第三章 Intel XScale 系統晶片核心介紹

3.1 簡介

Intel XScale 的核心架構是沿襲自 ARM 5TE 版本，其設計上有著高效能、低耗功率的特性。此核心並非設計成一項獨立的產品，而是為了嵌入式市場建立一個符合 ASSP (Application Specific Standard Product) 標準的區塊模組。其架構上特性，讓程式設計者能針對與其應用程式相關的特性來做設計而達到更好的效能。

Intel XScale 的核心提供拇指指令集 (Thumb instruction set) 以及數位訊號處理 (DSP) 的延伸功能。另外透過 16 位元的資料型態與 16 位元的運算，也能有效率的操作其聲音的裝置。

3.1.1 前言

Intel XScale 的核心架構是沿襲自 ARM 5TE 版本，其設計上有著高效能、低耗功率的特性。此核心並非設計成一項獨立的產品，而是為了嵌入式市場建立一個符合 ASSP（Application Specific Standard Product）標準的區塊模組。其架構上特性，讓程式設計者能針對與其應用程式相關的特性來做設計而達到更好的效能。

Intel XScale 的核心提供拇指指令集（Thumb instruction set）以及數位訊號處理（DSP）的延伸功能。另外透過 16 位元的資料型態與 16 位元的運算，也能有效率的操作其聲音的裝置。

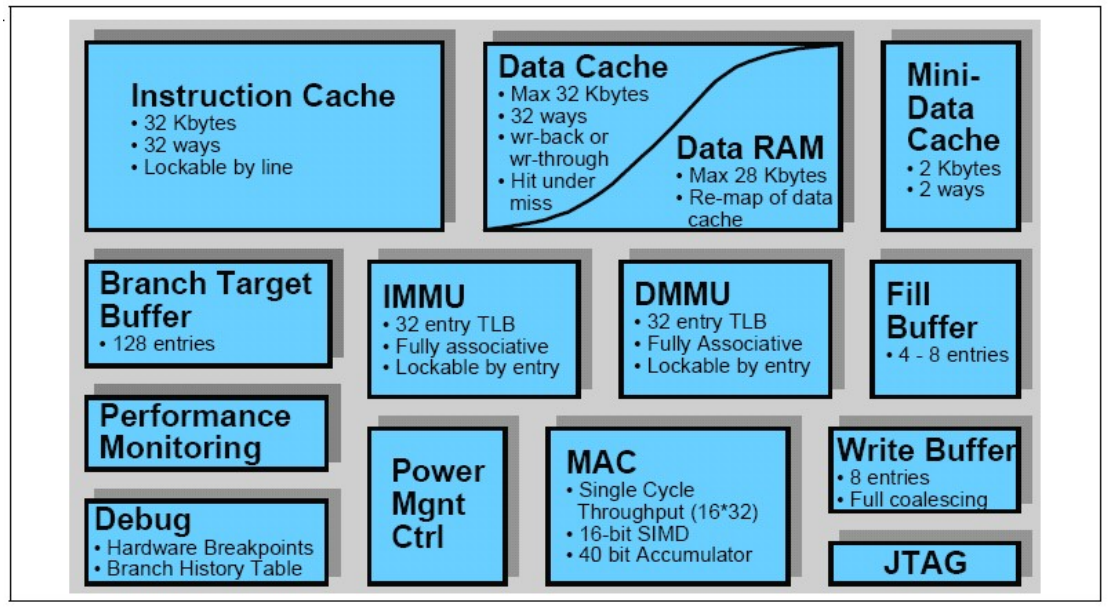
本章先大略簡介 XScale 核心的功能區塊，在後面數個章節會再對每個功能區塊一一的詳述。章節分布如下：

- 第 3.2 節：程式模型
- 第 3.3 節：記憶體管理
- 第 3.4 節：指令快取

另外，附錄中會再介紹：

- 附錄 A：最佳化導引 （ Optimization Guide ），介紹指令排程的技巧。
- 附錄 B：測試的特色 （ Test Features ） 則詳述 JTAG 這個單元。

3.1.2 XScale 核心功能區塊



3.1.2.1 加乘單元（Multiply/ACcumulate, MAC）

MAC 提供能於兩個週期完成加乘運算，以及維持每個週期能執行一個 MAC 運算指令的能力。而數項 MAC 架構上的強化，以及使用二個 16 位元的 SIMD（Single Instruction/Multiple Data）與一個 40 位元累進器（Accumulator）的相乘，可以有效率的執行聲音的處理。

3.1.2.2 記憶體管理

Intel XScale 核心所採用的記憶體管理單元（Memory Management Unit, MMU），其提供存取的保護，及虛擬位址與實體位址間的轉換。

MMU 架構亦描述了指令快取和資料快取的快取策略，其策略包括：

- 可設定程式碼為快取或不快取
- 選擇資料快取或迷你資料快取
- 批次回寫（write-back）或逐一回寫（write-through）資料快取
- 可啟動資料寫入配置策略

- 可啟動寫入緩衝區將資料直接儲存至外部記憶體

3.1.2.3　指令快取

Intel XScale 核心提供一個 32K 位元組的指令快取，是由 32 組（set）、每組 32 路（way）、每路有 32 位元為一行（line）的字集（word）所組成的。當向指令快取提出要求卻發生「未擊中」（miss）的狀況時，會向外部記憶體產生一組 32 位元組的要求。另外也提供在快取內鎖定關鍵碼的機制。

3.1.2.4　分支目的緩衝區

Intel XScale 核心提供一個分支目的緩衝區以預測分支型態指令的結果，它提供一個儲存空間儲存分支型態指令之目的位址，以及預測當目前指令位址是一個分支時，其出現在指令快取中的下一個位址。

3.1.2.5　資料快取

Intel XScale 核心提供一個 32K 位元組之資料快取及一個 2K 位元組之迷你資料快取，此兩者快取每條皆為 32 位元組，提供逐一回寫或全部回寫的快取方式。

資料快取和迷你資料快取是由定義在 MMU 架構之頁屬性及協同處理器 15 所控制。

Intel XScale 核心允許應用程式重新設定一部分的資料快取作為資料隨機存取記憶體（data RAM），但軟體上需放置特殊表格或經常使用的變數在 RAM 上。

3.1.2.6　效能監控

兩個效能監控計數器加入至 Intel XScale 核心中，可設定來監控 Intel XScale 核心中不同的事件，這些事件允許軟體發展者來測量快取效率、偵測系統瓶頸、及降低程式的所有延遲。

