory，所以它沒有用。但是，有VirtualMemory和DMAMemory的方法可用來分別地將此實體記憶體對映到虛擬記憶體或DMA。

VirtualMemory抽象類別

VirtualMemory抽象類別包含用來設定對映到PhysicalMemory物件的方法(對映最終由微核心設定)。如上述，即使在沒有MMUs的系統上，VirtualMemory物件一對一對映到PhysicalMemary物件。VirtualMemory類別也有用來配置特定數量虛擬記憶體(最後也由微核心完成)的

五、發明說明 (13)

方法，而沒有描述任何特定的實體記憶體。甚至有一方法，它接著允許一個人獲取VirtualMemory物件下潛在的PhysicalMemory物件的陣列，這在最初PhysicalMemory物件沒有傳遞來建構VirtualMemory物件時尤其有用。匯流排管理器有可能將DMA設定到虛擬記憶體，因此，只要給予VirtualMemory物件，就能獲取潛在的PhysicalMemory物件的陣列，然後傳遞到DMAMemory的方法來設定DMA。VirtualMemory類別也支援在設定DMA運作之前，能在獲取潛在的實體記憶體參數之前使用的lock()和unlock()方法。如以下陳述的細節，甚至有lockContiguous()方法，它確使所對映潛在的實體頁是連續的，它在沒有進行散播-聚集之方法如IOMMU的平台上是必須的。當然，它是由了解平台特性的特定匯流排管理器決定，以決定什麼方法對特定的平台是需要的。

VirtualMemory抽象類別也滿足快取管理方法(最後由微核心執行)，它由MainMemory摘錄。這些方法，也，因平台的一貫特性(還有匯流排管理器所知道的平台特性)而定，在設定DMA的時候是有用的。

PortIOMemory抽象類別

在VirtualMemory之外其他型態的AccessibleMemory是PortIOMemory，它是為中央處理單元，例如英代爾x86系列，發出特別的埠輸入／輸出運作用的。PortIOMemory抽象類別本身只滿足MainMemory所摘錄的快取管理運作。雖然，在此情況下沒有快取，且這些方法無多大用處。

五、發明說明 (14)

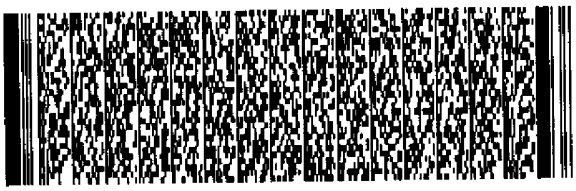
交換-埠-輸出入-記憶體(SwappedPortIOMemory)和非交換-埠-輸出入-記憶體(UnSwappedPortIOMemory)類別

對PortIOMemory的大部份支援在它的二個子類別SwappedPortIOMemory和UnSwappedPortIOMemory中，每一個都滿足了AccessibleMemory所摘錄的所有記憶體存取方法。如名字所隱含的，一個做所寫入資料的位元組交換(以支援在中央處理單元和記憶體之間不同的序法)，而另一個則不做。由匯流排管理器(知道此序法的)決定要例示這兩個的那一個。

要有二個類別，一個交換和一個不交換，的理由是如此不需要有一條件句以檢查在每一個原生方法中記憶體是否為位元組交換。另外，驅動程式不應該必須知道記憶體是否為交換，因為那將與驅動程式完全平台獨立的需求背道而馳。

虛擬-輸出入-記憶體(VirtualIOMemory)和虛擬-正規-記憶體(VirtualRegularMemory)抽象類別

VirtualMemory若非映射到輸入／輸出空間，因而是VirtualIOMemory，即是它不映射到其上，因而是VirtualRegularMemory。這二個不同抽象類別的理由，是在一些平台上對輸入／輸出空間的存取也可能需要運作臉發紅出管道。目前VirtualIOMemory和VirtualRegularMemory兩者都沒有提供功能，而是留給其子類別來提供所有的功能。包括這二個類別的每一個只為了萬一有任何應該在一較高層次摘錄的事物與其子類別共




五、發明說明 (15)

通 。

交換-虛擬-輸出入-記憶體(SwappedVirtualIOMemory)和非交換-虛擬-輸出入-記憶體(UnSwappedVirtualIOMemory類別

與PortIOMemory相似的，VirtualIOMemory有二個子類別，SwappedVirtualIOMemory和UnSwapped VirtualIOMemory，每一個滿足了AccessibleMemory所摘錄的所有的記憶體存取方法。

交換-虛擬-正規-記憶體(SwappedVirtualRegularMemory)和非交換-虛擬-正規-記憶體(UnSwappedVirtualRegularMemory類別

也跟PortIOMemory類似的，VirtualRegularMemory有二個子類別，SwappedVirtualRegularMemory和UnSwappedVirtualRegularMemory，每一個滿足了AccessibleMemory所摘錄的所有的記憶體存取方法。

位址(Addresses)

DMA記憶、實體記憶體和虛擬記憶體全部由來自相同Address抽象類別的位址所定址。由於有顯著範例的系統有非二乘方數目個位元的位址，所以Address類別支援有1和64之間任何數目個位元的位址。

無符號(Unsigned)值(Values)代表Addresses

在類別的內部，Address物件表示成一無符號Values，以及一位元數目的指示。由於爪哇程式語言沒有無符號算術做為它基礎類別的一部份，有必要發展一些簡單有效的

五、發明說明 (16)

演算以做位址的加法與比較。摘錄Address抽象類別主要是因為，依據用來表示位址值的位元數目而定，使用無符號算術的不同演算法，是最有效的。有二個子類別，位址最大到32個位元的Address32，和位址最大到64個位元的Address64。

平台相關和平台獨立

當匯流排管理器建構任何一個記憶和DMA類別時，它必須提供基底位址和長度(也表示成一Address)。為了要這樣做，首先它必須建構這些Address物件的每一個，且如此做需要知道在特定平台上的特定型態位址的位元數目(匯流排管理器知道的平台特性資訊)。

在另一方面，驅動程式總是存取已經由匯流排管理器建構的AccessibleMemory物件，且驅動程式總是藉提供與AccessibleMemory物件基底的偏移量這樣做。所以驅動程式已經與平台特性無關，例如特定位址中的位元數目，AccessibleMemory的每個記憶體存取方法描述偏移量為一整數。這允許記憶體物件最大可達2個十億位元組。雖然目前沒有已知的平台使用更大的記憶體物件，類別中能輕易地加入方法以達到甚至更大的記憶體物件。

對微核心的虛擬位址之表示法

為了單純化，一旦位址被傳遞到微核心，它單純地表示成(在原生方法中)64個位元的無符號整數(數值)。

虛擬位址空間(VirtualAddressSpace)類別

使用本發明的一些機器可能有MMUs，因此支援多重虛擬

五、發明說明 (17)

位址空間是令人期待的，它允許有多個虛擬機器。對一些習知技術的系統，應用程式總是存取與驅動程式在相同虛擬位址空間中的記憶體。但是，對支援在多重位址空間執行的多重虛擬機器來說，有時應用程式將會在與其驅動程式不同的位址空間中。也就是，給定的驅動程式將會存取記憶體物件並會有同步方法做這樣的存取，以處理相同虛擬機器的多重線同時地執行該驅動程式中的可能性。由於虛擬機器沒有有效的溝通機制以提供同步，舉例來說相同的驅動程式在個別的虛擬機器中執行，但可能存取相同的潛在記憶體物件(例如被兩者對映的暫存器)，它遵循特定驅動程式可能總是在相同的虛擬位址空間，而使用那個驅動程式的應用程式可能在不同的位址空間。

為了要定址上述的考慮，提供VirtualAddressSpace類別做為Memory和DMA類別的一部份，且它有一單一的靜態方法，getVasID()，以獲取目前正在執行的線之虛擬位址空間識別。這個識別是從微核心取得的。為了效率，它由getVasID()貯存，因此微核心不必在每次被呼叫。

虛擬位址空間識別的使用

在建構VirtualMemory物件的時候，虛擬位址空間識別是建構子參數中的一個。此識別最後在微核心要求對映虛擬記憶體的時候提供給微核心。它遵循只有微核心指派語意給虛擬位址空間識別的值。

<u>裝置介面類別</u>

以下的資料描述類別50，它構成資料型態的第一程式套

五、發明說明 (18)

件，包括記憶體類別。這些類別是平台獨立的，且因而可以帶入平台獨立的作業系統之爪哇程式語言碼中，如驅動程式，藉此提供完整功能的平台獨立驅動程式。另外，由於它們是平台獨立的，這些類別不因任何平台相關記憶體類別72(圖2)而決定。

附記每個類別的所有公用欄位。

Address(函數名稱)public abstract class Address

Memory和DMA類別的所有記憶體物件使用Address物件做為它們的位址。既然有一些真實的重要的硬體例子有非二乘方數目個位元的位址，支援1和64之間任何數目個位元的位址。在摘錄Address類別時，它擴展為

·Address32，它對在1和32個位元之間的位址提供支援，和

·Address64，它對在33和64個位元之間的位址提供支援。

Address類別定義公用抽象方法，它由Address32和Address64實施：

```
public abstract String toString();
public abstract int intvalue();
public abstract long longValue();
```

在以下除說明，對兩個子類別是共通的，保護的建構子和公用方法之外，Address抽象類別也實現程式套件私用方法，以檢查二個Addresses是否為相同型態(相同數目的位元)。

五、發明說明 (19)

Address(函數名稱)

protected Address (int bits)

Address類別本身只保留位元的數目。位址的值儲存在子類別中。

getBitWidth(函數名稱)

public int getBitwidth()

傳回：

此位址的位元數目(在1和64之間)。

addAddr(函數名稱)

public Address addAddr(Address other) throws InvalidAddressException;

參數：

other-要加到此Address的Address。

傳回：

一Address，其值為此Address與其他Address的總和，且和此Address與其他Address有相同數目的位元。

丟出：

如果其他Address沒有和此Address相同數目的位元，或這兩個Addresses值的無符號總和溢出這二個Address共同的位元數目所能代表之最大無符號值，丟出InvalidAddressException。

包括這個方法是為了完整性，以便用戶端能組成一Address基底與一偏移量之和(也以Address表示)的位址，而不必用戶端實施它自己的無符號算術演算法。然而，由

五、發明說明 (20)

於用戶端總是藉由只描述偏移量來存取(讀或寫)記憶體物件，讓記憶體類別計算存取的位址(基底位址和偏移量的總和)，用戶端並不真的需要使用這個方法。

此方法啟動潛在Address子類別的相關方法。

addrCompare(函數名稱)

public int addrCompare(Address other) throws InvalidAddressException;

參數：

other -要與這個位址比較的Address。

傳回：

-1 如果此Address的值比其他Address的值更大。

0- 如果此Address的值與其他Address的值相等。

-1- 如果此Address的值比其他Address的值更小。

丟出：

如果其他Address沒有和此Address相同數目的位元，丟出InvalidAddressException。

用戶端藉描述偏移量(也表示成一Address)存取(讀或寫)記憶體物件。這個方法讓用戶端決定這個偏移量是否是有效的(比記憶體物件的長度小)，不用實施它自己的無符號算術演算法。然而，用戶端不是絕對必要使用這個方法，因為記憶體類別存取方法已經做範圍檢查。

這個方法啟動潛在Address子類別的相關方法。

Address32(函數名稱)

public class Address32 extends Address

五、發明說明 (21)

Address32類別擴充抽象Address類別，且對位元數目包含在1和32之間的位址提供支援。這裡描述的公用方法，由建構方法和Address抽象類別的抽象公用方法之實施組成。

應該注意的是Address32類別也實施程式套件私用方法以做無符號32位元加法和無符號32位元比較；它比公用方法，Address類別的addAddr()和addrCompare()，的內部計算要更有效率。這些程式套件私用方法因為不必配置Address物件來回傳，所以比較有效率。

Address32(函數名稱)

public Address32(int addrvalue, int bitwidth)
throws InvalidAddressException

public Address32(int addrvalue)

public Address32(long addrvalue, int bitwidth)
throws InvalidAddressException

參數：

addrvalue-Address的真實值。如果描述了位元長度，且它比32小，僅最小有效位元長度數目的位元用來做為Address的值。

bitwidth-Address的位元數目(在1和32之間)。

丟出：

如果描述的位元長度不包含在1和32之間，丟出Invalid Address Exception。

第一個建構子允許用戶端建立1和32位元之間的位址的

五、發明說明 (22)

一般型態。第二個建構子提供來方便建構更通用的32位元Address。第三個建構子提供來方便，在Address的值是長整數時。在這情況下，雖然，只有最小有效的32個位元(或更少)被使用。

to String(函數名稱)

public String to String()

傳回：

代表此Address32物件的字串。

intValue(函數名稱)

public int intvalue()

傳回：

位址值最小有效的32個位元。

long Value(函數名稱)

public long Value()

傳回：

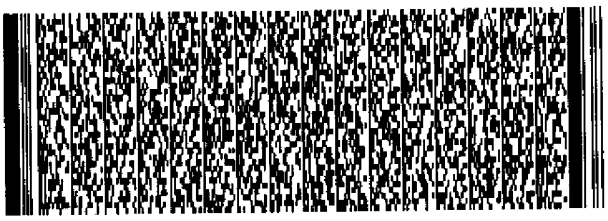
位址值最小有效的64個位元。

Address64(函數名稱)

public class Address 64 extends Address

Address64類別擴充抽象Address類別，並提供對位元數目包含在33和64之間位址的支援。這裡描述的公用方法，由建構方法和Address抽象類別的抽象公用方法之實施組成。

應該注意的是Address64類別也實施程式套件私用方法以做無符號64位元加法和無符號64位元比較；它比公用方



五、發明說明 (23)

法，Address類別的addAddr()和addrCompare()，的內部計算要更有效率。這些程式套件私用方法因為不必配置Address物件來回傳，所以比較有效率。

Address64(函數名稱)

public Address64(long addrvalue, int bitwidth) throws InvalidAddressException public Address64 (long addrvalue)

參數：

addrvalue-Address的真實值。如果描述了位元長度，且它比64小，僅最小有效位元長度數目的位元用來做為Address的值。

bitwidth-Address的位元數目(在33和64之間)。

丟出：

如果描述的位元長度不包含在33和64之間，丟出InvalidAddressException。

第一個建構子允許用戶端建立33和64位元之間的位址的一般型態。第二個建構子提供來方便建構更通用的64位元Address。

toString(函數名稱)

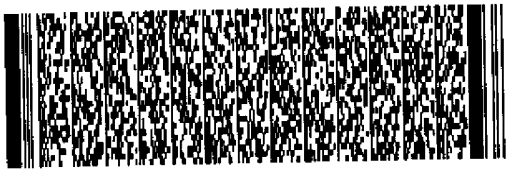
public StringtoString

傳回：

代表此Address64物件的字串。

intValue(函數名稱)

public int intvalue()



五、發明說明 (24)

傳回：

位址值最小有效的32個位元。

longValue(函數名稱)

public long longvalue()

傳回：

位址值最小有效的64個位元。

AddressSpace(函數名稱)

public abstract class AddressSpace

每個匯流排管理器管理一組位址空間。記憶體類別本身有三個位址空間(見VirtualAddressSpace、PhysicalAddressSpace和DMAAddressSpace)，它們全部特定的匯流排管理器，稱為PlatformBusManager的最高階層匯流排管理器，的一部份。除了這些位址空間之外，有其他匯流排管理器可以例示的其他位址空間。所有此種位址空間是AddressSpace摘要類別的子類別。

AddressSpace(函數名稱)

protected AddressSpace (ExpansionBus b, String n)

參數：

b-描述有此特定&位址空間的匯流排。

n-命名此位址空間的字串。

此建構子只是以所提供的值設定保護變數的初值。

也提供了保護設定初值常式，以使子類別能設定位址空間的位元-大小和該位址空間的識別。這兩個初始值可由子類別經由原生方法獲得。在可能有特定型態多重位址空

五、發明說明 (25)

間的情況下(例如虛擬)，識別是由微核心唯一地描述，以便它可明確地識別。使用位址空間位元數目的初始值，AddressSpace類別計算和儲存代表最低的和最高的可能位址之Address物件。

getID(函數名稱)

public int getID()

傳回：

描述特定位址空間的識別(同上)。

getName(函數名稱)

public String getName()

傳回：

用來設定建構子中此位址空間初值的字串。

getExpansionBus(函數名稱)

public String getName()

傳回：

用來設定建構子中此位址空間初值的匯流排。

getLowestAddress(函數名稱)

public String getLowestAddress()

傳回：

在此位址空間中可能的最小位址值。

getHighestAddress(函數名稱)

public String getHighestAddress()

傳回：

在此位址空間中可能的最高位址值。

五、發明說明 (26)

MemoryConstraints(函數名稱)

public class MemoryConstraints

當匯流排管理器建立記憶體物件時，它可能需要描述記憶體物件配置和後來驅動程式對記憶體物件存取兩者的限制。例如，一些DMA控制器可能對它們能發出位址的範圍有限制，以迫使DMA物件(記憶體物件)只配置在該位址範圍。當成存取限制的一個例子，一些暫存器可能只當成位元組存取，使得任何其他的存取可能引起某種系統失敗。

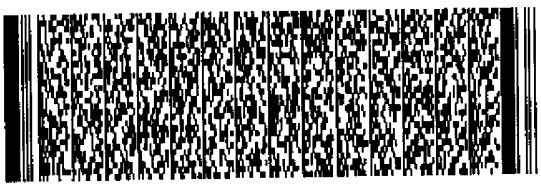
MemoryConstraints物件壓縮此種配置與存取限制，以便它們能一起傳送到記憶體物件的建構子。

MemoryConstraints類別包含用來描述所有各種限制的建構子，和幾個用來獲取並設定每一各種限制之方法。這個類別也包含程式套件私用方法，以檢查所要求的存取是否滿足限制，及所描述的配置是否與限制相符。

定義了下列各項常數：

```
public static final int ALIGN_BYTE = 0;
public static final int ALIGN_SHORT = 1;
public static final int ALIGN_INT = 2;
public static final int ALIGN_LONG = 3;
public static final int ALIGN_CACHE = 4;
public static final int ALIGN_PAGE = 5;
public static final int READONLY = 0;
public static final int READWRITE = 1;
```

MemoryConstraints(函數名稱)



五、發明說明 (27)

public MemoryConstraints (Address alloc_min_a, Address alloc_max_a, int alloc_align, int cache, boolean lock, boolean waitformem, boolean swapped, boolean io, boolean virtual, int access_min_s, int access_max_s, int access_align, int rw) throws Invalid Address Exception

參數：

alloc_min_a-配置的最小位址。

alloc_max_a-配置的最大位址。

alloc_align-配置的基底位址必須全部是0的最小有效位元的最小數目。一個0值指示對配置沒有對齊的限制。alloc_align支援的最大值是16(64K位元組對齊)。

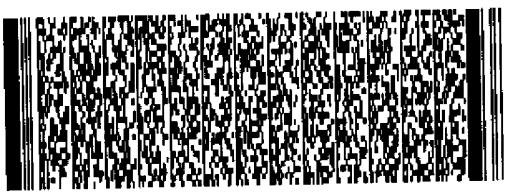
cache-指示記憶體的快取(MainMemory.CACHE_MODE_DEFAULT, MainMemory.CACHE_MODE_INHIBITED.MainMemory.CACHE_MODE_WRITE_THROUGH,MainMemory.CACHE_MODE_COPY_BACK之中的一個)。

lock-指示是否要鎖定對記憶體的對映。

swapped-如果為真，指出將建構一種型態的記憶體，其中將會完成位元組-交換。

io-如果為真，指出將建構一種型態的記憶體，它是輸入/輸出記憶體(不是PortIOMemory就是VirtualIOMemory)。

virtual-如果為真，指出將建構一種型態的記憶體，它是VirtualMemory。



五、發明說明 (28)

waitformem-指示如果有資源問題，是否要無限期地等待記憶體的配置。

access_min_s-可以位元組存取記憶體的最小大小的對數(基數2)。因此，0指出可存取單一位元組。

access_max_s-可以位元組存取(最大到3)記憶體的最大大小的對數(基數2)。

access_align-一存取位址的最低有效位元必須全部是0的最小數目。因此一0值指示對存取沒有對齊限制。access_align支援的最大值是16(64K位元組對齊)。

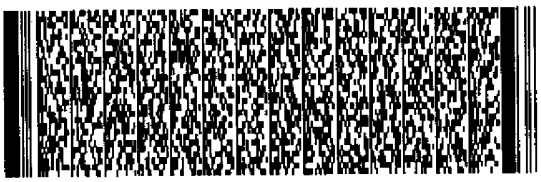
rw-READONLY(0)指示唯讀存取，而READWRITE(1)指示讀-寫存取。

丟出：

如果alloc_min_a和alloc_max_a在它們的位元數目上不一致，或如果alloc_min_a的值不小於alloc_max_a的值，丟出InvalidAddressException。

配置對齊支援的最大大小是ALIGN_PAGE(5)，而最小是，當然，ALIGN_BYTE(0)，因此如果所描述的alloc_align比ALIGN_PAGE(5)更大，它被設定成5，或如果所描述的alloc_align比ALIGN_BYTE(0)更小，它被設定成0。