3.1.2.7　能源管理

Intel XScale 核心結合電源與時計管理單元，用以控制時間及管理電源。

3.1.2.8　偵錯

Intel XScale 核心透過兩個指令位址中斷暫存器、一個資料位址中斷暫存器、一個資料位址/遮罩中斷暫存器、及一個追蹤緩衝區以提供軟體偵錯。

3.1.2.9 JTAG

Intel XScale 核心透過以 IEEE 1149.1（JTAG）標準測試埠及邊界掃描為架構之測試存取埠（Test Access Port, TAP）控制器來提供測試能力。TAP 控制器的目的是提供如建立自我測試、邊界掃描、及掃描等內、外部邏輯至 Intel XScale 核心的測試。

3.2 程式模型（Programming Model）

3.2.1 與 ARM 架構相容

Intel XScale 核心如 ARM 5TE 版本所描述，進行整數指令集架構。T 代表拇指（Thumb）指令集、E 表示數位訊號處理（DSP）增強指令集。

ARM 版本 5 的介紹較版本 4 多一些架構特性，特別是迷你頁的增加（1K 位元組）、一個計算資料其前導零的新指令、加強 ARM-Thumb 傳輸指令、和一個系統控制協同處理器 CP15 的修正。

3.2.2 ARM 架構實行之選擇

3.2.2.1 高位元組在前與低位元組在前

Intel XScale 核心提供 big endian 和 little endian 兩種資料的表示法。控制暫存器的 B 位元（協同處理器 15 中，暫存器 1 的第 7 位元）是負責選擇要使用 big endian 或 little endian。在 big endian 模式下執行，B 位元必須是於企圖在記憶體中存取任何子字集之前設定，否則會有未定義的結果發生。注意此位元即使是在 MMU 未啟動下仍會有作用。

3.2.2.2 26 位元碼

Intel XScale 核心不支援 26 位元碼。

3.2.2.3 Thumb

Intel XScale 核心支援拇指指令集。

3.2.2.4 ARM DSP 增強指令集

Intel XScale 核心提供了一些可以提高訊號處理應用程式效能之 ARM DSP 增強指令集，這些是 16 位元資料與新的飽和（saturation）指令運算之新乘法指令。一些新的指令如：

- SMLAxy 32<=16x16+32
- SMLAWy 32<=32x16+32
- SMLALxy 64<=16x16+64
- SMULxy 32<=16x16
- SMULWy 32<=32x16
- QADD 兩個暫存器相加，如果發生溢位就飽和結果
- QDADD 一個輸入暫存器加倍且飽和後再相加、飽和
- QSUB 兩個暫存器相減，如果發生溢位就飽和結果
- QDSUB 一個輸入暫存器加倍且飽和後再相減、飽和

Intel XScale 核心也提供 LDRD、STRD、PLD 等指令，其運作方式為：

- PLD 在 MMU 下是當作讀取指令，但在資料中斷點單元下則是被忽略，換言之，PLD 絕不會產生資料中斷點事件。
- PLD 在沒有快取的頁面是不會執行的，而且如果目的地的快取線已經常駐，這個指令也是不會有作用。
- 當位址位元[2:0]為 0b100 時，LDRD 和 STRD 兩個指令都將產生校正上的例外（alignment exception）。

Intel XScale 核心只有在當指向協同處理器 0 及存取內部的累加器時才有支援 MCRR 和 MRRC 這兩個指令。存取協同處理器 15 及 14 會產生一個未定義指令的例外。

3.2.2.5 基底暫存器更新

如果一個資料中止是在一個指定要回寫的記憶體指令的訊號，基底暫存器就不會被更新，這對所有的載入與儲存的指令都有效。

3.2.3 ARM 架構擴充

Intel XScale 核心較 ARM5 版本的架構做了些許的擴充，是爲了要符合不同的市場與設計上的需求，以下條列擴充的內容：

- 加入了包含一個 40 位元累加器與 8 個新指令的 DSP 協同處理器 0（CP0）。
- 在頁面表格描述符號中加入了新的頁面屬性，多了一個位元擴充了 C 和 B 頁面屬性編碼更多些：寫入配置與迷你資料快取。另外也加入了一個 ASSP 可定義的屬性（P 位元）。
- 協同處理器 14 與 15 都有加入額外的功能。
- 事件架構、指令快取和資料快取同位元錯誤的例外、中斷點事件、以及不正確的外部資料中止等都有做增強。

3.2.3.1 DSP 之協同處理器 0（CP0）

Intel XScale 核心於架構中加入了一個 DSP 協同處理器，爲了增加聲音處理演算法的效能與精確性，這個協同處理器包含了一個 40 位元的累加器與 8 個新的指令。

有數個加入到架構中之心的指令會參考到這 40 位元的累加器，MIA、MIAPH、及 MIAxy 這幾個加/乘指令會參考 40 位元累加器而非暫存器內所指令的累加器；MAR 和 MRA 則提供讀寫 40 位元累加器的能力。

當協同處理器存取暫存器（Coprocessor Access Register）的 0 位元被設定後，在所有處理器模式下都允許存取 CP0；而當此 0 位元被清除後，任何存取 CP0 將導致出現未定義的例外。但注意，僅有特定的軟體可以在協同處理器存取暫存器設定此位元。

當多重處理在使用此 40 位元累加器時，此累加器內容需儲存在一個可資料交換的地方。

協同處理器 0 加入了兩個新的指令格式：乘法與內部累加的格式（Multiply with Internal Accumulate Format）及存取內部累加器的格式（Internal Accumulator Access Format），這兩種格式及指令將在以下詳細敘述。

3.2.3.1.1 乘法與內部累加的格式

一個定義在 40 位元累加器上運作的新乘法格式，如表 3-1「乘法與內部累加格式」所示，這個格式的指令碼是位於協同處理器暫存器傳輸指令型態內，這些指令有屬於自己的語法。

表3-1　乘法與內部累加格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond				1	1	1	0	0	0	1	0	opcode_3				Rs				0	0	0	0	acc				1		Rm	

位元	敘述	備註
31：28	cond – ARM 狀態碼	-
19：16	opcode_3 – 描述乘法使用在內部累加的型態	0b0000 = MIA 0b1000 = MIAPH 0b1100 = MIABB 0b1101 = MIABT 0b1110 = MIATB 0b1111 = MIATT 其餘的編碼結果是不可預期的
15：12	Rs – 乘數	
7：5	acc – 選擇八個累加器之一	Intel XScale 核心僅使用 acc0，存取其他的 acc 將會導致無可預期的影響
3：0	Rm – 被乘數	-