如果所描述的access_min_s比0小，它被設定成0(1位元組)，或如果所描述的access_minds比3大，它被設定成3(8位元組，一長整數)。同樣地，如果所描述的access_max_s比0更小，它被設定成0(1位元組)，或如果




五、發明說明 (29)

所描述的access_max_s比3大，它被設定成3(8位元組，一長整數)。如果access_max_a仍然比access_min_a小，access_max_a被設定成access_min_a的值。

所支援存取對齊最大的大小是ALIGN_LONG(3)，而所支援最小的是，當然，ALIGN_BYTE(0)，因此如果所描述的access_align比ALIGN_LONG(3)更大，它被設定成3，或如果所描述的access_align比ALIGN_BYTE(0)更小，它被設定成0。

如果rw不被設定成READONLY，它就被設定成READWRITE。

getAllocMinAddr(函數名稱)

public Address getAllocMinAddr()

傳回：

最小配置的Address，alloc_min_a，由建構子或setAllocAddrs()方法描述。

getAllocMaxAddr(函數名稱)

public Address getAllocMaxAddr()

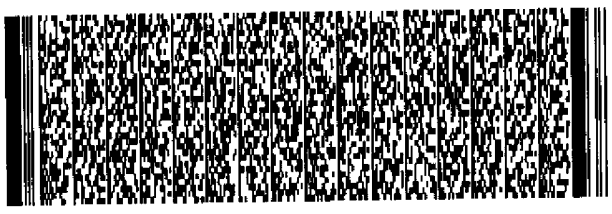
傳回：

最大配置的Address，alloc_max_a，由建構子或setAllocAddrs()方法描述。

setAllocAddrs(函數名稱)

public void setAllocAddrs(Address alloc_min_a, Address alloc_max_a) throws InvalidAddressException

參數:



五、發明說明 (30)

alloc_mill_a-配置的最小位址。

alloc_max_a-配置的最大位址。

丟出：

如果alloc_min_a和alloc_max_a在它們的位元數目上不一致，或如果alloc_mill_a的值不小於alloc_max_a的值，丟出InvalidAddressException。

getAllocAlign(函數名稱)

public int getAllocAlign()

傳回：

配置的對齊，alloc_align，如建構子或setAllocAllign()方法所描述。注意這個值可能不同於所描述的值(見建構子或setAllocAlign方法的描述)。

setAllocAlign(函數名稱)

public void setAllocAlign(int alloc_align)

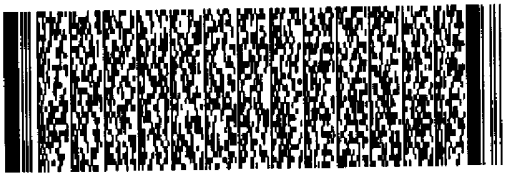
參數:

alloc_align-基底位址的最低有效位元必須全部是0的最小數目。因此一0值指示對配置沒有對齊限制。

alloc_align支援的最大值是16(64K位元組對齊)。

配置對齊支援的最大大小是ALIGN_PAGE(5)，而最小是，當然，ALIGN_BYTE(0)，因此如果所描述的alloc_align比ALIGN_PAGE(5)更大，它被設定成5，或如果所描述的alloc_align比ALIGN_BYTE(0)更小，它被設定成0。

getCacheMode(函數名稱)



五、發明說明 (31)

public int getCacheMode()

傳回：

快取模式，cache，如建構子setMiscAlloc()方法所描述。

getLocked(函數名稱)

public boolean getLocked()

傳回：

指示在配置時對映是否將鎖定的，lock，如建構子或setMiscAlloc()方法所描述。

getWaitforMem(函數名稱)

public boolean getwaitFormMem()

傳回：

指示在配置時是否要無限期地等待記憶體，waitformem，如建構子或setMiscAlloc()方法所描述。

getSwapped(函數名稱)

public int getSwapped()

傳回：

參數，swapped，如建構子或setMiscAlloc()方法所描述。

getIO(函數名稱)

public int getIO()

傳回：

參數，io，如建構子或setMiscAlloc()方法所描述。

getVirtual(函數名稱)

五、發明說明 (32)

public int getVirtual()

傳回：

參數，virtual，如建構子或setMiscAlloc()方法所描述。

setMiscAlloc(函數名稱)

public void setMiscAlloc(int cache, boolean lock, boolean waitformem, boolean swapped, boolean io, boolean virtual)

參數：

cache-指示記憶體的快取(MainMemory.CACHE_MODE_DEFAULT, MainMemory.CACHE_MODE_INHIBITED, MainMemory.CACHE_MODE_WRITE_THROUGH或MainMemory.CACHE_MODE_COPY_BACK之中的一個)。

lock-指示是否要鎖定對記憶體的對映。

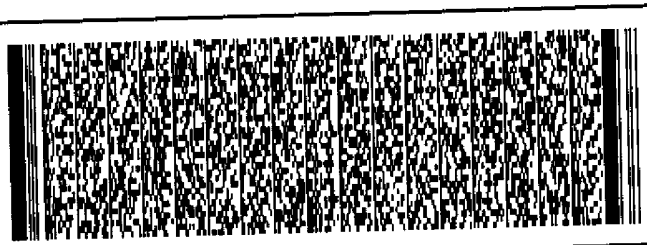
waitformem-指示如果有資源問題，是否要無限期地等待記憶體的配置。

swapped-如果為真，指出將建構一種型態的記憶體，其中將會完成位元組-交換。

io-如果為真，指出將建構一種型態的記憶體，它是輸入/輸出記憶體(不是PortIOMemory就是VirtualIOMemory)。

virtual-如果為真，指出將建構一種型態的記憶體，它是VirtualMemory。

getAccessMinSize(函數名稱)



五、發明說明 (33)

public int getAccessMinSize()

傳回：

存取的最小大小，access_min_s，如建構子或setAccessSizes()方法所描述。注意這個值可能不同於所描述的值(見建構子或setAccessSize()方法的描述)。

getAccessMaxSize(函數名稱)

public int getAccessMaxSize()

傳回：

存取的最大大小，access_max_s，如建構子或setAccessSizes()方法所描述。注意這個值可能不同於所描述的值(見建構子或setAccessSizes()方法的描述)。

setAccessSizes(函數名稱)

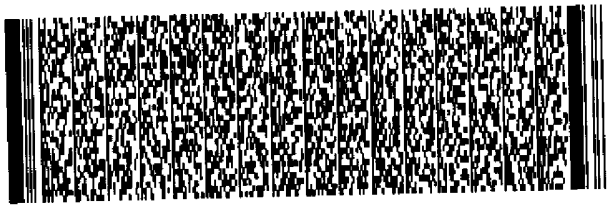
public void setAccessSizes(int access_min_s, int access_max_s)

參數:

access_min_s-可以位元組存取記憶體的最小大小的對數(基數2)。因此，0指出可存取單一位元組。

access_max_s-可以位元組存取(最大到3)記憶體的最大大小的對數(基數2)。

如果所描述的access_min_s比0小，它被設定成0(1位元組)，或如果所描述的access_min-s比3大，它被設定成3(8位元組，一長整數)。同樣地，如果所描述的access_max_s比0更小，它被設定成0(1位元組)，或如果所描述的access_max_s比3大，它被設定成3(8位元組，一



五、發明說明 (34)

長整數)。如果access_max_a仍然比access_min_a小，access_max_a被設定成access_min_a的值。

getAccessAlign(函數名稱)

public void getAccessAlign()

傳回：

存取的對齊，access_align，如建構子或setAccessAlign()方法所描述。注意這個值可能不同於所描述的值(見建構子或setAccessAlign方法的描述)。

setAccessAlign(函數名稱)

public void setAccessAlign(int access_align)

參數：

access_align-一存取位址的最低有效位元必須全部是0的最小數目。因此一0值指示對存取沒有對齊限制。access_align支援的最大值是16(64K位元組對齊)。

存取對齊支援的最大大小是ALIGN_LONG(3)，而最小是，當然，ALIGN_BYTE(0)，因此如果所描述的access_align比ALIGN_LONG(3)更大，它被設定成3，或如果所描述的access_align比ALIGN_BYTE(0)更小，它被設定成0。

getReadWrite(函數名稱)

public int getReadwrite()

傳回：

存取的模式，不是READONLY，就是READWRITE。注意此值可能不同於所描述的值(見建構子或setReadWrite方法

五、發明說明 (35)

的描述)。

setReadWrite(函數名稱)

public void setReadwrite(int rw)

參數:

rw-READONLY指示唯讀存取，而READWRITE指示讀-寫存取。

rw若不是被設定成READONLY，就是設定成READWRITE。

Memory(函數名稱)

public abstract class Memory

所有Memory和DMA類別的記憶體物件最後都是Memory抽象類別的子類別。Memory抽象類別定義二個抽象方法：

public abstract String toString();

除了以下描述的對所有記憶體物件共通的基本公用方法之外，Memory類別也實施程式套件的私用方法以檢查存取的有效性(以記憶體物件中的偏移量、大小、模式(讀或寫)、和位元組的計數描述)，且為獲取存取的位址，給予物件的一偏移量。

Memory(函數名稱)

protected Memory (Address ba, Address len, MemoryConstraints mc) throws InvalidAddressException

protected Memory ()

protected Memory (Memory superrange, Address offset, Address len) throws

五、發明說明 (36)

InvalidAddressException, AllocationException

參數:

ba-Memory物件的基底位址。

len-Memory物件的以位元組計算的長度。

mc-Memory物件的限制。

丟出:

如果在將基底位址ba加到長度len的時候發生溢位，丟出InvalidAddressException。

在第三個保護的建構子的情形中，如果超範圍Memory物件是無效的，丟出AllocationException。

第一個保護的建構子只是計算Address的終點(ba+len)，將所有的參數連同所計算Address的終點一起儲存在這個類別的保護變數中，初始化記憶體成為有效，並配置保護的內部資料結構。

第二個保護的建構子使用於潛在Memory物件在第一次建構時不能設定初值(例如VirtualMemory的基底位址要直到記憶體被對映時才能知道)。提供一種保護的方法以在稍後設定Memory物件初值，如此完成應該被第一個保護的建構子用來設定初值。

第三個保護的建構子用在被建構的記憶體物件是一已存在記憶體物件的一個子範圍。記憶體Address和Address終點根據與Memory物件超範圍的偏移量計算，且新的Memory物件被結合成超範圍的一個子範圍，並初始化成有效的。也配置了保護的內部資料結構區域。

五、發明說明 (37)

getMemBaseAddr(函數名稱)

public Address getMemBaseAddr()

傳回：

Memory物件的基底Address，如果記憶體是有效的，否則0。

getMemLength(函數名稱)

public Address getMemLength()

傳回：

Memory物件以位元組計算的長度，如果記憶體是有效的，否則0。

getMemEndAddr(函數名稱)

public Address getMemEndAddr()

傳回：

Memory物件的Address終點，如果記憶體是有效的，否則0。

getMemValid(函數名稱)

public boolean getMemvalid()

傳回：

記憶體是否有效的指示。

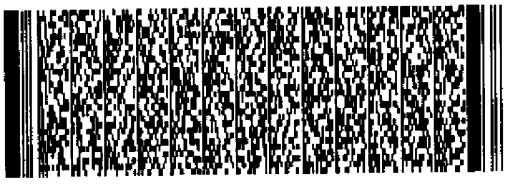
getMemConstraints(函數名稱)

public MemoryConstraints getMemConstraints ()

傳回：

記憶體的限制，如果記憶體是有效的，否則0。

getSuperRange(函數名稱)



五、發明說明 (38)

public Memory getSuperRange()

傳回：

此Memory物件的超範圍之Memory物件，如果它是一子範圍；否則傳回0。

getSubRanges(函數名稱)

public Vector getSubRanges()

傳回：

此Memory物件的子範圍之Memory物件的一Vector。如果沒有子範圍或如果這個Memory物件是無效的，傳回0。

MemoryDescriptor(函數名稱)

public class MemoryDescriptor extends Memory

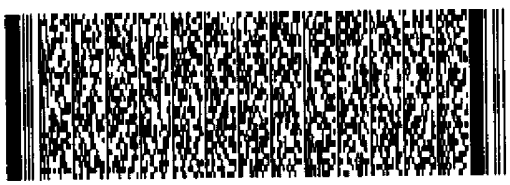
所有真的例示d記憶體物件最後是平台相關，並因此而由sun.javaos.memory程式套件中的程式碼建構。雖然，有時它對驅動程式或匯流排管理器能夠例示一記憶體物件，將在稍後配置的記憶體的描述，是有用的。

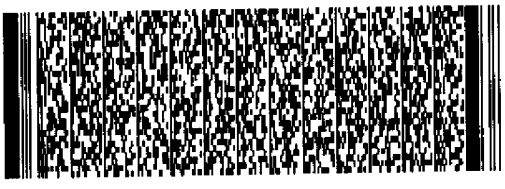
MemoryDescriptor類別，作為可例示的的Memory子類別，符合需要。與MemoryDescriptor物件有關的是位址空間，由一些匯流排管理器管理，在其中定義此記憶體的描述。

MemoryDescriptor(函數名稱)

public MemoryDescriptor(Address ba, Address len,
MemoryConstraints mc,
AddressSpace as, String name) throws
InvalidAddressException

public MemoryDescriptor(Address ba, Address len,





五、發明說明 (39)

AddressSpace as, String name) throws InvalidAddressException

參數:

ba-Memory物件的基底位址。

len-Memory物件的以位元組計算的長度。

mc-Memory物件的限制。

as-這個記憶體物件存在的AddressSpace。

name-用來識別這個記憶體物件的一些名字。當匯流排管理器在配置MemoryDescriptors的一個陣列時這是有用的，描述驅動程式必須對映的所有各種記憶體物件，因為它允許驅動程式識別每個記憶體物件。

丟出：

如果超類別(Memory)的建構子丟出InvalidAddressException，則丟出InvalidAddressException。換句話說，此建構子不捕捉例外。

第一個建構子單單只呼叫超類別，Memory，的建構子，除了AddressSpace之外使用所有的參數，然後儲存AddressSpace在一私用變數中。

第二個建構子為了方便而提供，且與第一個建構子相同，除了零被傳送到Memory建構子的mc(MemoryConstraints)參數。

toString(函數名稱)

public String toString()

五、發明說明 (40)

傳回：

代表此MemoryDescriptor物件的字串。

getAddressSpace(函數名稱)

public AddressSpace getAddressSpace ()

傳回：

The AddressSpace in which this descriptor of Memory exists, as specified in the constructor.

getName(函數名稱)

public String getName()

傳回:

命名這個特別MemoryDescriptor的字串，如建構子中所描述的。

MainMemory(函數名稱)

public abstract class MainMemory extends Memory

代表最後可由中央處理單元存取的記憶體(除了DMAMemory之外的每一個，已經描述過的超-類別Memory、和MainMemory本身)之所有記憶體物件，都是抽象類別MainMemory的子類別。

MainMemory定義下列的抽象方法：

public abstract void setCacheMode(int mode);

public abstract int getCacheMode();

public abstract void flushCache();

MainMemory也定義下列各項常數，以用做實施setCacheMode()的參數和實施getCacheMode()的傳回值：

五、發明說明 (41)

```
public static final int CACHE_MODE_DEFAULT=0x0;
public static final int CACHE_MODE_AMBIGUOUS=0x1;
public static final int CACHE_MODE_INHIBITED=0x2;
public static final int
CACHE_MODE_WRITE_THROUGH=0x3;
public static final int CACHE_MODE_COPY_BACK=0x4;
```

對這個類別實施的唯一方法是保護的建構子。沒有公用方法被實施。

MainMemory(函數名稱)

```
protected MainMemory (Address ba, Address len,
MemoryConstraints mc) throws
InvalidAddressException
protected MainMemory ()
protected MainMemory (MainMemory superrange,
Address offset,Address len)throws
invalidAddressException, AllocationException
```

參數：

ba-Memory物件的基底Address。

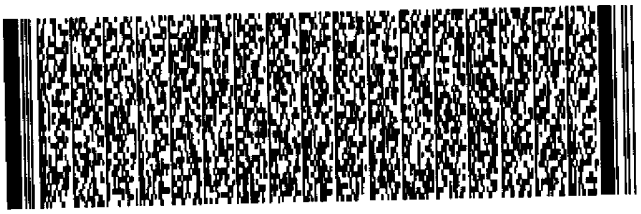
len-Memory物件的以位元組計算的長度。

mc-Memory物件的限制。

丟出：

如果在將基底Address，ba，加到長度，len時發生溢位，丟出InvalidAddressException。

在第三個保護的建構子之情況中，如果超範圍Memory物



五、發明說明 (42)

件是無效的，丟出AllocationException。

這些保護的建構子的每一個只單純地呼叫對應超類別Memory保護的建構子。

AccessibleMemory(函數名稱)

public abstract class AccessibleMemory extends MainMemory

Memory和DMA類別的所有記憶體物件，代表能直接地由中央處理單元存取的記憶體(除了PhysicalMemory和AccessibleMemory本身之外，MainMemory的每個子類別)，是抽象類別AccessibleMemory的子類別。

平台獨立程式碼，例如驅動程式，總是經由AccessibleMemory的方法存取Memory，不知道或在意潛在的記憶體物件可能是什麼。

AccessibleMemory定義下列保護的抽象方法，它是公開地定義的存取方法之內部實施：

```
protected abstract int ncksum(long adr, long len);
protected abstract void nSetByte(long adr, byte
x);
protected abstract void nSetShort(long adr, short
x);
protected abstract void nSetInt(long adr, int x);
protected abstract void nsetLong(long adr, long
x);
protected abstract void nSetBytes(long adr, byte
```

五、發明說明 (43)

```
bytes[],int bytesOffset, intlength);
protected abstract void nsetIntArray(long adr,
byte bytes[],int bytesOffset, intlength_int);
protected abstract byte nGetByte(long adr);
protected abstract short nGetShort(long adr);
protected abstract int nGetInt(long adr);
protected abstract long nGetLong(long adr);
protected abstract void nGetBytes(long adr, byte
bytes[],int bytesOffset, int length);
protected abstract void nGetIntArray(long adr,
byte bytes[],int bytesOffset, int length_int);
protected abstract int nRegisterSource(long adr1,
long len);
protected abstract void nCopy(long adr2, long len,
int copyID);
```

除了建構子之外，沒有定義其他非公用方法。

AccessibleMemory未實施在架構中在它之上的二個類別，MainMemory和Memory，之任何抽象方法。

應該要注意以下的每一個存取方法檢查無效的存取，而且如果有違反，記錄一系統訊息，且不執行該存取。不簡單地讓這些方法丟出例外的理由是那樣子，當驅動程式被移植到新的Memory類別的時候，所有在各種驅動程式中的目前呼叫-且在其中有許多-將需要修改以宣告例外丟出。

AccessibleMemory(函數名稱)

五、發明說明 (44)

protected AccessibleMemory(Address ba, Address len, MemoryConstraints mc)
throws InvalidAddressException
protected AccessibleMemory ()
protected AccessibleMemory (AccessibleMemory superrange, Address offset,
Address len) throws InvalidAddressException, AllocationException

參數：

ba-Memory物件的基底Address。

len-Memory物件以位元組計算的長度。

mc-Memory物件的限制。

丟出：

如果在將基底Address，ba，加到長度，len時發生溢位，丟出InvalidAddressException。

在第三個保護的建構子之情況中，如果超範圍Memory物件是無效的，丟出AllocationException。

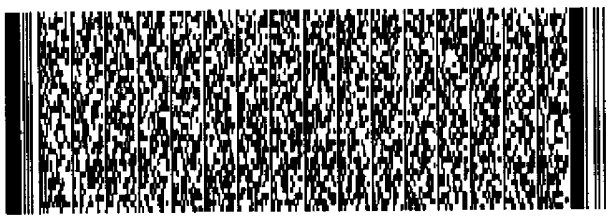
這些保護的建構子的每一個只單純地呼叫對應超類別MainMemory保護的建構子。

setAddress(函數名稱)

public void setAddress(int offset, Address value)

參數：

offset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量。



五、發明說明 (45)

value-值要被寫在存取位置的Address。

註：

如果記憶體不是有效的(定義在Memory類別中)、如果偏移量(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

cksum(函數名稱)

public int cksum(int offset, int len)

參數：

offset-由記憶體物件的基底到檢和計算要開始處的偏移量。

len-檢和要計算的位元組長度。

傳回:

所計算出的檢和(模數 0x10000)，或0，如果不允許存取(見註)。

註：

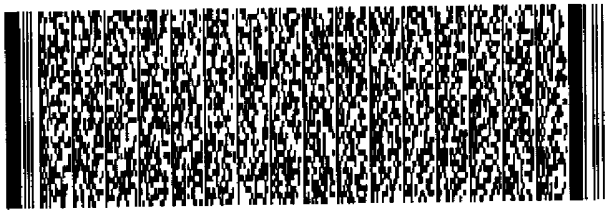
如果記憶體不是有效的(定義在Memory類別中)、或如果偏移量(加存取的長度，len)超過記憶體物件的終點，則不做檢和並記錄一系統訊息。

setByte(函數名稱)

public void setByte(int offset, byte x)

參數：

offset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量。



五、發明說明 (46)

x-要寫在存取位置的值。

註：

如果記憶體不是有效的(定義在Memory類別中)、如果偏移量(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

setShort(函數名稱)

public void setShort(int offset, short x)

參數：

offset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量。

x-要寫在存取位置的值。

註：

如果記憶體不是有效的(定義在Memory類別中)、如果偏移量(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

setInt(函數名稱)

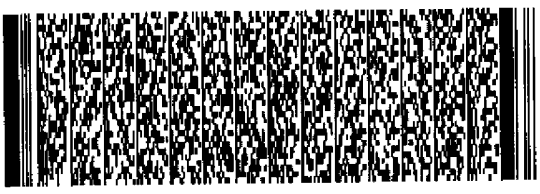
public void setInt(int offset, int x)

參數：

offset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量。

x-要寫在存取位置的值。

註：



五、發明說明 (47)

如果記憶體不是有效的(定義在Memory類別中)、如果偏移量(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

setLong(函數名稱)

public void setLong(int offset, long x)

參數：

offset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量。

x-要寫在存取位置的值。

註：

如果記憶體不是有效的(定義在Memory類別中)、如果偏移量(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

setBytes(函數名稱)

public void setBytes(int offset, byte bytes[], int bytesOffset, int length)

參數：

offset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量。

bytes-要寫到記憶體位元組的陣列。

bytesOffset-在陣列、位元組中開始的索引。

length-要寫的位元組數目。

五、發明說明 (48)

註：

如果記憶體不是有效的(定義在Memory類別中)、如果bytesOffset為負、如果length不大於0、如果offset(加存取的總大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

setIntArray(函數名稱)

public void setIntArray(int offset, byte bytes[], int bytesOffset, int length_int)

參數：

offset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量。

bytes-要寫到記憶體位元組的陣列。

bytesOffset-在陣列、位元組中開始的索引。

length_int-要寫的整數數目。

註：

如果記憶體不是有效的(定義在Memory類別中)、如果bytesOffset為負、如果length_int不大於0、如果offset(加存取的總大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

getByte(函數名稱)

public byte getByte(int offset)

參數：

五、發明說明 (49)

offset-由記憶體物件的基底到存取(在此情況下是讀)要完成處的偏移量。

傳回:

由存取位置讀取的值。

註:

如果記憶體不是有效的(定義在Memory類別中)、如果offset(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制,則不做存取且記錄一系統訊息。

getShort(函數名稱)

public short getShort(int offset)

參數:

offset-由記憶體物件的基底到存取(在此情況下是讀)要完成處的偏移量。

傳回:

由存取位置讀取的值。

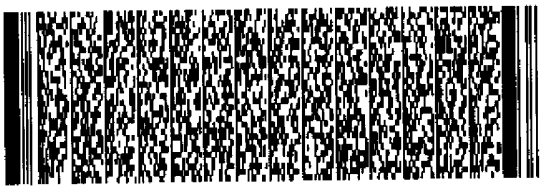
註:

如果記憶體不是有效的(定義在Memory類別中)、如果offset(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制,則不做存取且記錄一系統訊息。

getInt(函數名稱)

public int getInt(int offset)

參數:



五、發明說明 (50)

offset-由記憶體物件的基底到存取(在此情況下是讀)要完成處的偏移量。

傳回:

由存取位置讀取的值。

註:

如果記憶體不是有效的(定義在Memory類別中)、如果offset(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制,則不做存取且記錄一系統訊息。

getLong(函數名稱)

public long getLong(int offset)

參數:

offset-由記憶體物件的基底到存取(在此情況下是讀)要完成處的偏移量。

傳回:

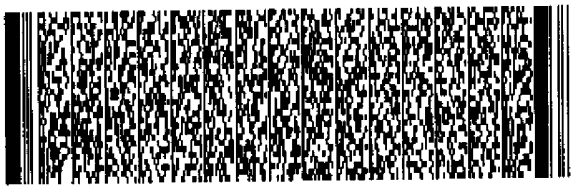
由存取位置讀取的值。

註:

如果記憶體不是有效的(定義在Memory類別中)、如果offset(加存取的大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制,則不做存取且記錄一系統訊息。

get Bytes(函數名稱)

public void getBytes(int offset, byte bytes[], int bytesOffset, int length)



五、發明說明 (51)

參數：

offset-由記憶體物件的基底到存取(在此情況下是讀)要完成處的偏移量。

bytes-要由記憶體讀取位元組的陣列。

bytesOffset-在位元組陣列中開始的索引。

length-要讀的位元組數目。

註：

如果記憶體不是有效的(定義在Memory類別中)、如果bytesOffset為負、如果length不大於0、如果offset(加存取的總大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

getIntArray(函數名稱)

public void getIntArray(int offset, int ints[], int intsOffset, int length_int)

參數：

offset-由記憶體物件的基底到存取(在此情況下是讀)要完成處的偏移量。

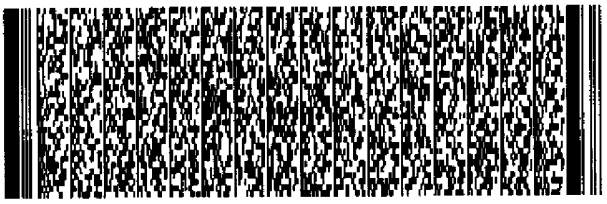
ints-要寫到記憶體之整數的陣列。

intsOffset-在整數的陣列中開始的索引。

length_int-要寫的整數的數目。

註：

如果記憶體不是有效的(定義在Memory類別中)、如果intsOffset為負、如果length_int不大於0、如果



offset(加存取的總大小)超過記憶體物件的終點、或如果違反定義在超類別Memory物件之限制的任何一個限制，則不做存取且記錄一系統訊息。

copy(函數名稱)

public void copy(int offset, AccessibleMemory dest, int destOffset, int length)

參數：

offset-由記憶體物件的基底到存取(在此情況下是讀)要完成處的偏移量。

dest-包含copy的目的之記憶體物件。

destOffset-由記憶體物件的基底到存取(在此情況下是寫)要完成處的偏移量(表示成一Address、short、int或long中之一)。

註：

如果記憶體(來源或目的)不是有效的(定義在Memory類別中)、如果offset(加上對來源與目的存取之總大小)超過記憶體物件的終點、如果違反定義在超類別Memory物件之限制的任何一個限制、或如果兩個Memory物件不是相同的基本型態，則不做存取且記錄一系統訊息。

Platform Interface Classes

以下的資料說明類別72，它構成記憶體類別的第二個程式套件。不像平台獨立的記憶體類別50，記憶體程式套件72的類別是平台相關的。這些類別主要給平台相關用戶端，例如，匯流排管理器的程式碼使用。這些類別依

五、發明說明 (53)

javax.system.memory的平台獨立類別而定並將其擴大。

PhysicalAddressSpace(函數名稱)

public final class PhysicalAddressSpace extends AddressSpace

PhysicalAddressSpace並不真的被記憶體類別本身使用，因為一般假定一機器只有一個實體位址空間，但為了完整性、及為了在記憶體類別之外由各種匯流排管理器以MemoryDescriptor使用，而提供它。

除建構子之外，定義下列的公用字串：

public final static String name = "physical";

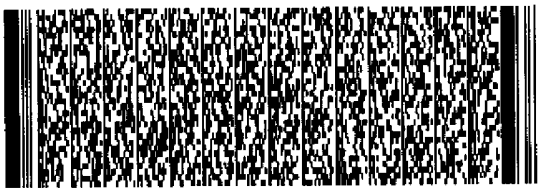
PhysicalAddressSpace(函數名稱)

public PhysicalAddressSpace (ExpansionBus b)

此建構子首先呼叫超類別AddressSpace的建構子，描述ExpansionBus參數和上面定義的字串，以便它們可被公用方法，AddressSpace的getExpansionBus()和getName()使用。然後它呼叫getAddressSize()和alloc_id()原生方法(見以下)且提供它們給AddressSpace的一個保護的初始化常式，以便之後它們可被AddressSpace的公用方法getLowestAddress()、getHighestAddress()和getID()使用。

PhysicalAddressSpace定義下列，實施在微核心中的，原生方法，它分別地提供位址的大小(以位元計算)和該位址空間的識別。

private native int getAddressSize();



五、發明說明 (54)

```
private native int alloc_id();
```

PhysicalMemory(函數名稱)

```
Public class PhysicalMemory extends MainMemory
```

PhysicalMemory類別允許匯流排管理器取得對特定實體位址上之物件的存取(例如被列在爪哇系統資料庫(JSD)裡特定實體位址上之裝置註記)，然後將它們對映到虛擬記憶體(見VirtualMemory建構子方法)之內，以使它們能被裝置驅動程式(經由AccessibleMemory存取方法)存取。對匯流排管理器來說，將DMA設定到所敘述PhysicalMemory物件的陣列(見DMAMemory建構子方法)也是可能的。事實上，藉由首先鎖定虛擬對實體的對映(見VirtualMemory的lock()方法)，然後呼叫VirtualMemory的相關方法(見getPhysicalMemory()方法)以取得PhysicalMemory物件的陣列，然後將此陣列傳送到相關的DMAMemory建構子，匯流排管理器甚至能將DMA設定在對映到虛擬記憶體一個範圍內的記憶體。反之也是可能的，其中DMAMemory物件首先建構，然後取得潛在的PhysicalMemory物件以建構VirtualMemory物件(可經由AccessibleMemory存取)，它對映到相同的PhysicalMemory物件。

雖然PhysicalMemory結合VirtualMemory(它是AccessibleMemory)或結合DMAMemory是非常有用的，光它本身是無用的。這甚至在沒有MMU的系統上也是真的，其中中央處理單元直接地存取實體記憶體。甚至在該情況中，PhysicalMemory物件不是所存取的物件，但當

五、發明說明 (55)

PhysicalMemory物件對映到虛擬記憶體時，VirtualMemory物件可被存取，且它的虛擬位址正好與潛在的實體位址相同。這樣，語意上，驅動程式或任何其他爪哇用戶端總是存取VirtualMemory物件，而永遠不存取PhysicalMemory物件。

定義了下列建構子和公用方法：

PhysicalMemory(函數名稱)

public PhysicalMemory (Address basePA, Address len) throws InvalidAddressException

參數：

basePA-要配置的實體記憶體之基底實體位址。

len-要配置的實體記憶體以位元組計算的長度。

丟出：

如果呼叫超類別(MainMemory)的建構子丟出InvalidAddressException，則丟出InvalidAddressException。

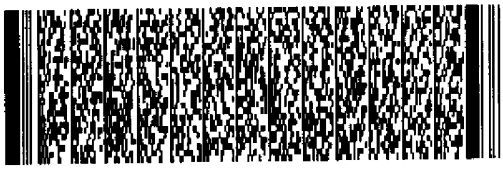
private PhysicalMemory(PhysicalMemory superrange, Address offset, Address len) throws InvalidAddressException, AllocationException

superRange-超範圍PhysicalMemory 物件。

offset-在目前的PhysicalMemory物件裡面對新的子範圍的偏移量。

len-新的子範圍之長度。

丟出:



五、發明說明 (56)

如果offset或offset+newLength超過子範圍被取得之PhysicalMemory物件的範圍，丟出InvalidAddressException。

如果超範圍Memory物件不再是有效的，丟出AllocationException。

第一建構子是公用的，且能給匯流排管理器用來配置PhysicalMemory物件。此配置不實際涉及微核心對應於真正實體記憶體上所做的任何事。然而，PhysicalMemory物件僅是實體記憶體的建構子，及用來建構VirtualMemory(實際上VirtualMemory的可例示的子類別)或DMAMemory物件。就是在此種VirtualMemory或DMAMemory物件建構時，微核心查證PhysicalMemory物件。

此第一建構子不將MemoryConstraints物件當成參數。基本上理由如下。在MemoryConstraints物件中描述了二種型態的限制，配置限制和存取限制。配置限制不適用，因為無任何事物被實際配置，也因為呼叫者描述了基底位址。存取限制也不適用，因為PhysicalMemory物件是不能存取的。

第二的建構子是私用的，且被用在getSubRange()方法內部。它僅將超類別(MainMemory)建構子帶給子範圍。

toString(函數名稱)

public String toString()

傳回:

五、發明說明 (57)

代表此PhysicalMemory物件的字串。

getSubRange(函數名稱)

public PhysicalMemory getSubRange(Address offset, Address newLength) throws InvalidAddressException

參數：

offset-在目前PhysicalMemory物件中對新的子範圍之偏移量。

newLength-新的子範圍之長度。

傳回：

所描述子範圍的PhysicalMemory物件。

丟出：

如果offset或offset+newLength，超出取得子範圍之PhysicalMemory物件的範圍，丟出InvalidAddressException。

如果超範圍Memory物件不再是有效的，丟出AllocationException。

這個方法僅帶來此PhysicalMemory物件的私用建構子。

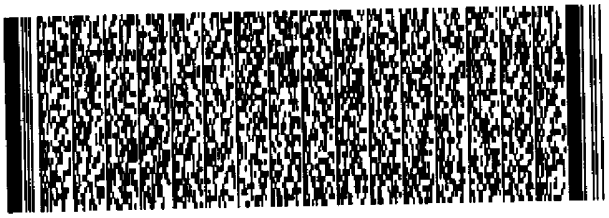
getCacheMode(函數名稱)

public int getCacheMode()

傳回:

快取模式，MainMemory.CACHE_MODE_INHIBITED、MainMemory.CACHE_MODE_WRITE_THROUGH或MainMemory.CACHE_MODE_COPY_BACK中之一。

注意此方法允許匯流排管理器決定，如果



五、發明說明 (58)

PhysicalMemory物件才剛建構、或CACHE_MODE_DEFAULT已經由setCacheMode()所描述，要如何快取實體記憶體。

setCacheMode(函數名稱)

public void setCacheMode(int mode)

參數：

mode-設定成：MainMemory.CACHE_MODE_DEFAULT、MainMemory.CACHE_MODE_INHIBITED、MainMemory.CACHE_MODE_WRITE_THROUGH或MainMemory.CACHE_MODE_COPY_BACK中之一。

這個方法允許匯流排管理器，它應該知道有關所支援的特定平台上各種元件之快取連貫性，控制實體記憶體的快取。例如，在實體記憶體快取與DMA存取不連貫的平台上，在設定DMA傳送時，匯流排管理器有必要暫時將快取模式設定為CACHE_MODE_IN1BIEED。

明確地說，此方法因特定的模式值而定，要求下列事項：

CACHE_MODE_DEFAULT-根據此平台上對此實體記憶體型態的內定值，快取此範圍的實體記憶體。注意，甚至在有實體位址快取的平台上，一些實體記憶體(例入輸入／輸出暫存器)可能需要是不快取的以正確地使用，所以此種實體記憶體將會做成不快取的。另外，注意如果實體記憶體的這個範圍包含需要不同地處理的不同型態之實體記憶體，這個模式將造成不同子範圍不同地被快取，以使實體記憶體的整個範圍能正確地快取。只要可能，這個模式將

五、發明說明 (59)

進行快取以產生可能的最佳效率。

CACHE_MODE_INHIBITED-如果有快取，且它可以是不快取的，它就不快取。如果這不可能，但在子範圍上是可能的，它在任何此種子範圍上這樣做。否則，不做任何事。

CACHE_MODE_WRITE_THROUGH-如果有快取，且若它能轉換成write-through模式，它就這樣轉換。如果這不可能，但在子範圍上是可能的，它在任何此種子範圍上這樣做。否則，不做任何事。

CACHE_MODE_COPY_BACK-如果有快取，且若它能轉換成copy-back模式，它就這樣轉換。如果這不可能，但在子範圍上是可能的，它在任何此種子範圍上這樣做。否則，不做任何事。

flushCache(函數名稱)

```
public void flushCache()
```

如果實體記憶體有快取，只單純地清除快取。如果它是不快取的，這個方法不做任何事。注意對這個物件的位址的整個範圍之實體快取被清除。

PhysicalMemory類別對應MainMemory中所定義的抽象快取管理方法定義下列原生方法。這些原生方法的每一個由微核心中的功能實施。更多細節，參考微核心介面章節。

```
private native void setcachemode (long base, long
len, int mode);
private native int getcachemode(long base, long
len);
```

五、發明說明 (60)

```
private native void flushcache(long base, long
len);
```

VirtualAddressSpace(函數名稱)

```
public final class virtualAddressSpace extends
AddressSpace
```

每個虛擬機器有它自己的VirtualAddressSpace。

除建構子之外，定義下列的公用字串：

```
public final static String name = "virtual";
```

VirtualAddressSpace(函數名稱)

```
public VirtualAddressSpace (ExpansionBus b)
```

此建構子首先呼叫AddressSpace超類別的建構子，描述在上面定義的ExpansionBus參數和字串，以便它們可被AddressSpace的公用方法getExpansionBus()和getName()使用。然後，它呼叫getAddressSize()和alloc_Id()原生方法(見以下)，並將它們提供給AddressSpace的保護的初始化常式，以便之後它們可被AddressSpace的公用方法，getLowestAddress()、getHighestAddress()和getID()使用。

VirtualAddressSpace定義下列原生方法，實施在微核心中，它分別地提供位址(以位元計算)的大小和對該位址空間的識別。識別是用在建構VirtualMemory物件時，以使對映之特定虛擬位址空間可對微核心描述。

```
private native int getAddressSize();
private native int alloc_id();
```

五、發明說明 (61)

VirtualMemory(函數名稱)

public abstract class VirtualMemory extends AccessibleMemory

VirtualMemory物件代表能由中央處理單元藉由載入與儲存運作直接地存取的記憶體，所以它遵循VirtualMemory應該是AccessibleMemory的子類別。VirtuaiMemory抽象類別實際上不實施由所摘錄的任何存取方法，如同那些抽象方法最後全部都由VirtualMemory的子類別實施。VirtualMemory類別確實提供管理VirtualMemory物件對映到PhysicalMemory物件的功能，且這些功能對VirtualMemory的所有子類別是共通的。

VirtualMemory物件總是對映到一些實體記憶體(它對映到那一些不能夠確定，除非對映被鎖定)。然而，甚至在沒有MMU的機器上執行的情況中，仍然有VirtualMemory物件，且它們只單純地一對一對映以使虛擬位址總是等於潛在的實體位址。

VirtualMemory類別是抽象的，因為它擴充AccessibleMemory而沒有實施在AccessibleMemory中定義任何抽象方法。雖然VirtualMemory未定義任何更抽象方法。

當建構virtualMemory物件時，注意有時對映所描述的全體記憶體是不可能的，所以應該呼叫記憶體的getMemLength()方法以決定對映的記憶體之真正總數。

定義下列建構子和公用方法：

五、發明說明 (62)

VirtualMemory(函數名稱)

當VirtualMemory類別是抽象類別，且因而不能例示d，以下所有三個保護的建構子被子類別使用。注意有時對映所描述的全體記憶體是不可能的，所以應該呼叫記憶體的getMemLength()方法以決定對映的記憶體之真正總數。

protected VirtualMemory (int vasid, PhysicalMemory [] PMList, MemoryConstraints constraints) throws InvalidAddressException, AllocationException

參數：

vasid-虛擬位址空間識別，如藉呼叫AddressSpace類別的getID()方法所取得，是VirtualAddressSpace的超類別。

PMList-PhysicalMemory物件的陣列，將以它們出現在陣列中的次序對映到虛擬記憶體中。

constraints-配置限制現在使用在虛擬記憶體的配置和對映期間；存取限制儲存在Memory物件中並在稍後存取時使用。

丟出：

如果微核心不能夠對映所描述參數之虛擬記憶體，丟出InvalidAddressException。

如果在嘗試為所描述的對映配置虛擬記憶體時，微核心面臨配置失敗，丟出AllocationException。

對微核心要求進行對映，提供vasid、基底實體位址和長度的陣列、配置限制和此對映必須鎖定的指示。微核心

五、發明說明 (63)

傳回對映的基底虛擬位址和長度，然後此建構子用來初始化這個Memory物件。

protected virtualMemory(int vasid, Address len, MemoryConstraints constraints) throws InvalidAddressException, AllocationException

參數：

vasid-虛擬位址空間識別，如藉呼叫AddressSpace類別的getID()方法所取得，是VirtualAddressSpace的超類別。

len-要配置的虛擬記憶體的長度。

constraints-配置限制現在使用在虛擬記憶體的配置和對映期間；存取限制儲存在Memory物件中並在稍後存取時使用。

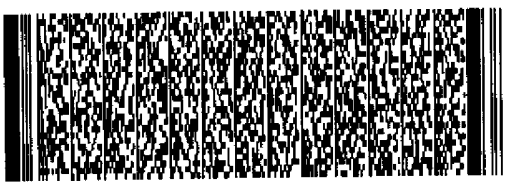
丟出：

如果微核心不能夠對映所描述參數之虛擬記憶體，丟出InvalidAddressException。

如果在嘗試為所描述的對映配置虛擬記憶體時，微核心面臨配置失敗，丟出AllocationException。

在此第二個建構子的情況中，對微核心要求進行對映，提供記憶體的長度和配置限制。微核心傳回對映的基底虛擬位址和長度，然後此建構子用來初始化這個Memory物件。

protected VirtualMemory (VirtualMemory superRange, Address offset, Address len) throws InvalidAddressException, AllocationException