在這個格式中有兩個新的欄位：acc 及 opcode_3。Acc 欄位指定八個累加器的其中一個來運算；opcode_3 則定義運算。Intel XScale 核心定義僅使用單一個 40 位元的累加器 ─acc0，未來有可能會在定義多重的內部累加器；而 opcode_3 中則可以指定六種指令：MIA、MIAPH、MIABB、MIABT、MIATB、及 MIATT。

表3-2　MIA{<cond>}acc0, Rm, Rs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	1	0	0	0	1	0	0	0	0	0	Rs				0	0	0	0	0	0	0	1	Rm			

運算：if ConditionPassed（<cond>） then acc0 = （ Rm[31：0] * Rs[31：0] ） [39：0] + acc0[39：0]
例外：無
合格狀況碼：無狀況碼旗標會被更新
備註：有提供初期中止。指令時間將於10.4.4章節的「多重指令時間」中詳述。暫存器Rs和Rm若用到R15會有不可預期的結果。Intel XScale核心定義

acc0值為0b000。

除了 40 位元累加器的使用外，MIA 指令運作類似於 MLA。MIA 乘法是將有正負號值得暫存器Rs和Rm相乘，並把結果累加到40位元的累加器中（acc0）。

MIA 不支援無正負號的乘法，所有在 Rs 和 Rm 中的值都將被解釋成有正負號的資料值。由 LDRSH 指令將有正負號的 16 位元資料載入到一般用途型暫存器（general purpose register）後，MIA 再來運算。

這個指令只有在其前四個位元 cond 符合狀況碼的狀態才會執行。

表3-3　MIAPH{<cond>}acc0, Rm, Rs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	1	0	0	0	1	0	1	0	0	0	Rs				0	0	0	0	0	0	0	1	Rm			

運算：if ConditionPassed（<cond>） then
　　acc0 = sign_extend（ Rm[31：16] * Rs[31：16] ） +
　　sign_extend（ Rm[15：0] * Rs[15：0] ） +
　　acc0[39：0]
例外：無
合格狀況碼：S位元都是被清為0；無狀況碼旗標會被更新
備註：指令時間將於10.4.4章節的「多重指令時間」中詳述。暫存器Rs和Rm若用到R15會有不可預期的結果。Intel XScale核心定義acc0值為 0b000。

MIAPH 指令執行兩個具正負號的 16 位元資料相乘，再將結果累加到 40 位元的累加器中。第一段的乘法是乘數的低位 16 位元與被乘數的低位 16 位元相乘；第二段的乘法是乘數的高位 16 位元與被乘數的高位 16 位元相乘；最後再將以上兩個 32 位元的結果擴展成 40 位元再累加至 40 位元的累加器中（acc0）。

這個指令只有在其前四個位元 cond 符合狀況碼的狀態才會執行。

表3-4　MIAxy{<cond>}acc0, Rm, Rs

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	1	0	0	0	1	0	1	1	x	y	Rs				0	0	0	0	0	0	0	1	Rm			

運算：if ConditionPassed（<cond>） then
　　if （ bit[17] == 0 ）
　　<operand1> = Rm[15：0]

```
        else
          <operand1> = Rm[31：16]

        if （ bit[16] == 0 ）
          <operand2> = Rs[15：0]
        else
          <operand2> = Rs[31：16]

        acc0[39：0] = sign_extend（<operand1> * <operand2>） + acc0[39：0]
```

例外：無
合格狀況碼：S位元都是被清為0；無狀況碼旗標會被更新
備註：指令時間將於10.4.4章節的「多重指令時間」中詳述。暫存器Rs和Rm若用到R15會有不可預期的結果。Intel XScale核心定義acc0值為0b000。

MIAxy 指令執行具正負號的 16 位元乘法運算，並在疊加到 40 位元的累加器中。X 代表被乘數 Rm 要取上半段或下半段 16 位元的資料，y 則代表乘數 Rs 要取上半或下半段 16 位元的資料。若值是 0x1 時，代表取上半段即[31：16]位元的資料，助記符號為 T（top）；若值是 0x0 時，代表取下半段即[15：0]位元的資料，助記符號為 B（bottom）。

MIAxy 不支援無正負號的乘法，所有 Rs 和 Rm 的值都被解釋為有正負號值。

這個指令只有在其前四個位元 cond 符合狀況碼的狀態才會執行。

3.2.3.1.2　存取內部累加器的格式

Intel XScale 核心為了在 CP0 內存取內部累加器定義了一個新的指令格式。表 3-5「存取內部累加器的格式」顯示指令碼屬於協同處理器暫存器傳輸空間。

RdHi 及 RdLo 欄位允許在 StrongARM 暫存器與內部累加器之間的至多 64 位元的資料傳輸。Acc 指定 8 個內部累加器的其中一個來收送資料，在 Intel XScale 核心只使用 acc0 這個 40 位元的累加器，未來可以使用多重內部累加器做至多 64 位元的變動資料大小的處理。

當協同處理器存取暫存器的 0 位元被設定，在所有處理器模式（使用者模式和特殊權利模式）下允許存取內部累加器。

Intel XScale 核心提供 MAR 及 MRA 兩個指令以執行兩個 ARM 暫存器與 acc0 之間的資料移動。

表3-5　存取內部累加器的格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	0	0	0	1	0	L	RdHi				RdLo				0	0	0	0	0	0	0	0	0	acc		

位元	敘述	備註
31：28	cond – ARM狀態碼	-
20	L-移至/從內部累加器 0= 移至內部累加器（MAR） 1= 從內部累加器移出（MRA）	-
19：16	RdHi – 設定內部累加器的8個高位元（39：32）	在acc的讀取，這8個高位元欄位將被做有正負號的延伸。 在acc的寫入，這個暫存器的8個低位元將寫入acc[39：32]
15：12	RdLo – 設定內部累加器的32個低位元（31：0）	-
7：4	設為0	這個欄位保留未來使用以指定飽和的型態來執行內部累加器的讀取（例如：一個具正負號值被飽和至16位元將在一些篩選演算法中相當有用）
3	設為0	-
2：0	acc – 選擇八個累加器之一	Intel XScale核心僅使用acc0，存取其他的acc將會導致無可預期的影響

備註：MAR 的編碼同 MCRR（至協同處理器 0），MRA 的編碼同 MRRC（至協同處理器 0）。這些指令處理 ARM 暫存器與協同處理器暫存器之間的資料搬移。MCRR 與 MRRC 定義於 ARM 的 DSP 指令集。

表3-6　MAR{<cond>}acc0, RdLo, RdHi

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	0	0	0	1	0	0	RdHi				RdLo				0	0	0	0	0	0	0	0	0	0	0	0