五、發明說明 (64)

參數：

superRange-超範圍VirtualMemory物件。

offset-在目前的VirtualMemory物件中對新的子範圍之偏移量。

len-新的子範圍之長度。

丟出:

如果offset或offset+newLength超過取得子範圍之VirtualMemory物件的範圍，丟出InvalidAddressException。

如果超範圍Memory物件不再是有效的，丟出AllocationException。

此第三個保護建構子被子類別用做它們的getSubRange()方法。此建構子啟動AccessibleMemory超類別對應的保護建構子。另外，初始化私用變數以指示這個子範圍VirtualMemory物件未被鎖定。注意即使超範圍被鎖定，我們也不在最初就指示子範圍被鎖定。這是因為我們允許微核心在一特定範圍的位址上維持一鎖定計數。這表示當我們鎖定這個子範圍時，不管對超範圍的計數已經累積到什麼數字，對它的鎖定計數將會對這個鎖定加一。

子範圍保存在超範圍VirtualMemory物件的私用鍊接表列上，所以unMap()方法如所描述般工作。

unMap(函數名稱)

public void unMap() throws AllocationException

五、發明說明 (65)

丟出:

如果VirtualMemory物件子範圍的鍊接表列是非零的，或如果在微核心的解除對映有一內部失敗，丟出AllocationException。

如果Memory超類別的有效性被設定成零(已經解除對映)，此呼叫立刻返回。這會發生在如果二個執行線在大約相同的時間嘗試呼叫unMap，且其中一個在另一個之前得到。

假使VirtualMemory物件不是子範圍，微核心被呼叫以進行解除對映，詳述基底虛擬位址和長度。如果它只是一個子範圍，它僅是從其超範圍的子範圍Vector中移除。最後，Memory超類別的有效性被設定成偽。

lock(函數名稱)

public long lock() throws AllocationException

傳回:

此VirtualMemory物件的位元組數目，如果成功地鎖定;否則傳回由起頭開始能夠鎖定的位元組數目，讓它成為一子範圍。

丟出:

如果微核心不能配置實體記憶體來支持虛擬記憶體的整個範圍，或如果微核心在使此記憶體對映方面有任何其他的失敗，丟出AllocationException。

如果已經鎖定，此方法立刻返回。否則，要求微核心取得實體記憶體來支持虛擬記憶體的整個範圍(如果還沒

五、發明說明 (66)

做)，並鎖定該對映(或在子範圍的情況中，增加鎖定計數)。微核心將需要保留可能在多重虛擬位址空間中，已經鎖定虛擬記憶體或DMA對實體記憶體的對映，之多個要求(包括鎖定到每個實體頁之對映的鎖定計數)。

lockContiguous(函數名稱)

public void lockContiguous() throws AllocationException

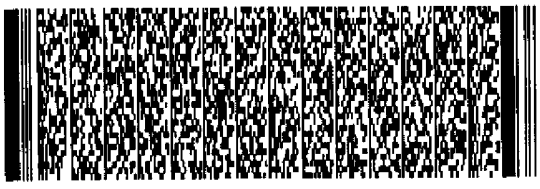
傳回:

此VirtualMemory物件的位元組數目，如果成功地連續地鎖定；否則如果可能鎖定一較小的子範圍(由物件的起頭處開始)，傳回可以被連續地鎖定的位元組總數。

丟出:

如果VirtualMemory已經被鎖定但非連續地，或如果微核心不能配置連續的實體記憶體來支持虛擬記憶體的整個範圍，或如果在微核心使此記憶體對映方面有任何其他的失敗，丟出AllocationException。應該注意的是微核心可能中止運作並傳回失敗，造成此例外被丟出；如果這樣做，將造成任何子範圍的鎖定被解開。

如果已經連續地鎖定，此方法立刻返回。否則，假若VirtualMemory物件不是已經被鎖定，要求微核心取得連續的實體記憶體支持虛擬記憶體的整個範圍，並鎖定該對映。微核心將需要保留可能在多重虛擬位址空間中，已經鎖定虛擬記憶體或DMA對實體記憶體的對映，之多個要求(包括鎖定到每個實體頁之對映的鎖定計數)。



五、發明說明 (67)

當VirtualMemory物件已經非連續地被鎖定時，不允許做連續的鎖定之理由是因為，或許有一用戶端非連續地鎖定它，因此對非連續地鎖定它的軟體最好是做解除鎖定（並處理暫時解除鎖定的任何副作用），勝於引起在此發生的副作用。

unlock(函數名稱)

public void unlock() throws AllocationException

丟出:

如果在微核心解除對映的鎖定中有一些失敗，丟出AllocationException。

如果VirtualMemory物件已經解除鎖定，此方法立刻返回。注意這會使微核心減小其對這整個範圍的鎖定計數；它不影響任何可能被鎖定的子範圍。

getPhysicalMemory(函數名稱)

public PhysicalMemory [] getPhysicalMemory ()

throws AllocationException

傳回:

這個VirtualMemory物件所對映PhysicalMemory物件的陣列(陣列階數用來增加虛擬位址)，如果這個VirtualMemory物件被鎖定(經由lock()或lockContiguons())；否則，如果這個VirtualMemory物件不被鎖定，這個方法傳回零。

丟出:

如果在微核心取得潛在實體記憶體有關資訊(基底位址

五、發明說明 (68)

和長度的陣列)方面有任何失敗，丟出AllocationException。

如果VirtualMemory物件不被鎖定，傳回零。否則，如果PhysicalMemory物件已經被取得給此VirtualMemory物件(參考保存在VirtualMemory物件的一個私用欄位)，簡單地傳回這些頁的參考。否則，首先要求微核心取得基底實體位址和長度的陣列。然後，使用從微核心取得的資訊，建構每個PhysicalMemory物件。在PhysicalMemory物件的陣列傳回之前，對它的參考保存在此VirtualMemory物件的一個私用欄位，以便能處理進一步的要求而不涉及微核心(只要VirtualMemory維持鎖定)。

getCacheMode(函數名稱)

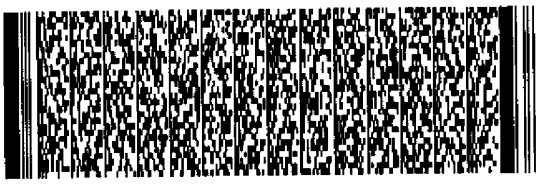
public int getcacheMode()

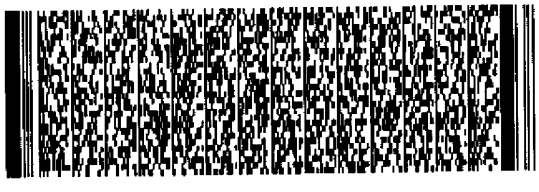
傳回:

快取模式，MainMemory.CACHE_MODE_INHIBITED、MainMemory.CACHE_MODE_AMBIGUOUS、MainMemory.CACHE_MODE_WRITE_THROUGH或MainMemory.CACHE_MODE_COPY_BACK中的一個。

注意這個方法允許匯流排管理器決定虛擬記憶體要如何快取，如果匯流排管理器沒有記錄虛擬記憶體被描述要如何快取，或如果CACHE_MODE_DEFAULT已經由setCacheMode()描述，且現在匯流排管理器要知道究竟內定它是如何快取。

setCacheMode(函數名稱)





五、發明說明 (69)

public void setcacheMode(int mode)

參數:

mode-設定成(見MainMemory):CACHE_MODE_DEFAULT、CACHE_MODE_INHIBITED、CACHE_MODE_WRITE_THROUGH或CACHE_MODE_COPY_BACK中之一。

這個方法允許匯流排管理器,它應該知道有關所支援的特定平台上各種元件之快取連貫性,控制虛擬記憶體的快取。例如,在虛擬記憶體快取與DMA存取不連貫的平台上,在設定DMA傳送時,匯流排管理器有必要暫時將快取模式設定為CACHE_MODE_INHIBITED(或首先呼叫flushCache()並設法保證中央處理單元在DMA進行時不會存取虛擬記憶體)。

明確地說,此方法因特定的模式值而定,要求下列事項:

CACHE_MODE_DEFAULT-根據對此虛擬記憶體型態的內定值、考慮潛在實體記憶體的快取需求,快取此範圍的虛擬記憶體。另外,注意如果虛擬記憶體的這個範圍含有需要不同地快取的不同子範圍,這會被處理,以便之後虛擬記憶體的整個範圍能正確地快取。只要可能,這個模式將進行快取以產生可能的最佳效率。

CACHE_MODE_INHIBITED-如果有快取,且它可以是不快取的,它就不快取。如果這不可能,但在子範圍上是可能的,它在任何此種子範圍上這樣做。否則,不做任何事。

CACHE_MODE_WRITE_THROUGH-如果有快取,且若它能轉換

五、發明說明 (70)

成write-through模式，它就這樣轉換。如果這不可能，但在子範圍上是可能的，它在任何此種子範圍上這樣做。否則，不做任何事。

CACHE_MODE_COPY_BACK-如果有快取，且若它能轉換成copy-back模式，它就這樣轉換。如果這不可能，但在子範圍上是可能的，它在任何此種子範圍上這樣做。否則，不做任何事。

flushCache(函數名稱)

public void flushCache()

如果虛擬記憶體有快取，只單純地清除快取。如果它是不快取的，這個方法不做任何事。注意對這個物件的位址的整個範圍之快取被清除。

VirtualMemory類別定義下列原生方法。這些原生方法的每一個由微核心中的功能實施。更多細節，參考微核心介面段落。

```
private native int physmap(long[] ret, int vasid,
long[] paddrs,long[] sizes, long
minaddr, long maxaddr, int align, int cacheMode,
boolean locked, boolean waitformem);
private native int map(long[] ret, int vasid, long
size, long minaddr,long maxaddr, int align, int
cacheMode, boolean locked,boolean waitformem);
private native int unmap(int vasid, long base,
long len,int howcreated); private native int
```

五、發明說明 (71)

```
dolock(long[] ret, int vasid, long base, long
len);
private native int dolockcontig(long[] ret, int
vasid, long base, long len); private native int
dounlock(int vasid, long base, long len);
private native int getphys(int[] ret, int vasid,
long[] pBase, long[] pLen, long base,long len);
private native void setcachemode(int vasid, long
base, long len, int mode);
private native int getcachemode(int vasid, long
base, long len);
private native void flushcache(int vasid, long
base, long len);
```

DMAAddressSpace(函數名稱)

```
public final class DMAAddressSpace extends
AddressSpace
```

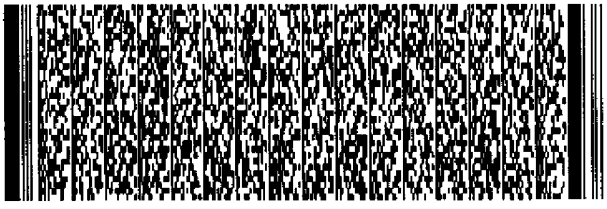
DMAAddressSpace並不真的被記憶體類別本身使用，因為一般假定一機器只有一個DMA位址空間，但是為了完整性而提供它，及給記憶體類別外部由各種匯流排管理器，以MemoryDescriptor使用。

除建構子之外，定義下列的公用字串：

```
public final static String name = "dma";
```

DMAAddressSpace(函數名稱)

```
public DMAAddressSpace (ExpansionBus b)
```



五、發明說明 (72)

此建構子首先呼叫AddressSpace超類別的建構子，描述在上面定義的ExpansionBus參數和字串，以便它們可被AddressSpace的公用方法getExpansionBus()和getName()使用。然後，它呼叫getAddressSize()和alloc_id()原生方法(見以下)，並將它們提供給AddressSpace的保護的初始化常式，以便之後它們可被AddressSpace的公用方法，getLowestAddress()、getHighestAddress()和getID()使用。

DMAAddressSpace定義下列原生方法，實施在微核心中，它分別地提供位址(以位元計算)的大小和對該位址空間的識別。

```
private native int getAddressSize();
private native int alloc_id();
```

DMAMemory(函數名稱)

```
Public class DMAMemory extends Memory
```

DMAMemory物件代表只能經由定址在中央處理單元之外某些DMA通道上的存取的記憶體。這些DMA位址經由DMA硬體對應到實體記憶體位址。因此，DMAMemory物件總是對映到PhysicalMemory物件。由於DMAMemory不能由中央處理單元直接或間接存取，DMAMemory既不是AccessibleMemory的子類別，也非MainMemory。相反的，DMAMemory是Memory的直接子類別。

當建構DMAMemory物件時，注意有時對映所描述的全體記憶體是不可能的，所以應該呼叫Memory的

五、發明說明 (73)

getMemLength() 方法以決定對映的記憶體之真正總數。

定義下列建構子和公用方法：

DMAMemory(函數名稱)

public DMAMemory (PhysicalMemory [] PMList, MemoryConstraints constraints) throws InvalidAddressException, AllocationException, NotSupportedException

參數：

PMList-PhysicalMemory物件的陣列，將以它們出現在陣列中的次序對映到DMAMemory中。

constraints-配置限制現在使用在虛擬記憶體的配置和對映期間；存取限制被忽略，因為中央處理單元不可能存取DMAMemory。

丟出：

如果對微核心的呼叫指示在此平台上不支援DMA，丟出NotSupportedException。

如果呼叫Memory超類別的建構子丟出InvalidAddressException、如果所描述的長度不符合配置限制、如果所描述的PhysicalMemory物件結合不能符合配置限制，或如果任何PhysicalMemory物件是無效的，丟出InvalidAddressException。

如果所描述的PhysicalMemory物件沒有對齊以便能對映到DMAMemory物件的連續範圍、或如果有微核心任何其他內部錯誤阻止它作所描述的對映，丟出

五、發明說明 (74)

AllocationException。

對微核心要求配置DMA對映，提供基底實體位址和長度的陣列、配置限制。微核心傳回對映的基底DMA位址和對映的DMA之長度，然後用此資訊來初始化Memory超類別。

public DMAMemory(Address len, MemoryConstraints constraints) throws InvalidAddressException, AllocationException, NotSupportedException

參數：

len-要配置的DMAMemory的長度。

constraints-只使用配置限制；存取限制忽略。

丟出:

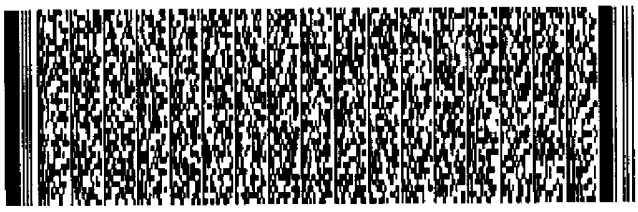
如果對微核心的呼叫指示在此平台上不支援DMA，丟出NotSupportedException。

如果呼叫Memory超類別的建構子丟出InvalidAddressException、或如果所描述的長度不符合配置限制，丟出InvalidAddressException。

如果有微核心任何其他內部錯誤阻止它作所描述的對映，丟出AllocationException。

對微核心要求配置DMA對映，提供要配置的長度和配置限制。微核心傳回基底DMA位址和對映的DMA之長度，然後用此資訊來初始化Memory超類別。

private DMAMemory(DMAMemory superRange, Address offset, Address len) throws InvalidAddressException, AllocationException



五、發明說明 (75)

參數:

superRange-超範圍DMA物件。

offset-在目前的DMA物件裡面對新的子範圍之偏移量。

len-新的子範圍的長度。

丟出:

如果offset或是offset+newLength超過取得子範圍DMAMemory物件之範圍，或如果這塊子範圍不符合原來DMAMemory物件的配置限制，丟出InvalidAddressException。

如果超範圍Memory物件不再有效，丟出AllocationException。

此私用建構子被getSubRange()方法使用。此建構子啟動Memory超類別的保護的建構子。

toString(函數名稱)

public String toString()

傳回:

代表這DMAMemory物件的字串。

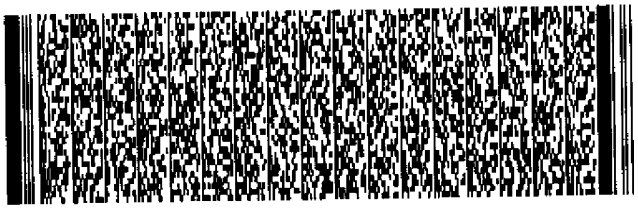
unMap(函數名稱)

public void unMap() throws AllocationException

丟出:

如果DMAMemory物件的子範圍之鍊接表列不是空的，或如果有微核心中所進行解除對映的內部失敗，丟出AllocationException。

如果Memory超類別的有效性被設定成零(已經被解除對



五、發明說明 (76)

映），此呼叫立刻返回。如果二個執行線在大約相同的時間嘗試呼叫unMap()方法，且其中一個在另一個之前取得，這可能發生。所提供DMAMemory物件不是一子範圍，微核心被呼叫進行DMA解除配置，描述基底DMA位址和長度。如果它只是一子範圍，它僅從其超範圍的子範圍之向量中移除。最後，Memory超類別的有效性被設定成偽。

getPhysicalMemory(函數名稱)

public PhysicalMemory [] getPhysicalMemory ()
throws AllocationException

傳回:

這個DMAMemory所對映PhysicalMemory物件的陣列(陣列階數用來增加DMA位址)。

丟出:

如果在微核心取得潛在實體記憶體有關資訊(基底位址和長度的陣列)方面有任何失敗，丟出AllocationException。

如果PhysicalMemory物件已經被取得給此DMAMemory物件(參考保存在DMAMemory物件的一個私用欄位)，簡單地傳回這些頁的參考。否則，首先要求微核心取得基底實體位址和長度的陣列。然後，使用從微核心取得的資訊，建構每個PhysicalMemory物件。在PhysicalMemory物件的陣列傳回之前，對它的參考保存在此DMAMemory物件的一個私用欄位，以便能處理進一步的要求而不涉及微核心。

getSubRange(函數名稱)

五、發明說明 (77)

public Memory getSubRange(Address offset, Address newLength) throws InvalidAddressException

參數:

offset-在目前的DMAMemory物件裡面對新的子範圍之偏移量。

newLength-新的子範圍的長度。

傳回:

所描述子範圍DMAMemory物件。

丟出:

如果offset或是offset+newLength超過取得子範圍VirtualMemory物件之範圍，丟出InvalidAddressException。

如果超範圍Memory物件不再有效，丟出AllocationException。

這個方法僅呼叫用來建構子範圍的私用DMAMemory建構子。

toString(函數名稱)

public String toString()

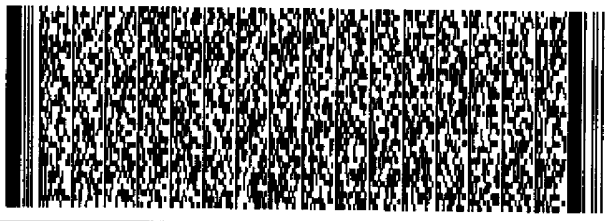
傳回:

代表這DMAMemory物件的字串。

DMAMemory類別定義下列的原生方法。這些原生方法的每一個由微核心中的功能實施。

private native boolean supported();

private native int physmap(long[] ret, long[]



五、發明說明 (78)

```
paddrs, long[] sizes,long minaddr, long maxaddr,
int align, boolean waitformem);
private native int map(long[] ret, long size, long
minaddr,long maxaddr, int align,
boolean waitformem);
private native int unmap(long base, long len);
private native int getphys(int[] ret, long[]
pBase, long[] pLen,long base, long len);
```

VirtualIOMemory(函數名稱)

```
public abstract class VirtualIOMemory extends
VirtualMemory
```

定義下列建構子和公用方法：

VirtualIOMemory(函數名稱)

```
protected VirtualIOMemory (int vasid,
PhysicalMemory[ ] PMList,
Memoryconstraints constraints) throws
InvalidAddressException, AllocationException
protected VirtualIOMemory(int vasid, Address len,
MemoryConstraints constraints) throws
InvalidAddressException, AllocationException
protected VirtualIOMemory (VirtualIOMemory
superrange, Address offset, Address len) throws
InvalidAddressException, AllocationException
```

這三個保護建構子的每一個只被子類別使用，且僅啟動

五、發明說明 (79)

VirtualMemory對應的建構子。更多的細節參照VirtuaMemory。

VirtualIOMemory抽象類別目前沒有提供附加的功能，而僅是為了萬一在SwappedVirtualIOMemory和UnSwappedVirtualIOMemory之間有共通的方法而定義。目前，此類別只定義三個保護的建構子，它們只啟動VirtualMemory超類別的對應建構子。

SwappedVirtualIOMemory(函數名稱)

public class SwappedVirtualIOMemory extends VirtualIOMemory

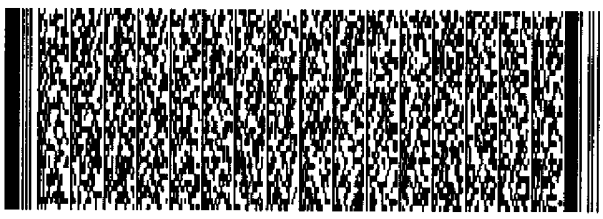
The following constructors and public methods are defined:

定義下列建構子和公用方法：

SwappedVirtualIOMemory(函數名稱)

public SwappedVirtualIOMemory (int vasid, PhysicalMemory[ ] PMList, MemoryConstraints constraints) throws InvalidAddressException, AllocationException, NotSupportedException

public SwappedVirtualIOMemory (int vasid, Address len,MemoryConstraints constraints) throws InvalidAddressException, AllocationException, NotSupportedException private SwappedvirtualIOMemory (SwappedVirtualIOMemory



五、發明說明 (80)

superrange, Address offset, Address len) throws InvalidAddressException, AllocationException

只要在這個平台上支援SwappedVirtualIOMemory，這三個建構子的每一個只啟動VirtualIOMemory的對應建構子，它們依次啟動VirtualMemory的對應建構子。有關更多細節參照VirtualMemory。如果在這個平台上不支援SwappedVirtualIOMemory，丟出NotSupportedException。

getSubRange(函數名稱)

public SwappedVirtualIOMemory getsubRange(Address offset, Address newLength) throws InvalidAddressException, AllocationException

參數：

offset-在目前的SwappedVirtualIOMemory物件裡面對新的子範圍之偏移量。

newLength-新的子範圍的長度。

傳回：

所描述子範圍SwappedVirtualIOMemory物件。

丟出:

如果offset或是offset+newLength超過取得子範圍SwappedVirtualIOMemory物件之範圍，或所交換的子範圍不符合原來SwappedVirtualIOMemory物件之配置限制，丟出InvalidAddressException。

如果超範圍Memory物件不再有效，丟出

五、發明說明 (81)

AllocationException。

這個方法僅啟動SwappedVirtualIOMemory的第三建構子方法，它最終啟動VirtualMemory的第三建構子方法來建構子範圍。

toString(函數名稱)

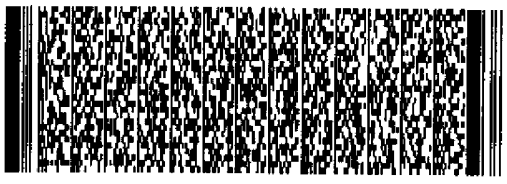
public String toString()

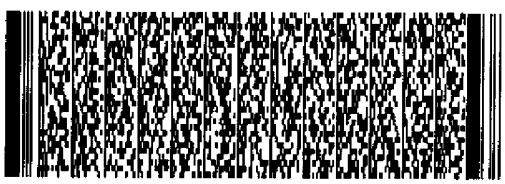
傳回:

代表此SwappedVirtualIOMemory物件的字串。

SwappedVirtualIOMemory類別定義下列原生方法對應於AccessibleMemory中所定義的抽象存取方法。這些原生方法的每一個由微核心中的功能實施，它們必須處理記憶體是位元組交換的、與它是輸入／輸出記憶體，依平台而定可能需要額外操作以處理輸入／輸出記憶體可能不與其他系統記憶體連貫兩個事實。

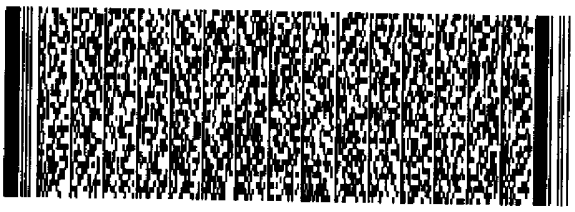
```
protected native boolean supported();
protected native int ncksum(long adr, long len);
protected native void nSetByte(long adr, byte x);
protected native void nSetShort(long adr, short
x);
protected native void nSetInt(long adr, int x);
protected native void nSetLong(long adr, long x);
protected native void nSetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nSetIntArray(long adr, byte
```





五、發明說明 (82)

```
bytes[ ], int bytesoffset, int length_int);
protected native byte nGetByte(long adr);
protected native short nGetshort(long adr);
protected native int nGetLnt(long adr);
protected native long nGetLong(long adr);
protected native void nGetBytes(long adr, byte
bytes[], int bytesoffset, int length);
protected native void nGetintArray(long adr, byte
bytes[], int bytesOffset, int length_int);
protected native void nRegisterSource(long adrl,
long len);
protected native void nCopy(long adr2, long len,
int copyID);
UnSwappedVirtualIOMemory(函數名稱)
public class UnswappedVirtualIoOMemory extends
VirtualIOMemory
UnswappedVirtualIOMemory(函數名稱)
public UnswappedvirtualIOMemory (int vasid,
PhysicalMemory [] PMList,
Memoryconstraints constraints) throws
InvalidAddressException,
AllocationException, NotSupportedException
public UnswappedvirtualIOMemory (int vasid,
Address len,MemoryConstraints constraints)
```



五、發明說明 (83)

throws InvalidAddressException,
AllocationException, NotSupportedException
private UnswappedvirtualIOMemory
(UnswappedVirtualIOMemory superrange,
Address offset, Address len) throws
InvalidAddressException, AllocationException

只要在這個平台上支援UnSwappedVirtualIOMemory，這三個建構子的每一個只啟動VirtualIOMemory的對應建構子，它們依次啟動VirtualMemory的對應建構子。有關更多細節參照VirtualMemory。如果在這個平台上不支援UnSwappedVirtualIOMemory，丟出NotSupportedException。

getSubRange(函數名稱)

public UnswappedvirtualIOMemory getSubRange
(Address offset, Address newLength) throws
InvalidAddressException, AllocationException

參數:

offset-在目前的UnswappedVirtualIOMemory物件裡面對新的子範圍之偏移量。

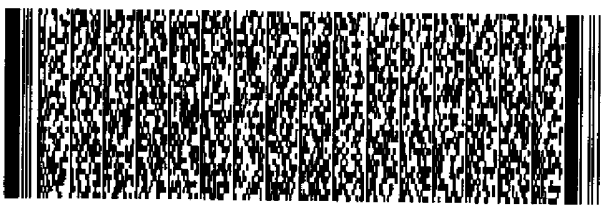
newLength-新的子範圍的長度。

傳回:

所描述子範圍UnSwappedVirtualIOMemory物件。

丟出:

如果offset或是offset+newLength超過取得子範圍



五、發明說明 (84)

UnSwappedVirtualIOMemory物件之範圍，或所提出的子範圍不符合原來UnSwappedVirtualIOMemory物件之配置限制，丟出InvalidAddressException。

如果超範圍Memory物件不再有效，丟出AllocationException。

這個方法僅啟動UnSwappedVirtualIOMemory的第三建構子方法，它最終啟動VirtualMemory的第三建構子方法來建構子範圍。

toString(函數名稱)

public String toString()

傳回:

代表這UnswappedVirtualIOMemory物件的字串。

UnSwappedVirtualIOMemory類別定義下列的原生方法，對應在AccessibleMemory中定義的抽象存取方法。這些原生方法的每一個由微核心中的功能實施，它們必須處理它是輸入／輸出記憶體的事實，可能因平台而定，需要額外的操作來處理輸入／輸出記憶體可能不與其他系統記憶體連貫的事實。不像UnSwappedVirtualIOMemory對應的原生方法，在這裡的原生方法不必處理位元組交換。

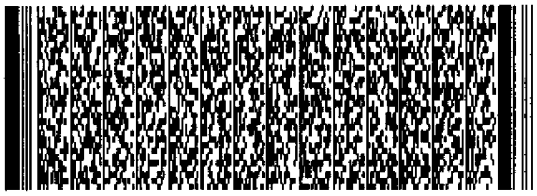
```
protected native boolean supported();
protected native int ncksum(long adr, long len);
protected native void nSetByte(long adr, byte x);
protected native void nSetShort(long adr, short
x);
```

五、發明說明 (85)

```
protected native void nSetInt(long adr, int x);
protected native void nSetLong(long adr, long x);
protected native void nSetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nSetIntArray(long adr, byte
bytes[], int bytesOffset, int length_int);
protected native byte nGetByte (long adr);
protected native short nGetShort(long adr);
protected native int nGetInt(long adr);
protected native long nGetLong(long adr);
protected native void nGetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nGetIntArray(long adr, byte
bytes[], int bytesoffset, int length_int);
protected native void nRegisterSource(long adrl,
long len);
protected native void nCopy(long adr2, long len,
int copyID);
VirtualRegularMemory
public abstract class VirtualRegularMemory extends
VirtualMemory
```

定義下列建構子和公用方法：

```
VirtualRegularMemory(函數名稱)
protected VirtualRegularMemory (int vasid,
```



五、發明說明 (86)

PhysicalMemory [] PMList,
MemoryConstraints constraints) throws
InvalidAddressException, AllocationException
protected VirtualRegularMemory(int vasid, Address
len, MemoryConstraints constraints) throws
InvalidAddressException, AllocationException
protected VirtualRegularMemory
(VirtualRegularMemory superrange, Address
offset, Address len) throws
InvalidAddressException, AllocationException

這三個保護建構子的每一個只被子類別使用，且只啟動對應的VirtualMemory建構子。更多細節參照VirtualMemory。

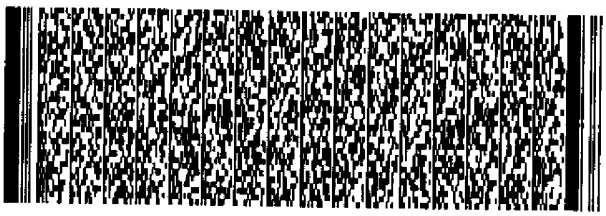
VirtualRegularMemory抽象類別目前沒有提供附加的功能，而只是定義以備萬一在SwappedVirtualRegularMemory和UnswappedVirtualRegularMemory之間有共通的方法。目前，這個類別只定義三個保護的建構子，它們只啟動VirtualMemory超類別的對應建構子。

swappedvirtualRegularMemory(函數名稱)

public class SwappedVirtualRegularMemory extends
VirtualRegularMemory

定義下列建構子和公用方法：

SwappedVirtualRegularMemory(函數名稱)



五、發明說明 (87)

public SwappedVirtualRegularMemory (int vasid, PhysicalMemory [] PMList, MemoryConstraint constraints) throws InvalidAddressException, AllocationException, NotSupportedException public SwappedVirtualRegularMemory (int vasid, Address len, MemoryConstraints constraints) throws InvalidAddressException, AllocationException, NotSupportedException private SwappedVirtualRegularMemory (SwappedVirtualRegularMemory superrange, Address offset, Address len) throws InvalidAddressException, AllocationException

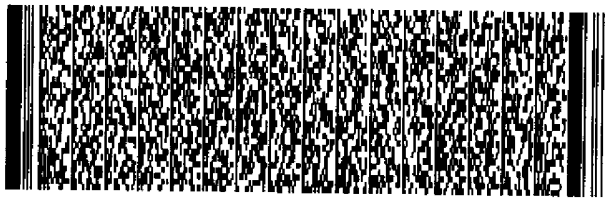
只要這個平台上支援SwappedVirtualRegularMemory，這三個建構子的每一個只啟動VirtualRegularMemory的對應建構子，它們依次啟動VirtualMemory的對應建構子。更多細節參照VirtualMemory。如果這個平台上不支援SwappedVirtualRegularMemory，丟出NotSupportedException。

getSubRange(函數名稱)

public SwappedVirtualRegularMemory getSubRange (Address offset, Address newLength) throws InvalidAddressException, AllocationException

參數：

offset-在目前的SwappedVirtualRegularMemory物件裡面



五、發明說明 (88)

對新的子範圍之偏移量。

newLength-新的子範圍的長度。

傳回:

SwappedVirtualRegularMemory物件，它是所描述的子範圍。

丟出:

如果offset或是offset+newLength超過取得子範圍SwappedVirtualRegularMemory物件之範圍，或所提出的子範圍不符合原來SwappedVirtualRegularMemory物件之配置限制，丟出InvalidAddressException。

如果超範圍Memory物件不再有效，丟出AllocationException。

這個方法僅啟動SwappedVirtualRegularMemory的第三建構子方法，它最終啟動VirtualMemory的第三建構子方法來建構子範圍。

toString(函數名稱)

public String toString()

傳回:

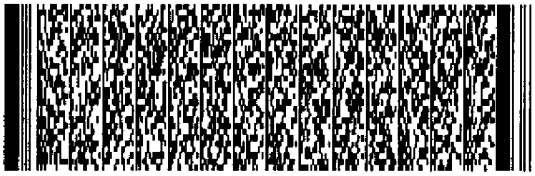
代表這SwappedVirtualRegularMemory物件的字串。

SwappedVirtualRegularMemory 類別定義下列原生方法，對應在AccessibleMemory中定義的抽象存取方法。這些原生方法每一個由微核心中的功能實施，微核心必須處理兩個記憶體都是位元組交換的事實。

protected native boolean supported();

五、發明說明 (89)

```
protected native int ncksum(long adr, long len);
protected native void nSetByte(long adr, byte x);
protected native void nSetShort(long adr, short
x);
protected native void nSetInt(long adr, int x);
protected native void nSetLong(long adr, long x);
protected native void nSetBytes(long adr, byte
bytes [], int bytesOffset, int length);
protected native void nSetIntArray(long adr, byte
bytes[], int bytesOffset, int length_int);
protected native byte nGetByte(long adr);
protected native short nGetShort(long adr);
protected native int nGetInt(long adr);
protected native long nGetLong(long adr);
protected native void nGetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nGetIntArray(long adr, byte
bytes[], int bytesOffset, int length_int);
protected native void nRegisterSource(long adrl,
long len);
protected native void nCopy(long adr2, long len,
int copyID);
UnSwappedVirtualRegularMemory
public class UnSwappedVirtualRegularMemory extends
```



五、發明說明 (90)

VirtualRegularMemory

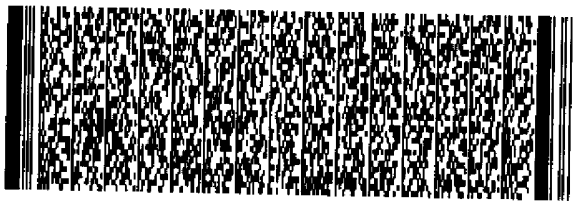
定義下列建構子和公用方法：

UnSwappedVirtualRegularMemory(函數名稱)

public UnSwappedVirtualRegularMemory (int vasid, PhysicalMemory [] PMList, MemoryConstraints constraints) throws InvalidAddressException, AllocationException, NotSupportedException

public UnSwappedVirtualRegularMemory (int vasid, Address len,MemoryConstraints constraints) throws InvalidAddressException, AllocationException, NotSupportedException private UnSwappedVirtualRegularMemory (UnSwappedVirtualRegularMemory superrange, Address offset, Address len) throws InvalidAddressException, AllocationException

只要在這個平台上支援UnSwappedVirtualRegularMemory，這三個建構子的每一個只啟動VirtualRegularMemory的對應建構子，它們依次啟動VirtualMemory的對應建構子。有關更多細節參照VirtualMemory。如果在這個平台上不支援UnSwappedVirtualRegularMemory，丟出NotSupportedException。



五、發明說明 (91)

getSubRange(函數名稱)

public UnSwappedVirtualRegularMemory getSubRange (Address offset, Address newLength) throws InvalidAddressException, AllocationException

參數:

offset-在目前的UnSwappedVirtualRegularMemory物件裡面對新的子範圍之偏移量。

newLength-新的子範圍的長度。

傳回:

UnSwappedVirtualRegularMemory物件，它是所描述的子範圍。

丟出:

如果offset或是offset+newLength超過取得子範圍UnSwappedVirtualRegularMemory物件之範圍，或所提出的子範圍不符合原來UnSwappedVirtualRegularMemory物件之配置限制，丟出InvalidAddressException。

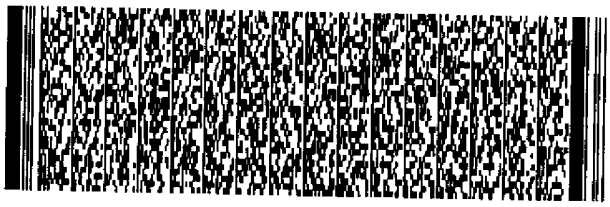
如果超範圍Memory物件不再有效，丟出AllocationException。

這個方法僅啟動UnSwappedVirtualRegularMemory的第三建構子方法，它最終啟動VirtualMemory的第三建構子方法來建構子範圍。

toString(函數名稱)

public String toString()

傳回:

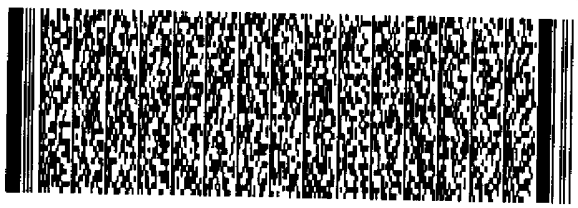


五、發明說明 (92)

代表這UnSwappedVirtualRegularMemory物件的字串。

UnSwappedVirtualRegularMemory類別定義下列原生方法，對應在AccessibleMemory中定義的抽象存取方法。不像SwappedVirtualRegularMemory的對應原生方法，在此的原生方法不用處理位元組交換。

```
protected native boolean supported();
protected native int ncksum(long adr, long len);
protected native void nSetByte(long adr, byte x);
protected native void nSetShort(long adr, short
x);
protected native void nSetInt(long adr, int x);
protected native void nSetLong(long adr, long x);
protected native void nSetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nSetIntArray(long adr, byte
bytes[], int bytesOffset, int length_int);
protected native byte nGetByte(long adr);
protected native short nGetShort(long adr);
protected native int nGetInt(long adr);
protected native long nGetLong(long adr);
protected native void nGetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nGetIntArray(long adr, byte
bytes(), int bytesOffset, int length_int);
```



```
protected native void nRegisterSource(long adrl,
long len);
protected native void nCopy(long adr2, long len,
int copyID);
```

PortIOAddressSpace(函數名稱)

```
public final class PortIOAddressSpace extends
AddressSpace
```

PortIOAddressSpace並不真的被記憶體類別本身使用，因為一般假定一機器只有一個埠輸入／輸出位址空間，但為了完整性而提供它，與為了在記憶體類別外面，以MemoryDescriptor，由各種匯流排管理器使用。

除建構子之外，定義下列的公用字串：

```
public final static String name = "portio";
```

PortIOAddressSpace(函數名稱)

```
public PortIOAddressSpace (ExpansionBus b)
```

此建構子首先呼叫AddressSpace超類別的建構子，描述ExpansionBus參數和在上面定義的字串，以便它們可被AddressSpace的公用方法getExpansionBus()和getName()使用。然後它呼叫getAddressSize()和alloc_id()原生方法(見以下)並將它們提供給AddressSpace的一個保護的初始化常式，以便之後它們可被AddressSpace的公用方法getLowestAddress()、getHighestAddress()和getID()使用。

PortIOAddressSpace定義下列原生方法，實施在微核心

五、發明說明 (94)

中，它分別地提供位址的大小(以位元計算)和位址空間的識別。

```
private native int getAddressSize();
private native int alloc_id();
```

PortIOMemory(函數名稱)

```
public abstract class PortIOMemory extends
AccessibleMemory
```

PortIOMemory抽象類別提供在SwappedPortIOMemory和UnSwappedPortIOMemory之間共通的一些次要功能。

定義下列建構子和公用方法:

PortIOMemory(函數名稱)

```
protected PortIOMemory (Address p, Address len,
MemoryConstraints constraints)
```

參數:

p-BaseAddress

len-長度

constraints-MemoryConstraints

此建構子只呼叫AccessibleMemory超類別保護的建構子。

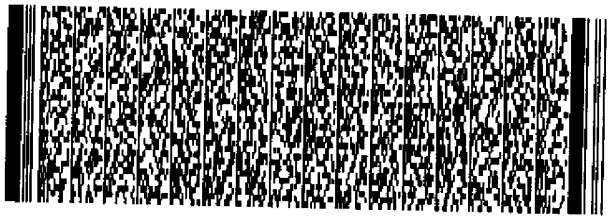
setCacheMode(函數名稱)

```
public void setCacheMode(int mode)
```

參數:

mode-所要求的快取模式。

這個方法什麼也不做；不管描述了什麼快取模式，對



五、發明說明 (95)

PortIOMemory沒有支援快取，所以模式總是保持為CACHE_MODE_INHIBITED。

getCacheMode(函數名稱)

public int getCacheMode()

傳回:

快取模式。

這個方法總是傳回CACHE_MODE_INHIBITED。

flushCache(函數名稱)

public void flushCache()

由於沒有快取要清除，這個方法不做任何事。

SwappedPortIOMemory(函數名稱)

public class SwappedPortIOMemory extends PortIOMemory

定義下列建構子和公用方法:

SwappedPortIOMemory(函數名稱)

public SwappedPortIOMemory(Address p, Address len, MemoryConstraints constraints)