運算：if ConditionPassed（<cond>） then acc0[39：32] = RdHi[7：0] acc0[31：0] = RdLo[31：0] 例外：無 合格狀況碼：無狀況碼旗標會被更新 備註：指令時間將於10.4.4章節的「多重指令時間」中詳述。RdHi和RdLo不可指

定同一個暫存器，否會有不能預期的結果，且RdHi和RdLo若用到R15也會有不可預期的結果。

MAR 指令從暫存器 RdLo 移動值至 40 位元 acc0 累加器之[31：0]位元，且從暫存器 RdHi 移動[7：0]位元的值至 acc0 的[39：32]位元內。

這個指令只有在其前四個位元 cond 符合狀況碼的狀態才會執行。

這個指令在任何處理器模式下都可執行。

表3-7　MRA{<cond>} RdLo, RdHi, acc0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				1	1	0	0	0	1	0	1	RdHi				RdLo				0	0	0	0	0	0	0	0	0	0	0	0

運算：if ConditionPassed（<cond>） then RdHi[31：0] = sign_extend（acc0[39：32]） RdLo[31：0] = acc0[31：0] 例外：無 合格狀況碼：無狀況碼旗標會被更新 備註：指令時間將於10.4.4章節的「多重指令時間」中詳述。RdHi和RdLo不可指定同一個暫存器，否會有不能預期的結果，且RdHi和RdLo若用到R15也會有不可預期的結果。

MRA 指令將 40 位元 acc0 累加器的值移至兩個暫存器內，acc0 的[31：0]位元值移至暫存器 RdLo，acc0 的[39：32]位元值有號擴展至 32 位元後移至暫存器 RdHi。

這個指令只有在其前四個位元 cond 符合狀況碼的狀態才會執行。

這個指令在任何處理器模式下都可執行。

3.2.3.2　新頁面屬性

Intel XScale 核心藉由頁面敘述器中的 C 和 B 位元，加上額外 X 位元的定義來擴充頁面屬性，當 X=1 時，這個位元允許編碼四個以上的屬性。這些新的編碼包含爲了迷你資料快取和寫入配置快取給予配置資料。對於完整的編碼將於章節 3.3.2.2「記憶體屬性」敘述。

Intel XScale 核心保留在 ARM 中當 X=0 時之 C 與 B 位元的定義，這是不同於 strongARM 產品的，對於迷你資料快取的記憶體屬性已被移除取代成逐一回

寫的快取屬性。

當寫入配置是啟動的，則一旦未擊中資料快取（僅可快取資料），儲存至快取的運作將產生；若寫入配置未啟動，只有當載入運作未擊中資料快取（僅可快取資料）時，才會產生儲存至快取的動作。

逐一回寫快取會使得所有儲存運作寫入至記憶體，不管是否為可快取。這個特色對於維持資料快取的一致性相當有用。

Intel XScale 核心也在第一層級敘述器中加入的 P 位元，允許 ASSP 去識別一個新的記憶體屬性，可參考 ASSP 架構部分的詳述關於 Intel XScale 核心如何定義 P 位元。在頁面表格運作時的記憶體存取，是使用控制暫存器（Control Register）的第一個位元（協同處理器 15，暫存器 1，指令碼=1）餵給記憶體屬性的 P 位元。

在轉換表格敘述器中將這些屬性設計程式，這些會列在表 3-8「第一階層敘述器」、表 3-9「粗略的頁面表格之第二階層敘述器」、表 3-10「詳細的頁面表格之第二階層敘述器」。在 Intel XScale 核心有定義兩個第二階層敘述器格式，一個是粗略的頁面表格的使用，另一個是詳細的頁面表格的使用。

表3-8　第一階層敘述器

31–10	9	8	7	6	5	4	3	2	1	0
SBZ									0	0
Coarse page table base address				P	Domain		SBZ		0	1
Section base address	SBZ	TEX	AP	P	Domain	0	C	B	1	0
Fine page table base address			SBZ	P	Domain		SBZ		1	1

表3-9　粗略的頁面表格之第二階層敘述器

31–16	15–12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ												0	0
Large page base address	TEX	AP3	AP2	AP1	AP0			C	B	0	1		
Small page base address		AP3	AP2	AP1	AP0			C	B	1	0		
Extended small page base address		SB	TEX			AP		C	B	1	1		

	Z						

表3-10　詳細的頁面表格之第二階層敘述器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SBZ																														0	0
Large page base address																TEX				AP 3		AP 2		AP 1		AP 0		C		B	0	1
Small page base address																				AP 3		AP 2		AP 1		AP 0		C		B	1	0
Tiny Page Base Address																						TEX				AP		C		B	1	1

TEX（型態擴充）欄位出現在數個敘述器型態中，在 Intel XScale 核心中，僅有此欄位的最低有效位元（LSB）被使用，這就是 X 位元。

所有的小頁面敘述器都沒有 TEX 欄位，在這些敘述器中，TEX 都是零，也就是在 X 位元值為 0 時才運作。

當設定後，X 位元修改 C 和 B 位元。頁面屬性的敘述及它們編碼的方式可查看第 3.3 節「記憶體管理」。

3.2.3.3　協同處理器 15（CP15）的功能新增

在 Intel XScale 核心中要容納功能，在 CP15 和 CP14 增加了暫存器。

當 CP15 更新有效時，有時必須要能夠精確地肯定。舉例來說，當啟動記憶體位址轉換（開啟 MMU），必須確切知道何時 MMU 是很肯定在運作的。為了滿足這個需求，對於 Intel XScale 核心有定義一個特定處理器碼序列，這個序列叫做 CPWAIT 顯示如下。

範例3-1　CPWAIT：標準法是等待CP15的更新

```
;; 當軟體需要確定CP15更新有效時，
;; 以下的巨指令會被用到
;; 它可能僅被使用在特殊模式下，
;; 因為它存取CP15

MACRO CPWAIT
```

```
MRC P15, 0, R0, C2, C0, 0     ; CP15的任意讀取
MOV R0, R0                    ; 等待它
SUB PC, PC, #4                ; 分支到下一個指令

; 在此點，任何先前CP15的寫入必須確保有效
ENDM
```

當設定多重的 CP15 暫存器，系統軟體應選擇等待它們更新的保證，只有在 MCR 指令序列之後發出 CPWAIT 才算完成。

在 CPWAIT 完成時，CPWAIT 序列確認 CP15 邊際效益完成；然而，在 CPWAIT 完成或發表之前，CP15 邊際效益就發生是有可能的，程式設計者可要小心不要影響到程式碼的正確性。