參數:

p-BaseAddress

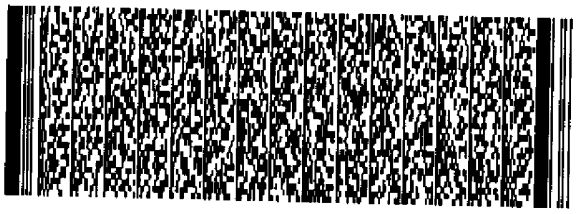
len-長度

constraints-Memory Constraints

此建構子只呼叫PortIOMemory超類別的保護的建構子。

getSubRange(函數名稱)

protected getSubRange(Address offset, Address



五、發明說明 (96)

newLength) throws InvalidAddressException, AllocationException

參數:

offset-在目前的SwappedPortIOMemory物件裡面對新的子範圍之偏移量。

newLength-新的子範圍的長度。

傳回:

Memory物件(實際上精確地SwappedPortIOMemory物件),它是所描述的子範圍。

丟出:

如果offset或是offset+newLength超過取得子範圍SwappedPortIOMemory物件之範圍,或所提出的子範圍不符合原來SwappedPortIOMemory物件之限制(例如對齊),丟出InvalidAddressException。

如果超範圍Memory物件不再有效,丟出AllocationException。

這個方法僅啟動SwappedPortIOMemory的建構子方法。

toString(函數名稱)

public String toString()

傳回:

代表這SwappedPortIOMemory物件的字串。

SwappedPortIOMemory類別定義下列原生方法,對應於AccessibleMemory中定義的抽象存取方法。這些原生方法的每一個由微核心中的功能實施,它必須處理記憶體是位

五、發明說明 (97)

元組交換且它是埠記憶體兩個事實。

```
protected native boolean supported();
protected native int ncksum(long adr, long len);
protected native void nSetByte(long adr, byte x);
protected native void nSetShort(long adr, short
x);
protected native void nSetInt(long adr, int x);
protected native void nSetLong(long adr, long x);
protected native void nSetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nSetIntArray(long adr, byte
bytes[], int bytesOffset, int lengt_int);
protected native byte nGetByte(long adr);
protected native short nGetShort(long adr);
protected native int nGetInt(long adr);
protected native long nGetLong(long adr);
protected native void nGetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nGetIntArray(long adr, byte
bytes[], int bytesOffset, Int length_int);
protected native void nRegisterSource(long adr1,
long len);
protected native void nCopy(long adr2, long len,
int copyID);
```

五、發明說明 (98)

UnSwappedPortIOMemory(函數名稱)

public class UnswappedPortIOMemory extends PortIOMemory

定義下列建構子和公用方法:

UnswappedPortIOMemory(函數名稱)

public UnSwappedPortIOMemory(Address p, Address len, MemoryConstraints constraints)

參數:

p-BaseAddress

len-Length

constraints-MemoryConstraints

此建構子只呼叫PortIOMemory超類別保護的建構子。

getSubRange(函數名稱)

protected getSubRange(Address offset, Address newLength) throws InvalidAddressException

參數:

offset-在目前的UnSwappedPortIOMemory物件裡面對新的子範圍之偏移量。

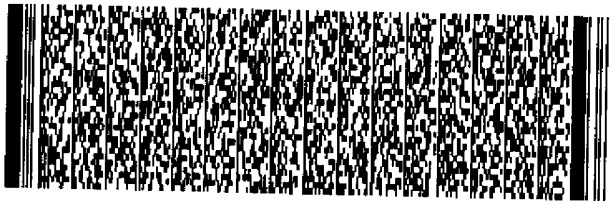
newLength-新的子範圍的長度。

傳回:

Memory物件(實際上精確地UnSwappedPortIOMemory物件),它是所描述的子範圍。

丟出:

如果offset或是offset+newLength超過取得子範圍



五、發明說明 (99)

UnSwappedPortIOMemory物件之範圍，或如果所提出的子範圍不符合原來UnSwappedPortIOMemory物件之限制(例如對齊)，丟出InvalidAddressException。

如果超範圍Memory物件不再有效，丟出AllocationException。

這個方法僅啟動UnSwappedPortIOMemory的建構子方法。

toString(函數名稱)

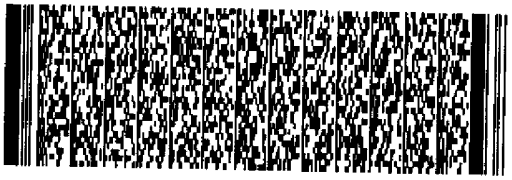
public String toString()

傳回:

代表這UnSwappedPortIOMemory物件的字串。

UnswappedPortIOMemory類別定義下列原生方法，對應於AccessibleMemory中定義的抽象存取方法。這些原生方法的每一個由微核心中的功能實施，它必須處理記憶體是埠記憶體的事實。

```
protected native boolean supported();
protected native int ncksum(long adr, long len);
protected native void nSetByte(long adr, byte x);
protected native void nSetShort(long adr, short
x);
protected native void nSetInt(long adr, int x);
protected native void nSetLong(long adr, long x);
protected native void nSetBytes(long adr, byte
bytes[], int bytesOffset, int length);
```



五、發明說明 (100)

```
protected native void nSetIntArray(long adr, byte
bytes[], int bytesOffset, int length_int);
protected native byte nGetByte(long adr);
protected native short nCetShort(long adr);
protected native int nGetInt(long adr);
protected native long nGetLong(long adr);
protected native void nGetBytes(long adr, byte
bytes[], int bytesOffset, int length);
protected native void nGetIntArray(long adr, byte
bytes[], int bytesOffset, int length_int);
protected native void nRegisterSource(long adr1,
long len);
protected native void nCopy(long adr2, long len,
int copyID);
```

微核心記憶體和DMA方法

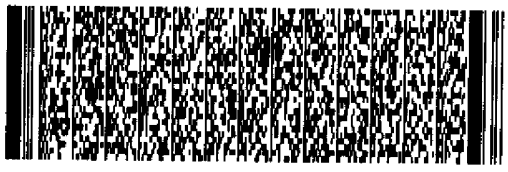
微核心62的DMA和Memory方法80和82，完全地包括各種爪哇類別原生方法的集合。在一起，這些原生方法包含在特定微核心的函數之程式館中，且該程式館的每個函數不是處理原生方法本身，就是對微核心執行系統呼叫以處理原生方法。

PhysicalAddressSpace

PhysicalAddressSpace類別定義下列的原生方法:

getAddressSize(函數名稱)

```
private native int getAddressSize();
```



傳回實體位址的位元數目。

alloc_id(函數名稱)

```
private native int alloc_id();
```

由於一般假定只有一個實際位址空間，微核心只是傳回0。

PhysicalMemory

PhysicalMemory類別定義下列原生方法，對應於在MainMemory中定義的抽象快取管理方法。

下列原生方法控制與特定平台上實體位址有關之資料的快取。

setcachemode(函數名稱)

```
private native void setcachemode(long base, long len, int mode);
```

此原生方法設定對實體記憶體的快取，由實體位址、基底開始，並持續len位元組的長度。快取根據下列模式的可能值(對模式的任何其他值，不作任何事)而設定：

·CACHE_MODE_DEFAULT-根據此平台上對此實體記憶體型態的內定值，快取此範圍的實體記憶體。注意，甚至在有實體位址快取的平台上，一些實體記憶體(例入輸入／輸出暫存器)可能需要是不快取的，且如果它們需要是不快取的以正確地使用，微核心被期待會使它們不快取。另外，注意如果實體記憶體的這個範圍包含需要不同地處理的不同型態之實體記憶體，如果有必要微核心被期待會不同地快取不同的子範圍，以使整個範圍能正確地快取。除

五、發明說明 (102)

這些限制之外，微核心應試圖設定快取以帶來正確的最大效率。換句話說，如果可以使用快取，而仍維持資料完整性，它們就應該如此。

·CACHE_MODE_INHIBITED-如果快取這個實體記憶體是可能的，且使它的快取失去作用是可能的，那麼使它的快取失去作用。否則，不作任何事。如果這對整個範圍是不可能的，但在一或多個子範圍是可能的，至少在任何這樣的子範圍上如此做。

·CACHE_MODE_WRITE_THROUGH-如果快取這個實體記憶體是可能的，且將快取設定為write-through模式是可能，則將它設定如此。否則，不作任何事。如果這對整體範圍是不可能的，但對一或多個子範圍是可能的，至少在任何此種子範圍上如此做。

·CACHE_MODE_COPY_BACK-如果快取這個實體記憶體是可能的，且將快取設定為copy-back模式是可能的，則將它如此設定。否則，不作任何事。如果這對整體範圍是不可能的，但對一或多個子範圍是可能的，至少在任何此種子範圍上如此做。

getcachemode(函數名稱)

```
private native int getcachemode(long base, long
len);
```

此原生方法取得對實體記憶體的快取，由實體位址、基底開始，並連續len個位元組的長度。將傳回下列值中的一個：

五、發明說明 (103)

CACHE_MODE_AMBIGUOUS-如果全體的範圍不是以相同方法快取，或下列的回傳值沒有一個適用。

CACHE_MODE_INHIBITED-如果沒有快取，或如果有快取，但是這個實體記憶體範圍現在是不快取的。

CACHE_MODE_WRITE_THROUGH-如果實體記憶體的這個範圍現在是以write-through模式快取的。

CACHE_MODE_COPY_BACK-如果實體記憶體的這個範圍現在是以copy-back模式快取的。

flushcache(函數名稱)

```
private native void flushcache(long base, long
len);
```

如果實體記憶體這個範圍的任何子範圍是以CACHE_MODE_COPY_BACK模式快取，如果可能就清除任何此種快取。如果不可能清除它們，將它們轉換成CACHE_MODE_INHIBITED。如果連這也不可能，一般假設不知何故該平台設計成DMA總是與任何有關的實體快取連貫，且在那些情況中，這個方法什麼也不做是可以的。

VirtualAddressSpace

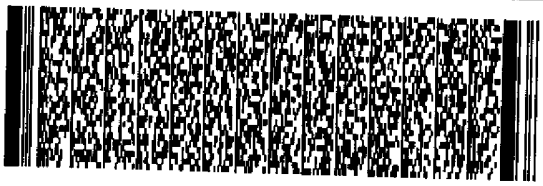
VirtualAddressSpace類別定義下列原生方法：

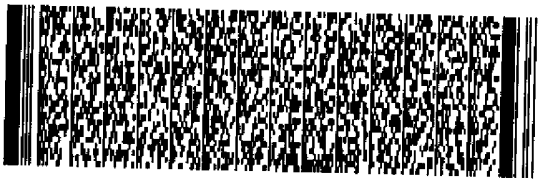
getAddressSize(函數名稱)

```
private native int getAddressSize();
```

傳回對這個虛擬位址空間(線正在其中執行那一個)之虛擬位址的位元數目。

alloc_id(函數名稱)





五、發明說明 (104)

```
private native int alloc_id();
```

每個虛擬機器在分開的虛擬位址空間(在一機器上只有一個虛擬位址空間真正有一虛擬機器)中執行。這個方法允許微核心描述一識別(只有微核心會解譯這個識別)，它可接著用來識別在VirtualMemory類別的原生方法對映的特定虛擬位址空間。特定虛擬位址空間會是線呼叫此原生方法設定要使用的那一個。

VirtualMemory

VirtualMemory類別定義下列原生方法，它們控制虛擬記憶體。如果平台不真的有虛擬記憶體，那麼虛擬記憶體定義成一對一對映(自動地)到實體記憶體，在此種方式中與實體位址相關的虛擬位址與該實體位址相同。

最先三個原生方法對映和解除對映虛擬記憶體。第一個原生方法，physmap()，設定虛擬記憶體對映到實體記憶體範圍的一個陣列，而第二個原生方法，map()，只是對映虛擬記憶體所描述的數量，而不描述它所對映的實體記憶體。第三個原生方法，unmap()，解除另外二個方法所建立的對映。

physmap(函數名稱)

```
private native int physmap(long[] ret, int vasid,
long[] paddrs, long[] sizes, long minaddr, long
maxaddr, int align, int cacheMode, boolean locked,
boolean waitformem);
```

此對映要進行的特定虛擬位址空間m，由vasid描述，它

五、發明說明 (105)

是vasid呼叫者將經由對VirtualAddressSpace類別的alloc_vasid()原生方法的呼叫取得。

要對映的實體記憶體由paddrs和sizes陣列描述，以此種方式paddrs的第n個元件和sizes的第n個元件，分別地描述實體記憶體一個範圍的基底實體位址和長度。實體記憶體的這些範圍將對映到越來越大的虛擬位址，這樣子該實體記憶體範圍n(如paddrs和sizes的第n個元件所描述)將對映到比對映到實體記憶體範圍n+I的虛擬位址更低的虛擬位址，此外，虛擬位址範圍會是連續的，以使它們形成一個虛擬位址範圍。

對於此對映，可以使用任何範圍的虛擬位址，除了在此範圍裡的所有虛擬位址必須比minaddr更大或與其相等，且此範圍裡的所有虛擬位址必須比maxaddr更小或與其相等。另外，基底虛擬位址將如align所描述對齊，它可以有下列值中之一：

MemoryConstraints.ALIGN_BYTE(0)-沒有對齊限制。

MemoryConstraints.ALIGN_SHORT(1)-對齊到短整數。

MemoryConstraints.ALIGN_INT(2)-對齊到整數。

MemoryConstraints.ALIGN_LONG(3)-對齊到長整數。

MemoryConstraints.ALIGN_CACHE(4)-對齊到快取線大小。

MemoryConstraints.ALIGN_PAGE(5)-對齊到頁大小。

虛擬記憶體將像cacheMode在對原生方法setcachemode()的呼叫中對模式的描述一樣快取。

五、發明說明 (106)

如果locked為真，對映到特定實體頁的虛擬記憶體將被鎖定，就像dolock()原生方法被呼叫過。

如果waitformem為真，且它目前，由於暫時地資源短缺，不可能完成對映要求，那麼無限期地等待以完成要求。否則，不等候。

如果不能符合所有這些限制，但它們能在較小的必須由實體記憶體第一個範圍開始的子範圍符合，那麼對較小的子範圍對映。舉例來說，如果這個平台並不真的有虛擬記憶體，且若實體記憶體範圍是非連續的，那麼明顯地只有實體記憶體的第一個範圍能對映。

此範圍的基底虛擬位址將在ret[0]中傳回，而所對映的虛擬記憶體的長度將在ret[1]中傳回。

可能的回傳值是：

・SUCCESS-對映是成功的。不是對映實體記憶體全體，就是至少對映一子範圍。

・ADDRESS_FAILURE-有某些與所描述的實體位址有關的失敗。因某種原因那些位址是未定義實體記憶體，或實體記憶體無法設定對映。

・ALLOC_FAILURE-在嘗試建立對映中有某些其他重大失敗。

map(函數名稱)

```
private native int map(long[] ret, int vasid, long
size, long minaddr,long maxaddr, int align, int
cacheMode, boolean locked, boolean waitformem);
```

五、發明說明 (107)

要進行此對映的特定虛擬位址空間由vasid描述，它是呼叫者將經由對VirtualAddressSpace類別的alloc_vasid()原生方法的呼叫取得。

要配置的虛擬記憶體總數由size以位元組計算描述。對於這對映，可以使用任何範圍的虛擬位址，除了在此範圍裡的所有虛擬位址必須比minaddr更大或與其相等，且此範圍裡的所有虛擬位址必須比maxaddr更小或與其相等。另外，基底虛擬位址將如align所描述對齊，它可以有下列值中之一：

MemoryConstraints.ALIGN_BYTE(0)-沒有對齊限制。

MemoryConstraints.ALIGN_SHORT(1)-對齊到短整數。

MemoryConstraints.ALIGN_INT(2)-對齊到整數。

MemoryConstraints.ALIGN_LONG(3)-對齊到長整數。

MemoryConstraints.ALIGN_CACHE(4)-對齊到快取線大小。

MemoryConstraints.ALIGN_PAGE(5)-對齊到頁大小。

虛擬記憶體將像cacheMode在對原生方法setcachemode()的呼叫中對模式的描述一樣快取。

如果locked為真，對映到特定實體頁的虛擬記憶體將被鎖定，就像dolock()原生方法被呼叫過。

如果waitformem為真，且它目前，由於暫時地資源短缺，不可能完成對映要求，那麼無限期地等待以完成要求。否則，不等候。

如果不能符合所有這些限制，但它們能在較小的必須由

五、發明說明 (108)

實體記憶體第一個範圍開始的子範圍符合，那麼對較小的子範圍對映。

此範圍的基底虛擬位址將在ret[0]中傳回，而所對映的虛擬記憶體的長度將在ret[1]中傳回。

可能的回傳值是：

·SUCCESS-對映是成功的。不是對映實體記憶體全體，就是至少對映一子範圍。

·ADDRESS_FAILURE-有某些與所描述的實體位址有關的失敗。因某種原因那些位址是未定義實體記憶體，或實體記憶體無法設定對映。

·ALLOC_FAILURE-在嘗試建立對映中有某些其他重大失敗。

unmap(函數名稱)

```
private native int unmap(int vasid, long base,
long len,int howcreated);
```

要進行此解除對映的特定虛擬位址空間由vasid描述，它是呼叫者將經由對VirtualAddressSpace類別的alloc_vasid()原生方法的呼叫取得。

要解除對映的虛擬記憶體是由基底開始，且有len個位元組的長度。

howcreated參數描述虛擬記憶體最先如何對映：

CREATED_MAP-經由physmap()建立。

CREATED_MALLOC-經由map()建立。

可能的回傳值是：

五、發明說明 (109)

・SUCCESS-解除對映是成功的。所描述的整個範圍被解除對映。

・ALLOC_FAILURE-在嘗試進行解除對映中有某些其他重大失敗。

下面三個原生方法鎖定與解除鎖定虛擬記憶體對映。dolock()原生方法，將虛擬記憶體鎖定到特定實體記憶體而不管潛在的實體記憶體是連續的，而dolockcontig()原生方法，鎖定連續的實體記憶體。unlock()原生方法，解除另外二個方法鎖定的對映。

dolock(函數名稱)

```
private native int dolock(long[] ret, int vasid,
long base, long len);
```

要進行此鎖定的特定虛擬位址空間由vasid描述，它是呼叫者將經由對VirtualAddressSpace類別的alloc_vasid()原生方法的呼叫取得。

要鎖定的虛擬記憶體是由基底開始，且有len個位元組的長度。

當鎖定虛擬記憶體的一個範圍時，它表示微核心不能夠改變潛在實體記憶體的識別。因此，當鎖定時，特定範圍的虛擬記憶體總是對映到實體記憶體的相同範圍。應該注意的是微核心需要維持一鎖定計數，以使每個鎖定特定範圍的呼叫必須由一樣多個對dounlock()原生方法的呼叫來恢復原狀。

如果鎖定不成功，這個原生方法將不會造成部分範圍任

五、發明說明 (110)

何鎖定被完成。如果鎖定是成功的，ret[0]將會是所鎖定(範圍的全體長度)範圍的位元組總數。如果鎖定不成功，但是有可能鎖定在此範圍的起點處開始的子範圍，則ret[0]是那個子範圍的長度。

可能的回傳值是：

·SUCCESS-鎖定是成功的。所描述的全體範圍被鎖定。

·NOT_ALL_LOCKED-不可能鎖定所描述的整個範圍，但可能鎖定在起點處開始的較小子範圍，其長度是ret[0]。

·ALLOC_FAILURE-在嘗試做解除對映中有某些其他的重大失敗。

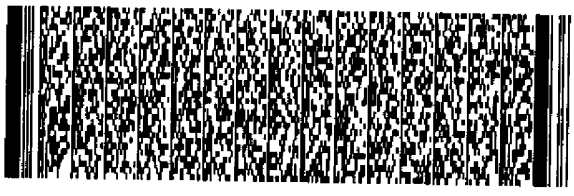
dolockcontig(函數名稱)

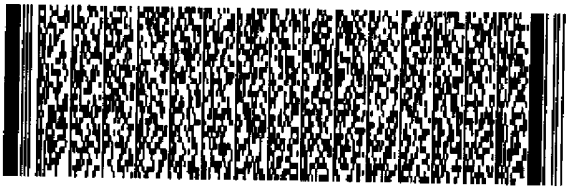
```
private native int dolockcontig(long[] ret, int vasid, long base, long len);
```

要進行此鎖定的特定虛擬位址空間由vasid描述，它是呼叫者將經由對VirtualAddressSpace類別的alloc_vasid()原生方法的呼叫取得。

要鎖定的虛擬記憶體是由基底開始，有len個位元組的長度。

當鎖定一個範圍的虛擬記憶體時，它表示微核心不能夠改變潛在實體記憶體的識別。因此，當鎖定時，特定範圍的虛擬記憶體總是對映到實體記憶體的相同範圍。應該注意的是微核心需要維持一鎖定計數，以使每個鎖定特定範圍的呼叫必須由一樣多個對dounlock()原生方法的呼叫來恢復原狀。





五、發明說明 (111)

與dolock()原生方法相對的，這個特別的原生方法進行鎖定以使虛擬位址的全體範圍對映到連續的實體記憶體。為了要這樣做，微核心可能需要重新對映這個虛擬範圍的部分。但是，要注意，如果此重新對映違反任何經由dolock()或dolockcontig()所做的鎖定，那麼這個dolockcontig()呼叫將失敗(以NOT_ALL_LOCKED)。

如果鎖定不成功，這個原生方法將不會引起任何部分範圍的鎖定。如果鎖定是成功的，ret[0]將會是所鎖定(範圍的全體長度)範圍的位元組總數。如果鎖定不成功，但是有可能鎖定在此範圍的起點處開始的子範圍，則ret[0]是那個子範圍的長度。

可能的回傳值是：

·SUCCESS-鎖定是成功的。所描述的全體範圍被連續地鎖定。

·NOT_ALL_LOCKED-不可能鎖定所描述的整個範圍，但可能連續地鎖定在起點處開始的較小子範圍，其長度是ret[0]。

·ALLOC_FAILURE-在嘗試做解除對映中有某些其他的重大失敗。

dounlock(函數名稱)

```
private native int dounlock(int vasid, long base,
long len);
```

要完成解除鎖定的特定虛擬位址空間由vasid描述，它是呼叫者將經由對VirtualAddressSpace類別的

五、發明說明 (112)

alloc_vasid()原生方法的呼叫取得。

要解除鎖定的虛擬記憶體是由基底開始有len個位元組的長度。此呼叫減少該範圍被dolock()或dolockcontig()原生方法增加的鎖定計數。

那可能的回傳值是：

・SUCCESS-解除對映是成功的。所描述的全體範圍被解除鎖定。

・ALLOC_FAILURE-在嘗試做解除鎖定中有某些其他的重大失敗。

getphys(函數名稱)

```
private native int getphys(int[] ret, int vasid,
long[] pBase, long[] pLen, long base,long len);
```

假設虛擬記憶體首先鎖定，它保証潛在的實體記憶體的識別會保持相同。對getphys()原生方法的呼叫是用來取得潛在的實體記憶體。

要進行此運作的特定虛擬位址空間由vasid描述，它是呼叫者將經由對VirtualAddressSpace類別的alloc_vasid()原生方法的呼叫取得。

虛擬記憶體的範圍之基底由base描述，而範圍的長度由len以位元組計算描述。

如果虛擬記憶體未鎖定，這個呼叫什麼也不做而只是傳回FAILURE。

實體記憶體將在所提供的pBase和pLen陣列描述，如此pBase[0]和pLen[0]是開始對映虛擬記憶體的起點之實體

五、發明說明 (113)

記憶體的基底和長度，且如此(pBase[n+1]，pLen[n+1])描述對映較高範圍虛擬位址的實體記憶體，相較於對映到(pBase[n]，pLen[n])所描述實體記憶體的虛擬位址的範圍。

呼叫者不可能知道實體記憶體多少不連續的範圍對映這個虛擬記憶體範圍，所以有可能對pBase與pLen所提供陣列太小。如果是這種情形，微核心將傳回ARRAY_TOO_SMALL並在ret[0]中描述在陣列中需要的元件數目。

如果這個呼叫是成功的，此平台上實體記憶體位元的數目描述在ret[1]中。

下一個原生方法控制與特定平台上虛擬位址有關資料的快取。

setcachemode(函數名稱)

```
private native void setcachemode(int vasid, long
base, long len, int mode);
```

要進行此操作的特定虛擬位址空間由vasid描述，它是呼叫者將經由對VirtualAddressSpace類別的alloc_vasid()原生方法的呼叫取得。

此原生方法設定對虛擬記憶體的快取，由虛擬位址，base，開始並持續len個位元組的長度。快取根據模式的下列可能值設定(對模式的任何其他值，不做任何事)：

·CACHE_MODE_DEFAULT-根據此平台上對此虛擬記憶體型態的內定值，快取此範圍的虛擬記憶體。注意，甚至在有

五、發明說明 (114)

虛擬位址快取的平台上，依所對映的潛在實體記憶體而定，對映某些實體記憶體(例入輸入／輸出暫存器)的虛擬記憶體可能需要是不快取的，且如果它們需要是不快取的以正確地使用，微核心被期待會使這些虛擬記憶體對映不快取。另外，注意如果所描述的這個範圍包含需要不同地處理之對映到不同型態實體記憶體的虛擬記憶體，如果有必要微核心被期待會不同地快取不同的子範圍，以使整個範圍能正確地快取。除這些限制之外，微核心應試圖設定快取以帶來正確的最大效率。換句話說，如果可以使用快取，而仍維持資料完整性，它們就應該如此。

·CACHE_MODE_INHIBITED-如果快取這個虛擬記憶體是可能的，且使它的快取失去作用是可能的，那麼使它的快取失去作用。否則，不作任何事。如果這對整個範圍是不可能的，但在一或多個子範圍是可能的，至少在任何這樣的子範圍上如此做。

·CACHE_MODE_WRITE_THROUGH-如果快取這個虛擬記憶體是可能的，且將快取設定為write-through模式是可能，則將它設定如此。否則，不作任何事。如果這對整體範圍是不可能的，但對一或多個子範圍是可能的，至少在任何此種子範圍上如此做。

·CACHE_MODE_COPY_BACK-如果快取這個虛擬記憶體是可能的，且將快取設定為copy-back模式是可能的，則將它如此設定。否則，不作任何事。如果這對整體範圍是不可能的，但對一或多個子範圍是可能的，至少在任何此種子

五、發明說明 (115)

範圍上如此做。

getcachemode(函數名稱)

```
private native int getcachemode(int vasid, long
base, long len);
```

此原生方法取得對虛擬記憶體的快取，由虛擬位址、base開始，並連續len個位元組的長度。將傳回下列值中的一個：

・CACHE_MODE_AMBIGUOUS-如果全體的範圍不是以相同方法快取，或下列的回傳值沒有一個適用。

・CACHE_MODE_INHIBITED-如果沒有快取，或如果有快取，但是這個虛擬記憶體範圍現在是不快取的。

・CACHE_MODE_WRITE_THROUGH-如果虛擬記憶體的這個範圍現在是以write-through模式快取的。

・CACHE_MODE_COPY_BACK-如果虛擬記憶體的這個範圍現在以copy-back模式快取的。

flushcache(函數名稱)

```
private native void flushcache(int vasid, long
base, long len);
```

如果虛擬記憶體這個範圍的任何子範圍是以CACHE_MODE_COPY_BACK模式快取，如果可能就清除任何此種快取。如果不可能清除它們，將它們轉換成CACHE_MODE_INHIBITED。

DMAAddressSpace

DMAAddressSpace類別定義下列原生方法：

五、發明說明 (116)

getAddressSize(函數名稱)

```
private native int getAddressSize();
```

傳回DMA位址的位元數目。

alloc_id(函數名稱)

```
private native int alloc_id();
```

由於一般假定只有一個DMA位址空間，微核心只是傳回0。

DMAMemory

DMAMemory類別定義下列的原生方法。這些原生方法的每一個由微核心中的功能實施。

supported(函數名稱)

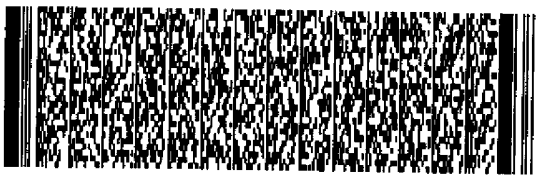
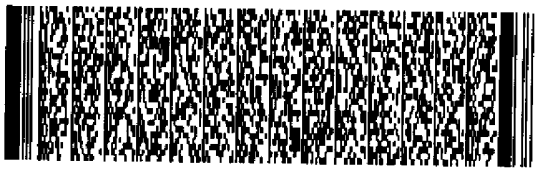
```
private native boolean supported();
```

如果在這個平台上支援DMA傳回真；否則傳回偽。

physmap(函數名稱)

```
private native int physmap(long[] ret, long[]
paddrs, long[) sizes,long minaddr, long
maxaddr, int align, boolean waitformem);
```

要對映的實體記憶體由paddrs和sizes陣列描述，以此種方式paddrs的第n個元件和sizes的第n個元件，分別地描述實體記憶體一個範圍的基底實體位址和長度。實體記憶體的這些範圍將對映到越來越大的DMA位址，這樣子該實體記憶體範圍n(如paddrs和sizes的第n個元件所描述)將對映到比對映到實體記憶體範圍n+1的DMA位址更低的虛擬位址。此外，DMA位址範圍會是連續的，以使它們形成

五、發明說明 (117)

一個DMA位址範圍。

對於此對映，可以使用任何範圍的DMA位址，除了在此範圍裡的所有DMA位址必須比minaddr更大或與其相等，且此範圍裡的所有DMA位址必須比maxaddr更小或與其相等。另外，基底DMA位址將如align所描述對齊，它可以有下列值中之一：

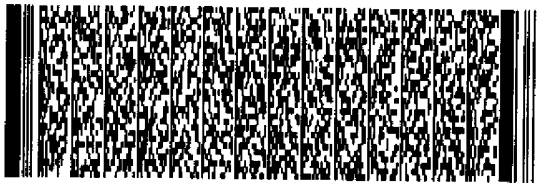
- MemoryConstraints.ALIGN_BYTE(0)-沒有對齊限制。
- MemoryConstraints.ALIGN_SHORT(1)-對齊到短整數。
- MemoryConstraints.ALIGN_INT(2)-對齊到整數。
- MemoryConstraints.ALIGN_LONG(3)-對齊到長整數。
- MemoryConstraints.ALIGN_CACHE(4)-對齊到快取線大小。
- MemoryConstraints.ALIGN_PAGE(5)-對齊到頁大小。

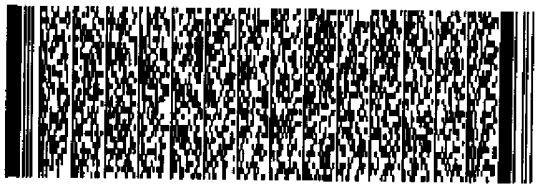
如果waitformem為真，且它目前，由於暫時地資源短缺，不可能完成對映要求，那麼無限期地等待以完成要求。否則，不等候。

如果不能符合所有這些限制，但它們能在較小的必須由實體記憶體第一個範圍開始的子範圍符合，那麼對較小的子範圍對映。舉例來說，如果這個平台並不真的有DVMA(直接虛擬記憶體定址)，且若實體記憶體範圍是非連續的，那麼明顯地只有實體記憶體的第一個範圍能對映。

此範圍的基底DMA位址將在ret[0]中傳回，而所對映的DMA記憶體的長度將在ret[1]中傳回。

可能的回傳值是：





五、發明說明 (118)

・SUCCESS-對映是成功的。不是對映實體記憶體全體，就是至少對映一子範圍。

・ADDRESS_FAILURE-有某些與所描述的實體位址有關的失敗。因某種原因那些位址是未定義實體體記憶體，或實體記憶體無法設定對映。

・ALLOC_FAILURE-在嘗試建立對映中有某些其他重大失敗。

map(函數名稱)

```
private native int map(long[] ret, long size, long
minaddr,long maxaddr, int align, boolean
waitformem);
```

要配置的DMA記憶體總數由size以位元組計算描述。對於這對映，可以使用任何範圍的DMA位址，除了在此範圍裡的所有DMA位址必須比minaddr更大或與其相等，且此範圍裡的所有DMA位址必須比maxaddr更小或與其相等。另外，基底DMA位址將如align所描述對齊，它可以有下列值中之一：

・MemoryConstraints.ALIGN_BYTE(0)-沒有對齊限制。

・MemoryConstraints.ALIGN_SHORT(1)-對齊到短整數。

・MemoryConstraints.ALIGN_INT(2)-對齊到整數。

・MemoryConstraints.ALIGN_LONG(3)-對齊到長整數。

・MemoryConstraints.ALIGN_CACHE(4)-對齊到快取線大小。

・MemoryConstraints.ALIGN_PAGE(5)-對齊到頁大小。

五、發明說明 (119)

如果waitformem為真，且它目前，由於暫時地資源短缺，不可能完成對映要求，那麼無限期地等待以完成要求。否則，不等候。

如果不能符合所有這些限制，但它們能在較小的必須由實體記憶體第一個範圍開始的子範圍符合，那麼對較小的子範圍對映。

此範圍的基底DMA位址將在ret[0]中傳回，而所對映的DMA記憶體的長度將在ret[1]中傳回。

可能的回傳值是：

·SUCCESS-對映是成功的。不是對映實體記憶體全體，就是至少對映一子範圍。

·ADDRESS_FAILURE-有某些與所描述的實體位址有關的失敗。因某種原因那些位址是未定義實體體記憶體，或實體記憶體無法設定對映。

·ALLOC_FAILURE-在嘗試建立對映中有某些其他重大失敗。

unmap(函數名稱)

```
private native int unmap(long base, long len);
```

要解除對映的DMA記憶體是由base開始，且有len個位元組的長度。

可能的回傳值是：

·SUCCESS-解除對映是成功的。所描述的整個範圍被解除對映。

·ALLOC_FAILURE-在嘗試進行解除對映中有某些其他重大

五、發明說明 (120)

失敗。

getphys(函數名稱)

private native int getphys(int[] ret, long[] pBase, long[] pLen, long base,long len);

DMA記憶體的範圍的基底由base描述，而範圍的長度由len以位元組計算描述。

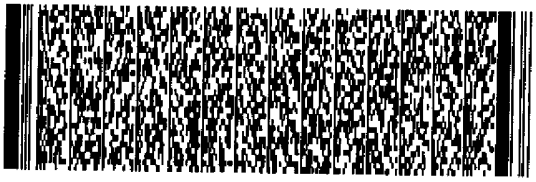
實體記憶體將在所提供的pBase和pLen陣列描述，如此pBase[0]和pLen[0]是開始對映DMA記憶體的起點之實體記憶體的基底和長度，且如此(pBase[n+1]，pLen[n+1])描述對映較高範圍DMA位址的實體記憶體，相較於對映到(pBase[n]，pLen[n])所描述實體記憶體的DMA位址的範圍。

呼叫者不可能知道實體記憶體多少不連續的範圍對映這個DMA記憶體範圍，所以有可能對pBase與pLen所提供陣列太小。如果是這種情形，微核心將傳回ARRAY_TOO_SMALL並在ret[0]中描述在陣列中需要的元件數目。

如果這個呼叫是成功的，此平台上實體記憶體位元的數目描述在ret[1]中。

AccessibleMemory子類別

AccessibleMemory的各種非抽象子類別如AccessibleMemory摘錄出的原生方法定義。這些子類別由SwappedVirtualIOMemory，UnSwappedVirtualIOMemory，SwappedVirtualRegularMemory，UnSwappedVirtualRegularMemory，



五、發明說明 (121)

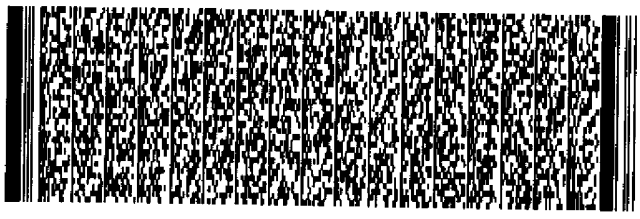
SwappedPortIOMemory，和UnSwappedPortIOMemory所組成。這些子類別的每一個有這些原生方法個別的版本。

這些原生方法的個別版本大體上彼此類似，它們的差異以下列的附加描敘：

・Swapped/UnSwapped-對交換類別的原生方法之版本(SwappedVirtualIOMemory、SwappedVirtualRegularMemory和SwappedPortIOMemory)在對其操作之前必須位元組交換(顛倒序法)它們由原生方法取得的任何資料，且必須位元組交換回復它們所提供給類別的任何資料。其他類別(UnSwappedVirtualIOMemory、UnSwappedVirtualRegularMemory和UnSwappedPortIOMemory)不需要做任何此種位元組交換。

・IO/Regular-支援輸入輸出記憶體(SwappedVirtualIOMemory和UnSwappedVirtualIOMemory)的類別在對該記憶體做存取時，可能需要特別的平台特定(實施相關)運作。支援一般隨機存取記憶體(SwappedVirtualRegularMemory和UnSwappedVirtualRegularMemory)的類別不需要任何此種特別的平台特定運作。注意在一些平台上，沒有差異。

・VirtualPort-虛擬記憶體的類別(SwappedVirtualIOMemory、UnSwappedVirtualIOMemory、SwappedVirtualRegularMemory和



五、發明說明 (122)

UnSwappedVirtualRegularMemory)可假定記憶體經由標準載入與儲存指令存取，且原生方法中描述的位址參照虛擬記憶體位址。埠記憶體的類別(SwappedPortIOMemory和UnSwappedPortIOMemory)必須做特別的輸入／輸出指令以存取記憶體。

上述附加規格模數，下列各項描述這六組原生方法的每一個：

supported(函數名稱)

```
protected native boolean supported();
```

傳回真，如果在這個平台上支援AccessibleMemory的這個特定子類別；否則傳回偽。

ncksum(函數名稱)

```
protected native int ncksum(long adr, long len);
```

計算由位址adr開始長度len個位元組，記憶體的位元組的總和(模數0x10000)。

nSetByte(函數名稱)

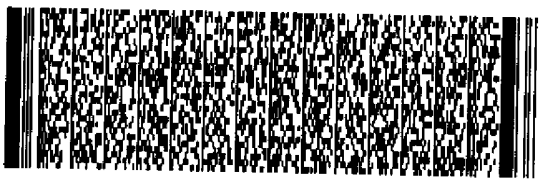
```
protected native void nsetByte(long adr, byte x);
```

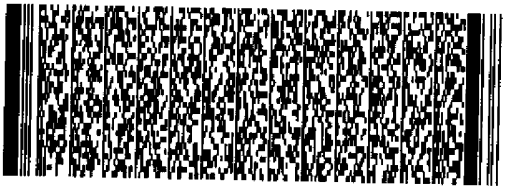
這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面)：IO/Regular, Virtual/Port。

位元組值，x，將寫在位址，adr。

nSetShort(函數名稱)

```
protected native void nsetshort(long adr, short x);
```





五、發明說明 (123)

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped;UnSwapped,IO/Regular,Virtual/Port。

短整數值，x，將寫在位址，adr。

nSetInt(函數名稱)

protected native void nSetInt(long adr, int x);

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped/UnSwapped, IO/Regular, Virtual/Port。

整數值，x，將寫在位址，adr。

nSetLong(函數名稱)

protected native void nsetLong(long adr, long x);

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped/UnSwapped, IO/Regular, Virtual/Port。

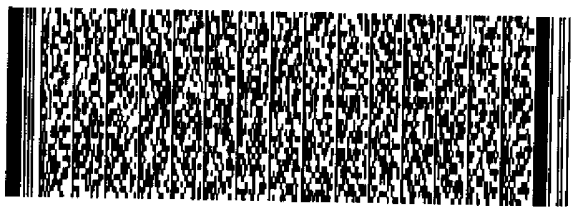
長整數值，x，將寫在位址，adr。

nSetBytes(函數名稱)

protected native void nsetBytes(long adr, byte bytes[], int bytesOffset, int length);

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped/UnSwapped，IO/Regular，Virtual/Port。

由陣列，bytes，的偏移量，bytesOffset，開始，位元組的陣列將被寫到由位址，adr，開始總長度，length，



五、發明說明 (124)

個位元組。

nSetIntArmy(函數名稱)

protected native void nSetIntArray(long adr, byte bytes[], int bytesoffset, int length_int);

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):Swapped/UnSwapped，IO/Regular，Virtual/Port。

由陣列，bytes，的偏移量，bytesOffset，開始，位元組的陣列將被，當成整數的列處理，寫到由位址，adr，開始總長度，length_int，個整數值。

nGetByte(函數名稱)

protected native byte nGetByte(long adr);

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):IO/Regular，Virtual/Port。

由位址，adr，讀取並傳回一位元組。

nGetShort(函數名稱)

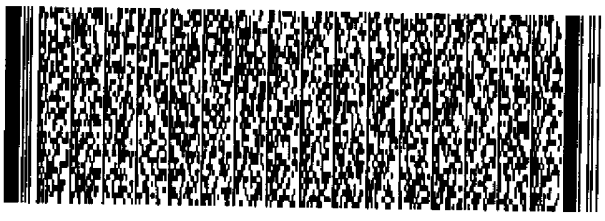
protected native short nGetshort(long adr);

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):Swapped/UnSwapped，IO/Regular，Virtual/Port。

由位址，adr，讀取並傳回一短整數。

nGetInt(函數名稱)

protected native int nGetInt(long adr);



五、發明說明 (125)

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped/UnSwapped，IO/Regular，Virtual/Port。

由位址，adr，讀取並傳回一整數。

nGetLong(函數名稱)

```
protected native long nGetLong(long adr);
```

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped/UnSwapped，IO/Regular，Virtual/Port。

由位址，adr，讀取並傳回一長整數。

nGetBytes(函數名稱)

```
protected native void nGetBytes(long adr, byte
bytes[], int bytesOffset, int length);
```