3.2.3.4 事件架構

3.2.3.4.1 例外情況摘要

表格 3-11 顯示 Intel XScale 核心會產生的所有例外及其屬性，隨後章節將對每個例外做詳述。

表3-11.例外情況摘要

例外敘述	例外型態[a]	是否精確?	是否更新FAR?
Reset	Reset	N	N
FIQ	FIQ	N	N
IRQ	IRQ	N	N
External Instruction	Prefetch	Y	N
Instruction MMU	Prefetch	Y	N
Instruction Cache Parity	Prefetch	Y	N
Lock Abort	Data	Y	N
MMU Data	Data	Y	Y
External Data	Data	N	N
Data Cache Parity	Data	N	N
Software Interrupt	Software Interrupt	Y	N
Undefined Instruction	Undefined Instruction	Y	N
Debug Events[b]	varies	varies	N

a.例外型態都有在ARM第2.5章節中詳述

b.參考第九章「軟體偵錯」詳述

3.2.3.4.2 事件優先權

Intel XScale 核心承襲 ARM 架構參考手冊（ARM Architecture Reference Manual）所描述的例外優先權。在偵錯時，處理器會產生額外的例外。更詳細的偵錯例外請看第九章「軟體偵錯」。

表3-12.事件優先權

例外	優先權
Reset	1（Highest）
Data Abort（Precise&Imprecise）	2
FIQ	3
IRQ	4
Prefetch Abort	5
Undefined Instruction, SWI	6（Lowest）

3.2.3.4.3 預先擷取中止

Intel XScale 核心偵測三種預先擷取中止的型態：指令 MMU 中止、再指令存取時的外部中止、以及指令快取同位元中止。這些中止詳述於表格 3-13。

當一個預先擷取中止發生，硬體在錯誤狀態暫存器的延伸狀態欄位回報最高優先權，放在 R14_ABORT（中止模式下的連結暫存器）的值是中止指令的位址+4。

表3-13.Intel XScale核心對於預先擷取中止之錯誤狀態的編碼

優先權	來源	FS[10,3：0][a]	範圍	FAR
最高	指令MMU例外 數個可以產生此編碼的例外 - 轉換錯誤 - 範圍錯誤 - 權限錯誤 此由軟體判別是哪一種錯誤	0b10000	無效的	無效的
	外部指令錯誤例外 當外部記憶體系統回報在指令快取擷取上的錯誤，這個例外會產生	0b10110	無效的	無效的
最低	指令快取同位元錯誤例外	0b11000	無效的	無效的

a.未列在此表格內的其他編碼目前保留

3.2.3.4.4　資料中止

Intel XScale 核心提供兩種資料中止的型態：嚴謹的、不嚴謹的。嚴謹的資料中止是定義為 R14_ABORT 包含導致例外的指令之程式計數（+8）；不嚴謹的資料中止是 R14_ABORT 包含下個指令的程式計數（+4）執行且不包含導致中止的指令位址。換句話說，指令執行會遠在導致資料中止的指令之後。

在 Intel XScale 核心對於嚴謹的資料中止是可回復的，不嚴謹的資料中止則否。

嚴謹資料中止

- 鎖定中止是一種嚴謹資料中止；錯誤狀態暫存器的延伸狀態欄位設定為 0xb10100。當一個直接對 MMU 的鎖定運作或指令快取導致例外、屬於轉換錯誤、存取權限錯誤、或外部排線錯誤都會產生此中止。
  錯誤位址暫存器是未定義，且 R14_ABORT 是中止指令位址+8。

- 一個資料 MMU 中止是嚴謹資料中止。這些是由於排列錯誤、轉換錯誤、範圍錯誤、權限錯誤、或在 MMU 轉換時之外部資料錯誤。這些狀態欄位早在 ARM 定義時已經設定好，如表格 3-14「Intel XScale 核心對於資料中止的錯誤狀態編碼。
  此錯誤位址暫存器設定指令的有效資料位址且 R14_ABORT 是中止指令的位址+8。

表3-14.Intel XScale核心對於資料中止的錯誤狀態編碼

優先權	來源		FS[10,3：0][a]	範圍	FAR
最高	排列		0b000x1	不合法	合法
	轉換時的外部中止	第一層級	0b01100	不合法	合法
		第二層級	0b01110	合法	合法
	轉換	片段頁面	0b00101	不合法	合法
			0b00111	合法	合法
	範圍	片段頁面	0b01001	合法	合法
			0b01011	合法	合法
	權限	片段頁面	0b01101	合法	合法
			0b01111	合法	合法
	鎖定中止		0b10100	不合法	不合法

	當一的MMU鎖定運作（資料或指令TLB）或當一個指令快取鎖定運作，此資料中止會發生			
	不嚴謹的外部資料中止	0b10110	不合法	不合法
最低	資料快取同位元錯誤例外	0b11000	不合法	不合法

a.沒有列在此表格的所有其他編碼都被保留

不嚴謹資料中止

- 資料快取同位元錯誤是屬於不嚴謹的，錯誤狀態暫存器的延伸狀態欄位是設定為 0xb11000。
- 除了那些在資料 MMU 轉換時產生的中止外，所有的外部資料中止都是不嚴謹的。

錯誤位址暫存器對於所有不嚴謹資料中指示位定義的，並且 R14_ABORT 是下個要執行的指令位址+4，這和 ARM 模式與 Thumb 模式是相同的。

雖然 Intel XScale 核心對於嚴謹中止保證基本的回復中止模型，但這在不嚴謹中止下是無法如此做的。如果因為不嚴謹中止而被觸發，資料中止處理器將使用一更新基底暫存器。

不嚴謹資料中止可能產生一種讓中止處理器很難回復的情況，外部資料中止與資料快取同位元錯誤都會導致有問題的資料存入目標暫存器。因為這些錯誤是不嚴謹的，有問題的資料將有可能在資料中止錯誤處理器被觸發前被使用。因為如此，軟體應將不嚴謹資料中止視為不可回復。

注意即使記憶體存取標記為「暫停直到完成為止」（查看 3.2.2.4 章節）也會導致不嚴謹資料中止，對於這些存取方式，此種錯誤較一般狀況還不嚴謹些，這錯誤的通知會在發生此種錯誤的指令之後的三個指令內產生。換句話說，如果一個「暫停直到完成為止」LD（載入）或 ST（儲存）指令觸發不嚴謹錯誤，則程式將在三個指令內收到錯誤通知。

了解以後，在不受影響的寫下存取「暫停直到完成為止」記憶體程式碼是有可能的，簡單的放數個 NOP 指令在此種存取之後即可。如果不嚴謹錯誤產生，這會在數個 NOP 的情況下執行，資料中止處理器將視同一個暫存器和記憶體狀態為嚴謹例外，則將可以做回復。關於此方式的例子可參考範例 3-2

範例3-2　避免潛在不嚴謹中止的程式碼

```
;; 在會產生不嚴謹錯誤下，維持架構狀態的範例程式碼

LD R0, [R1]                    ; R1 指到「暫停直到完成為止」的記憶體區域
NOP
NOP
NOP
; 在此點之後的程式碼保證不會從LD中看到任何中止
```

當然，如果系統設計上已對導致外部中止的事件有做預防，則上述範例的預防方式就不需要了。

多重資料中止

多重資料中止會被硬體偵測到，但僅有最高優先權的會有回報，如果回報資料中止是嚴謹的，軟體會修正導致終止的原因並重新執行中止的指令。如果較低優先權的中止仍然存在，這才將會回報。軟體會個別的執行每個中止直到指令成功地執行。

如果回報的資料中指示不嚴謹的，軟體必須確認 SPSR 是否之前的指令是在中止模式下執行的，如果是這樣，回到目前程序的連結已經遺失，並且資料中止是不可回復的。

3.2.3.4.5 預先載入指令的事件

一個 PLD 指令不會因為下列原因而導致資料 MMU 出差錯：

- 範圍錯誤
- 權限錯誤
- 轉換錯誤

如果 PLD 的執行會導致上述其中一種錯誤，則 PLD 會導致無效。

這項特性會讓軟體冒險的使用 PLD。舉例來說，範例 3-3 將 PLD 指令放在迴圈的初期，此 PLD 重複地在下個迴圈時擷取資料。在這個例子中，串列在一個無效的指標節點結束掉，當執行到此串列結尾，PLD 在位址 0x0 不會導致錯誤，確切的說，它將被忽略且迴圈會正常的結束。

範例3-3. Speculatively issuing PLD

```
;; R0指掉鏈結串列的一個節點。一個節點有下列的內容安排：
```

```
;; 位址偏移內容
;;---------------------------------
;; 0            data
;; 4            pointer to next node
;; 這些程式碼是計算所有在串列上的節點之總合，並將結果存放於R9
;;
MOV R9, #0          ; 清除累加器
sumList：
LDR R1, [R0, #4]    ; R1將指標指到下一個節點
LDR R3, [R0]         ; R3從目前的節點取得資料
PLD [R1]            ; 冒險地從下一個節點開始載入
ADD R9, R9, R3      ; 加到累加器內
MOVS R0, R1         ; 前進到下一個節點。是否已在串列最後？
BNE sumList         ; 如果沒有，迴圈繼續
```

3.2.3.4.6 偵錯事件

偵錯事件會在 9.5 章節「偵錯例外」中說明。

3.3 記憶體管理（Memory Management）

本章節將介紹在 Intel XScale 核心中執行的記憶體管理。

3.3.1 概要

Intel XScale 核心執行記憶體管理單元（MMU）架構在 ARM 架構參考手冊中有詳細說明。為了加速虛擬至實體位址的轉換，Intel XScale 核心使用指令的轉換側查緩衝（Translation Look-aside Buffer, TLB ） 及資料 TLB 來快取最後一個轉換。每一個 TLB 可容納 32 個項目並且是相關聯的，不只 TLB 包含轉換的位址，用於記憶體參考的存取權限也是。

如果一個指令或資料的 TLB 發生未擊中，將會觸發硬體的活動轉換表格（ translation-table-walking ） 機制來轉換虛擬位址為實體位址。一旦轉換，實體位址與頁面或片段的存取權限及屬性都會放入 TLB，這些轉換會被鎖定在任何一個 TLB 以保證關鍵性程序的效能。

Intel XScale 核心允許系統軟體將不同的屬性與記憶體區相結合：

- 可快取
- 可載入緩衝
- 線配置策略
- 寫入策略
- 輸入/輸出
- 迷你資料快取
- 接合
- ASSP 可定義屬性─ P 位元 （ 可參考 Intel XScale 核心執行 ASSP 架構敘述的部分以獲得更多資訊 ）

3.2.2 章節的「記憶體屬性」介紹頁面屬性，及在 2.3.2 章節的「新頁面屬性」可查詢到在 MMU 敘述器中對應的屬性

注意：PID 暫存器會將虛擬位址與被存取的 TLB 重新對應，可查看 7.2.13 章節「程序 ID」了解有關於 PID 暫存器的細節。

3.3.2 架構模型

3.3.2.1 版本 4 與版本 5

ARM MMU 版本 5 的架構介紹關於 1K 位元組大小的微小頁面，在第一階層敘述器的保留欄位是使用為詳細的頁面表格基底位址，有關於確切的位元欄位、和第一及第二階層敘述器的格式可以查看 2.3.2 章節「新頁面屬性」。

3.3.2.2 記憶體屬性

在記憶體管理頁面表格中設定屬性及其關聯的特別的記憶體區域，並且控制對於指令快取、資料快取、迷你資料快取、及寫入緩衝取的存取方式。當 MMU 不被啟動時，這些屬性是沒有作用的。

為了達到與舊系統軟體相容，新的 Intel XScale 核心屬性利用敘述器過去保留的編碼空間。

3.3.2.2.1 頁面（P）屬性位元

P 位元允許 ASSP 將它自己的頁面屬性設定給記憶體區域，可參考 Intel XScale 核心實行 ASSP 架構敘述的部分以找出它是如何被定義的。

3.3.2.2.2 可快取（C）、可載入緩衝（B）及延伸（X）位元

3.3.2.2.3 指令快取

當在敘述器中檢查這些位元時，指令快取僅使用 C 位元。如果 C 位元是被清除，指令快取將從記憶體擷取來的程式碼視為不可快取的，並且不會將它存入快取中；如果 C 位元被設定，從相關的記憶體區域擷取來的資料將被快取起來。

3.3.2.2.4　資料快取及寫入緩衝區

所有的敘述器位元影響資料快取及寫入緩衝區的行為。

如果 X 位元對於敘述器是 0 的話，則 C 與 B 位元會如同 ARM 架構所規定的運作，行為描述如表 3-15。

如果 X 位元對於敘述器是 1 的話，則 C 與 B 位元的意義代表是延伸，如同表 3-16 所述。

表3-15　當 X = 0 時，資料快取與緩衝區行為

C B	可否快取?	可否載入緩衝區?	寫入策略	線配置策略	備註
0 0	N	N	-	-	暫停直到完成為止[a]
0 1	N	Y	-	-	
1 0	Y	Y	逐一回寫	讀取配置	
1 1	Y	Y	批次回寫	讀取配置	

a. 一般來說，處理器存取一個資料之後，若是沒有再遇到與此存取相關的指令，則仍舊會持續執行下去。如果是這樣設定，處理迄會暫停執行直到資料存取完成為止，這樣資料存取指令完成的執行時間可以對軟體保證資料存取是有效的。存取資料所產生的外部資料中止是不嚴謹的 （ 查看 2.3.4.4 章節對於此不嚴謹中止之保護程式撰寫方法 ）。