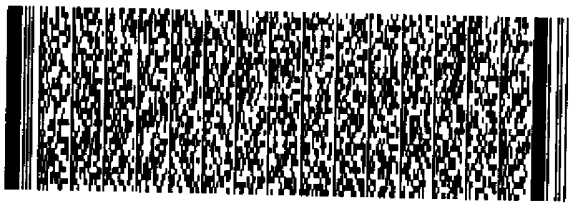
這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped/UnSwapped，IO/Regular，Virtual/Port。

由adr開始讀取位元組總共length個位元組，並將它們讀到陣列bytes中，由該陣列偏移量bytesOffset開始。

nGetIntArray(函數名稱)

```
protected native void nGetIntArray(long adr, byte
bytes[], int bytesOffset, int length_int);
```

這個運作必須對照此原生方法支援的AccessibleMemory的特定子類別的下列特別的方式(見上面):
Swapped/UnSwapped，IO/Regular，Virtual/Port。



五、發明說明 (126)

由adr開始讀取整數總共lengthint個整數，並將它們讀到陣列bytes中，由該陣列偏移量bytesOffset開始。

nRegisterSource(函數名稱)

```
protected native void nRegisterSource(long adrl,
long len);
```

這個運作記錄一複製運作的來源，並傳回一識別，它可以在之後傳給nCopy原生方法。來源的基底位址，adrl，和要複製的位元組長度，len，將被記錄。根據來源記憶體的特定性質考慮並記錄下列的方式也是必需的：Swapped/UnSwapped, IO/Regular, Virtual/Port。

nCopy(函數名稱)

```
protected native void nCopy(long adr2, long len,
int copyID);
```

此運作執行由copyID所描述的來源複製，到這裡描述的目的。

如果copyID不符合所記錄的來源，不做任何事。

目的位址，adr2，和位元組長度，len，如所描述。來源位址和長度可由記錄為來源者取得，如copyID所描述。如果來源和目的的長度不一致，使用二者中較小的。

在執行此複製時，考慮下列的方式也是必需的：Swapped/UnSwapped, IO/Regular, Virtual/Port。

以上描述的類別儲存在電腦可讀的媒介上，例如軟式磁碟，光碟或光學磁碟。或替代地，它們在唯讀記憶體裡提供給電腦系統10，或在網路上以電腦資料載波的型式提

五、發明說明 (127)

供。

對習知該項技藝人士會是顯而易見的，各種修改和變更能對所揭露的處理和產品完成而不離開本發明的範圍或精神。本發明的其他具體實施例對習知該項技藝人士，參照本說明書與本發明在此揭露的實施會是顯而易見的。本說明書與範例僅係用以說明，本發明以下的申請專利範圍的真正範疇與精神。

四、中文發明摘要 （發明之名稱：用於物件導向記憶系統之裝置及方法）

一種方法和裝置，用以提供記憶功能給一電腦系統的物件導向用戶端軟體元件，具有一中央處理單元，使用一第一組記憶體類別，第一組的每個類別是平台獨立的；一第二組記憶體類別，該第二組的每個類別是該第一組類別的一子類別，且是平台相關的；並藉由僅存取第一組類別的物件執行用戶端元件記憶函數。提供類別和方法的說明，以使平台獨立的裝置驅動程式能夠實現。

英文發明摘要 （發明之名稱：APPARATUS AND METHOD FOR OBJECT-ORIENTED MEMORY SYSTEM）

A method and apparatus for providing memory functionality to object-oriented client software components of a computer system having a CPU using a first set of memory classes, each class of the first set being platform-independent; a second set of memory classes, each class of the second set being a subclass of a class of the first set and being platform-dependent; and performing client component memory functions by accessing only objects of classes of the first set. Descriptions

四、中文發明摘要　（發明之名稱：用於物件導向記憶系統之裝置及方法）

英文發明摘要　（發明之名稱：APPARATUS AND METHOD FOR OBJECT-ORIENTED MEMORY SYSTEM）

of the classes and methods are provided, enabling platform-independent device drivers to be implemented.

六、申請專利範圍

1. 一種方法，用以提供記憶體功能給一具有中央處理單元電腦系統之物件導向用戶端軟體元件，包含步驟：

提供一第一組記憶體類別，該第一組的每個類別是平台獨立的；

提供一第二組記憶體類別，該第二組的每個類別是一第一組類別的子類別，且是平台相關的；和

藉由只存取第一組類別的物件，執行用戶端元件記憶體函數。

2. 如申請專利範圍第1項的方法，其中提供一第一組記憶體類別的步驟，包括提供只有基底位址、長度和限制的屬性之一第一抽象記憶體類別的次步驟。

3. 如申請專利範圍第2項的方法，其中提供一第一組記憶體類別的步驟，包括提供記憶體的一第一可例示的類別的次步驟，它擴展第一抽象記憶體類別且不提供第一抽象記憶體類別所提供以外的功能。

4. 如申請專利範圍第2項的方法，其中提供一第一組記憶體類別的步驟，包括提供一第二抽象記憶體類別的次步驟，它擴展第一抽象記憶體類別只描述用來控制快取記憶體的抽象方法。

5. 如申請專利範圍第4項的方法，其中提供一第一組記憶體類別的步驟，包括提供一第三抽象記憶體類別的次步驟，擴展第二抽象記憶體類別並代表所有可被中央處理單元直接地存取的記憶體。

6. 如申請專利範圍第5項的方法，其中提供一第三抽象

六、申請專利範圍

記憶體類別的次步驟，包括提供定義用來存取記憶體的平台獨立抽象方法的，一第三抽象記憶體類別的次步驟。

7. 如申請專利範圍第1項的方法，其中提供一第二組記憶體類別的步驟，包括提供一第二可例示的記憶體類別的次步驟，擴展代表實體記憶體的第二抽象記憶體類別。

8. 如申請專利範圍第7項的方法，其中提供一第二可例示的記憶體類別的次步驟，包括提供一第二可例示的記憶體類別的次步驟，它提供第二抽象記憶體類別中摘錄出來的快取管理方法的實施。

9. 如申請專利範圍第8項的方法，其中提供一第二組記憶體類別的步驟，包括提供一第四抽象記憶體類別的次步驟，它擴展第三抽象記憶體類別，並包含用以設定對映到第二可例示的記憶體類別的物件之方法，及用以配置一特定數量的虛擬記憶體而不描述特定實體記憶體的方法。

10. 如申請專利範圍第1項的方法，其中提供一第二組記憶體類別的步驟，包括提供一第三可例示的記憶體類別的次步驟，它擴展第一抽象記憶體類別，用以設定DMA對映到實體記憶體，及用以進行對應的解除對映。

11. 如申請專利範圍第1項的方法，其中提供一第二組記憶體類別的步驟，包括提供一第二組記憶體類別的次步驟，其中的方法分成兩個群組，第一群組以非本機語言寫成，而第二群組以本機語言寫成。

12. 如申請專利範圍第11項的方法，其中提供一第二組記憶體類別的步驟，包括將第二群組方法提供在存取微核

六、申請專利範圍

心的函數程式庫中的次步驟。

13. 如申請專利範圍第12項的方法，其中：

提供一第一組記憶體類別的步驟，包括提供一第二抽象記憶體類別的次步驟，它擴展第一抽象記憶體類別且只描述用來快取記憶體的抽象方法；及

提供一第二組記憶體類別的步驟，包括在第二群組方法中，提供對應於第二抽象記憶體類別中所定義的抽象快取管理方法之原生方法。

14. 一種裝置，用以在一物件導向電腦系統中執行記憶體函數，包含：

一第一組記憶體類別，該第一組的每個類別是平台獨立的；

一第二組記憶體類別，該第二組的每個類別是一第一組類別的子類別，且是平台相關的；

一裝置驅動程式，只存取第一組類別的物件；及

一匯流排管理器，存取第二組類別。

15. 一種電腦可讀媒介，包含用來提供記憶體功能給一具有中央處理單元之電腦系統的物件導向用戶端軟體元件的指令，包含：

一第一組記憶體類別，該第一組的每個類別是平台獨立的；及

一第二組記憶體類別，該第二組的每個類別是一第一組類別的子類別，且是平台相關的。

16. 如申請專利範圍第15項的電腦可讀媒介，其中第二

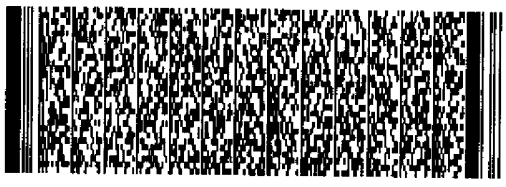
六、申請專利範圍

組類別包括以平台獨立語言寫成的第一子組方法，及以本機程式碼寫成的第二子組方法。

17. 一種載波上的電腦資料訊號，包含用來提供記憶體功能給一具有中央處理單元之電腦系統的物件導向用戶端軟體元件的指令，包含：

一第一組記憶體類別，該第一組的每個類別是平台獨立的；及

一第二組記憶體類別，該第二組的每個類別是一第一組類別的子類別，且是平台相關的。



圖式

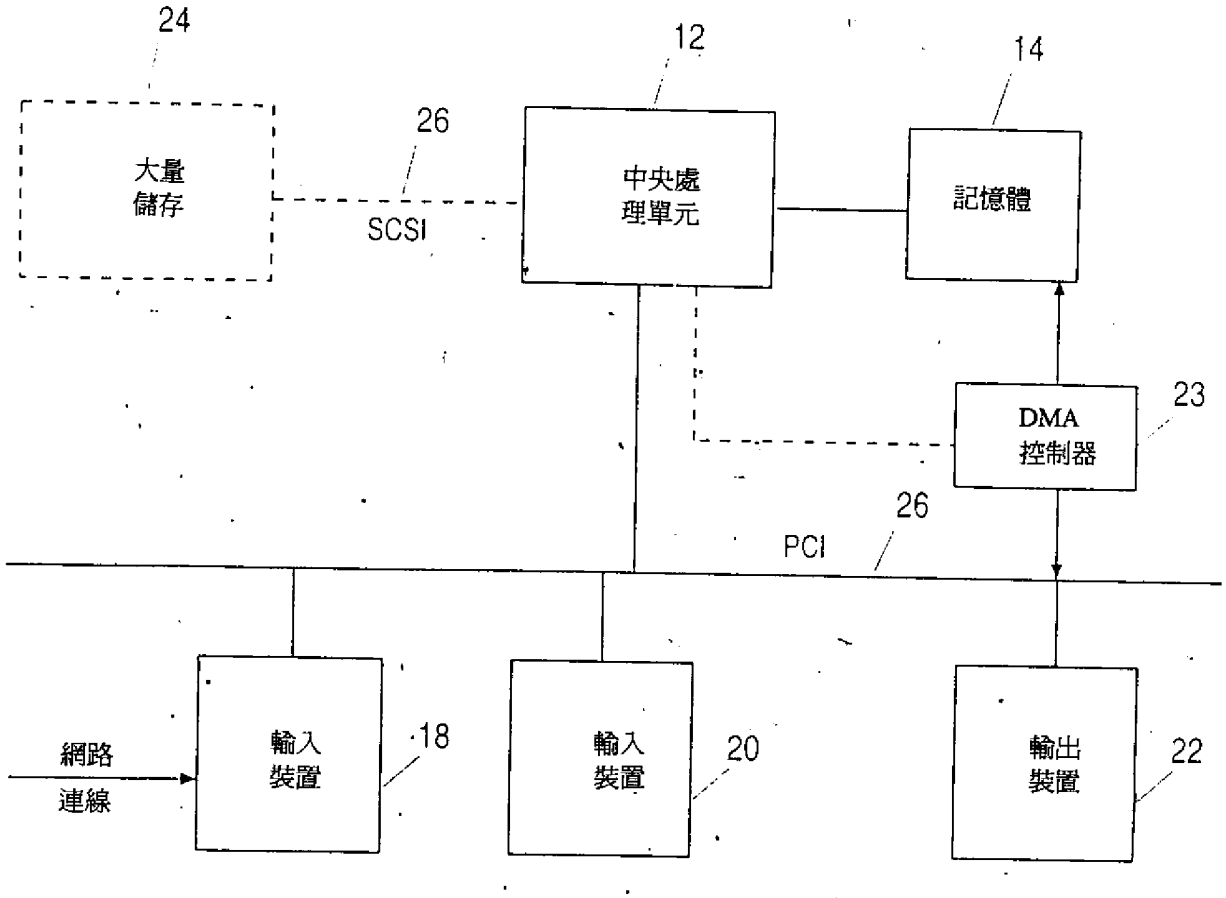

24
12
14
26
大量
儲存
SCSI
中央處
理單元
記憶體
DMA
控制器
23
26
PCI
網路
連線
輸入
裝置
18
輸入
裝置
20
輸出
裝置
22

圖1

圖式

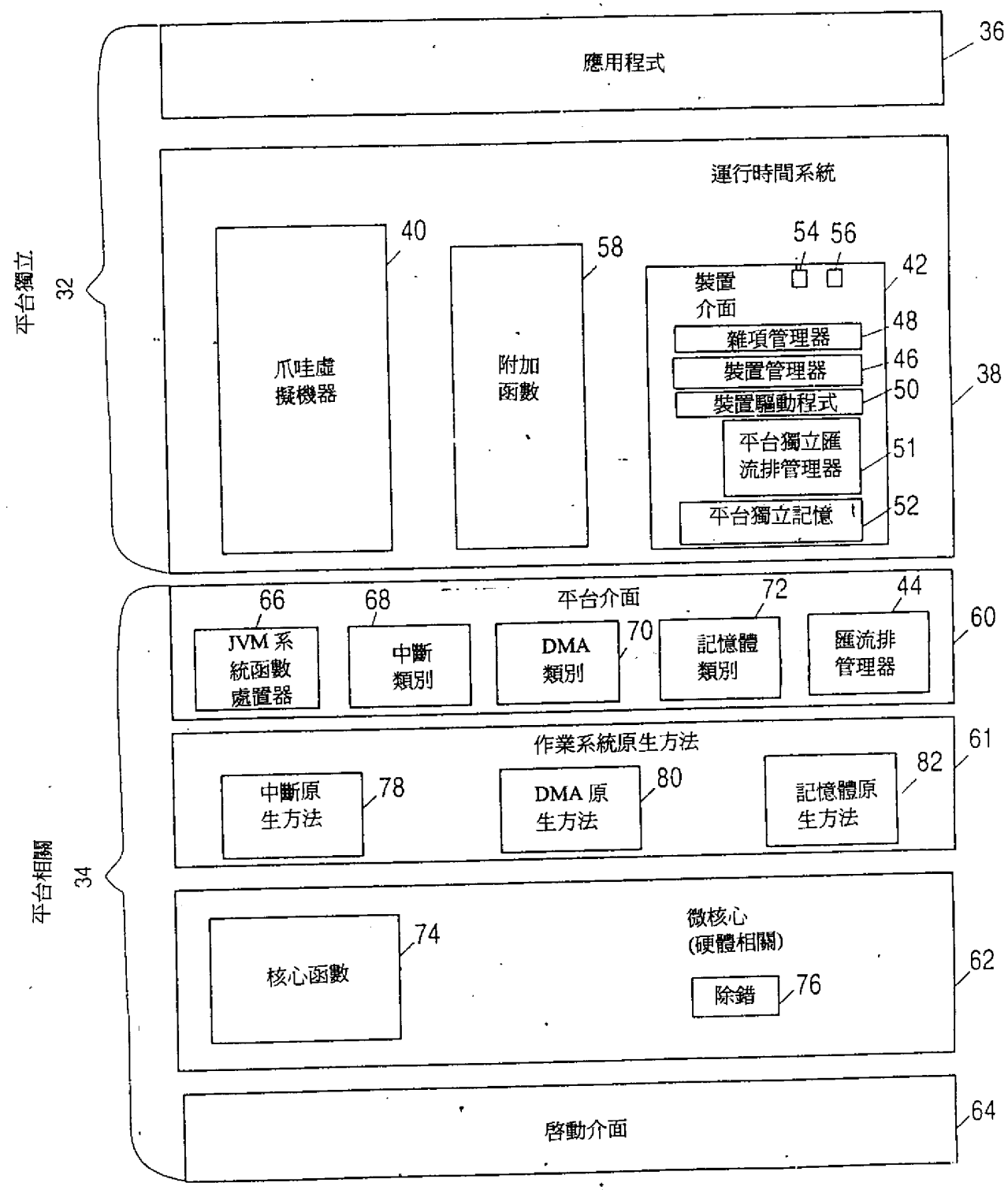

應用程式
36
運行時間系統
平台獨立
32
爪哇虛擬機器
40
附加函數
58
54
56
裝置介面
42
雜項管理器
48
裝置管理器
46
裝置驅動程式
50
38
平台獨立匯流排管理器
51
平台獨立記憶
52
平台介面
66
JVM 系統函數處置器
68
中斷類別
DMA 類別
70
72
記憶體類別
44
匯流排管理器
60
作業系統原生方法
61
中斷原生方法
78
DMA 原生方法
80
記憶體原生方法
82
平台相關
34
核心函數
74
微核心
(硬體相關)
除錯
76
62
啓動介面
64

圖2

圖式

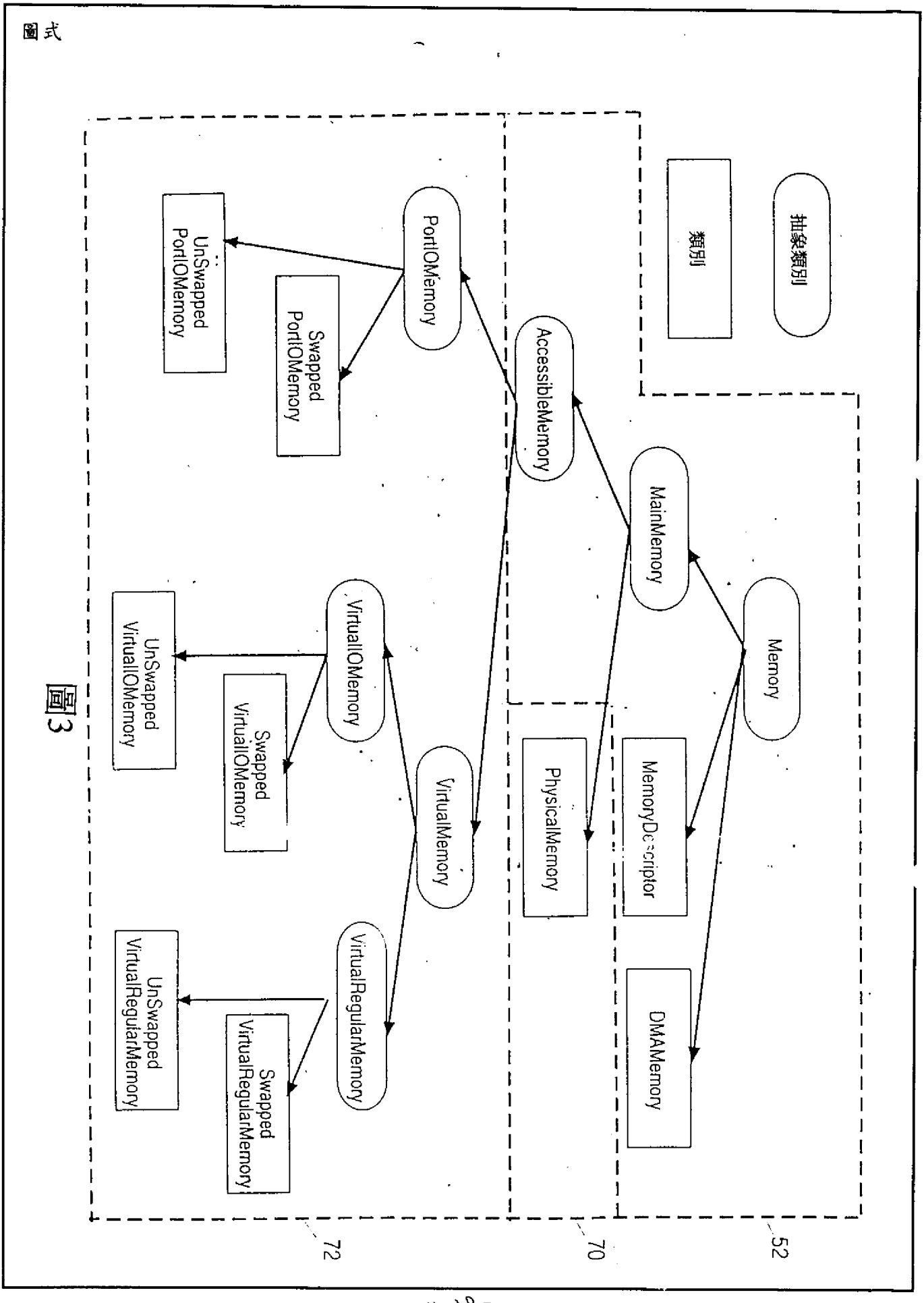

抽象類別
類別
Memory
MainMemory
MemoryDescriptor
DMAMemory
PhysicalMemory
AccessibleMemory
PortIOMemory
UnSwapped PortIOMemory
Swapped PortIOMemory
VirtualIOMemory
UnSwapped VirtualIOMemory
Swapped VirtualIOMemory
VirtualMemory
VirtualRegularMemory
UnSwapped VirtualRegularMemory
Swapped VirtualRegularMemory
52
70
72

圖3