表3-16　當 X = 1 時，資料快取與緩衝區行為

C B	可否快取?	可否載入緩衝區?	寫入策略	線配置策略	備註
0 0	-	-	-	-	不可預料的 – 勿使用
0 1	N	Y	-	-	寫入不會與緩衝區接合[a]
1 0	（ 迷你資料快取 ）	-	-	-	快取策略由輔助控制暫

					存器的 MD 欄位來決定[b]
1 1	Y	Y	批次回寫	讀 / 寫配置	

a. 一般來說，可寫入緩衝之寫入可與先前在同樣位址區域寫入緩衝區資料相接合。

b. 有關此暫存器的詳細說明，可查看 7.2.2 章節

3.3.2.2.5 資料快取及寫入緩衝區行為細節

如果 MMU 沒有啟動，則所有資料存取都是不可快取及不可寫入緩衝的，這與 MMU 在啟動狀態，但資料存取使用敘述器其 X、C、B 位元都設為 0 時的運作是相同的。

當處理器應放新資料至資料快取時，X、C、B 位元就被確立了。快取將資料以一行行 （ 也稱作區塊 ） 的方式放入快取，因此，決定如何將新資料放入快取就稱為「線配置策略」。

如果線配置策略是讀取配置，所有未擊中快取的載入運作會從外部記憶體要求一個 32 位元組的快取線，並且配置給資料快取或迷你資料快取 （ 假定快取是啟動的 ）。未擊中快取的儲存運作是不會產生線配置的。

如果讀 / 寫配置在運行中，且快取是啟動的，則未擊中快取的載入與儲存運作會從外部記憶體要求一個 32 位元組的快取線。

X、C、B 確立的其他策略是寫入策略。一個逐一回寫策略指示資料快取藉由儲存至外部記憶體及快取去維持外部記憶體的一致性。一個批次回寫策略僅在線在快取內被清空時或線需要被另一新線所取代時才會更新外部記憶體。一般來說，批次回寫提供較高的效能，因為它對外部記憶體產生較少的資料封包。

關於快取策略更多的細節可查看 6.2.3 章節的「快取策略」。

3.3.2.2.6 記憶體操作佈置

一個防護 （ fence ） 記憶體運作 （ memop ） 是保證所有在防護之前的 memop 發佈將在防護之後的任何 memop 發佈之前執行。因此軟體應發佈防護來利用記憶體存取的部分指令。

表 3-17 顯示 memop 運作如防護的情況。

任何交換 （ SWP 或 SWPB ） 到一個在載入或儲存時開啟防護的頁面是一種防護。

表3-17 利用防護的記憶體操作

操作	X	C	B
載入	-	0	-
儲存	1	0	1
載入或儲存	0	0	0

3.3.2.3 例外

MMU 會對於指令存取產生預先擷取中止及對於資料記憶體存取產生資料中止。這些例外的型態與優先權詳述於 2.3.4 章節「事件架構」。

設定控制暫存器的位元 1 （ CP15, 暫存器 1 ） 可啟動資料位址調準檢查。即使 MMU 不啟動，調準錯誤仍會回報。當 MMU 不啟動時，所有其他的 MMU 例外是不啟動的。

3.3.3 MMU、指令快取及資料快取之相互影響

MMU、指令快取、資料 / 迷你資料快取可獨立設定啟動與不啟動。指令快取可與 MMU 的啟動或不啟動而啟動。然而，資料快取僅可在 MMU 啟動時才能啟動。因此，MMU 與資料 / 迷你資料快取啟動的四種組合僅有三種是合法的，不合法的組合將導致未定義的結果。

表3-18 MMU 與資料/迷你資料快取的合法組合

MMU	資料/迷你資料快取
Off	Off
On	Off
On	On

3.3.4 控制

3.3.4.1 無效的（Flush）操作

全部的指令和資料 TLB 可以因一個指令而同時成為無效或個別的成為無效。一個個別在資料或指令 TLB 內的項目也可以是無效的。可於表格 7-13「TLB 函數」查看 Intel XScale 核心有支援的指令列表。

全域的使一個 TLB 無效並不會影響鎖定 TLB 的項目，然而，無效項目的運作可以讓個別鎖定的項目無效，在此狀況下，在 TLB 中仍舊保持鎖定狀態，但是在位址轉換下決不會「擊中」它。實際上，這是在 TLB 上的一個漏洞，此種狀況只要解開 TLB 鎖定即可矯正。

3.3.4.2 啟動/不啟動

設定協同處理器 15，暫存器 1 的位元 1（ 控制暫存器 ）可啟動 MMU。

當 MMU 不啟動時，對於指令快取的存取預設是為可快取的，而對於資料記憶體的所有存取預設是為不可快取的。

啟動 MMU 的建議程式撰寫順序請參考範例 3-4

範例3-4 啟動 MMU

```
; 這段程式提供軟體使用可預測的方式來啟動 MMU。在 CPWAIT 之後
; MMU 則被確認啟動.
; 了解有時 MMU 在 MCR 之後、在 CPWAIT 之後執行的指令之前被啟動
; 程式撰寫上須注意 ： 從 MMU 被啟動之後，在這段程式碼需要
; 虛擬位址至實體位址的一對一對應
; 這允許在 MCR 之後的指令，不管 MMU 的狀態如何，都能正確地執行

MRC P15,0,R0,C1,C0,0; 讀 CP15, 暫存器 1
ORR R0, R0, #0x1; 打開 MMU
MCR P15,0,R0,C1,C0,0; 寫入 CP15, 暫存器 1

; 有關 CPWAIT 的詳述，請查看 2.3.3 章節的「協同處理器 15 的功能新增」
CPWAIT
; MMU 從這點起確認啟動；下個指令或資料位址將被轉換
```

3.3.4.3 鎖定項目

在指令及資料 TLB 可以鎖定個別的項目，可查看表格 7-14「快取鎖定函數」獲得精確的指令。如果鎖定運作發現虛擬位址轉換已經常駐在 TLB 之中，則結果是不可預期的。在所定指令確保適當地執行之前，項目指令是無效的。所有項目無效，軟體仍舊可以完成，可查看範例 3-5。

鎖定項目至指令 TLB 或資料 TLB 可降低再硬體至快取其他虛擬至實體位址的可利用的項目數。

一段副程式有關於鎖定至指令 TLB 顯示於範例 3-5。

如果在指令或資料 TLB 鎖定運作之中產生 MMU 中止，則錯誤狀態暫存器更新來指令鎖定中止 （ 查看 2.3.4.4「資料中止」 ），及會有例外回報如同資料中止。

範例3-5 鎖定項目至指令 TLB

```
; R1、R2、及 R3 包含要轉換的虛擬位址及鎖定至指令 TLB

; 在如下的指令，其 R0 的值是被忽略的
; 硬體保證在程式的順序下， CP15 的存取會發生

MCR P15,0,R0,C8,C5,0; 使整個指令 TLB 無效

MCR P15,0,R1,C10,C4,0; 轉換虛擬位址（R1） 並且鎖定至指令 TLB
MCR P15,0,R2,C10,C4,0; 轉換虛擬位址（R2） 並且鎖定至指令 TLB
MCR P15,0,R3,C10,C4,0; 轉換虛擬位址（R3） 並且鎖定至指令 TLB

CPWAIT

; 在此點 MMU 保證被更新；下個指令將查看已鎖定的指令 TLB 項目
```

注意：如果在此段程式之中段允許例外發生，TLB 將會結束快取關於被鎖定的轉換快取。舉例來說，如果 R1 是一個中斷服務程式虛擬位址，並且在 TLB 為無效之後，中斷會即刻發生，當中斷服務程式回傳此程式碼序列後，鎖定的運作將被忽略。在此例中，軟體應當不啟動中斷 （ FIQ 或 IRQ ）。

在一般的規則當中，在所有其它的例外型態軟體應當避免鎖定。

鎖定項目至資料 TLB 的適當程序如範例 3-6。

範例3-6 鎖定項目至資料 TLB

```
; R1、R2 包含要轉換的虛擬位址、及鎖定至資料 TLB

MCR P15,0,R1,C8,C6,1 ; 在 R1 的虛擬位址詳述資料 TLB 項目的無效
```

```
MCR P15,0,R1,C10,C8,0 ; 轉換虛擬位址 （R1） 並且鎖定至資料 TLB

; 重複在 R2虛擬位址的序列
MCR P15,0,R2,C8,C6,1 ; 在 R2 的虛擬位址詳述資料 TLB 項目的無效
MCR P15,0,R2,C10,C8,0 ; 轉換虛擬位址 （R2） 並且鎖定至資料 TLB

CPWAIT ; 等待鎖定完成

; 在此點 MMU 保證被更新；下個指令將查看已鎖定的資料 TLB 項目
```

3.3.4.4 循環取代演算法

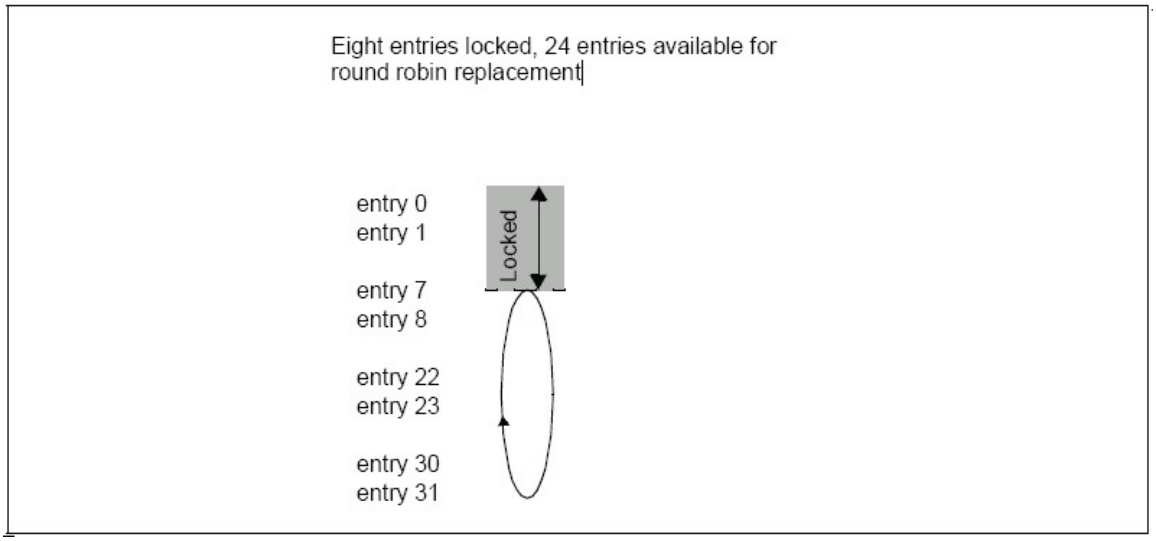
對於 TLB 的線取代演算法是循環方式，有一個循環指標負責追蹤下一個要取代的項目。下一個要取代的項目是在被寫入的最後一個項目其序列上之後一個項目。舉例來說，如果最後一個虛擬至實體位址轉換是寫入至項目 5，則下一個要被取代的項目是項目 6。

在重置之後，循環指標就被設定指向項目 31，一旦一個轉換寫入了項目 31，則循環指標會取得下一個可用的項目，如果沒有項目被鎖定的話，那就是最前頭的項目 0。隨後的轉換會移動循環指標至下一個序列上的後一個，直到到達項目 31 為止，又會再轉回至項目 0。

鎖定指標會使用在鎖定項目至 TLB，並且在重置時設置至項目 0。一個 TLB 鎖定運作放置指定的轉換至鎖定指標所指定的項目中，移動鎖定指標至下一個序列項目，並且重置循環指標至項目 31。鎖定項目至任一個 TLB 將有效地減少可用的更新項目。舉例來說，如果前三個項目被鎖定，則從項目 31 轉過來的循環指標將只到項目 3。

任何一個 TLB 只有項目 0 至項目 30 可以被鎖定，項目 31 是不能被鎖定的。如果鎖定指標在項目 31，鎖定操作將轉換並更新 TLB，但會忽略鎖定。在這個案例下，循環指標仍舊會停留在項目 31。

表3-19 在 TLB 中鎖定項目的範例



3.4 指令快取 （ Instruction Cache ）

概要

操作

當指令快取啟動時的運作
當指令快取不啟動時的運作
擷取策略
循環取代演算法
同位元保護
指令擷取延遲
指令快取一致性

指令快取控制

在重置 （ RESET ） 的指令快取狀態
啟動 / 不啟動
使指令快取無效
在指令快取內鎖定指令
在指令快取內解除鎖定指令

問題：

1. 請簡述 Intel XScale 核心的功能區塊。

2. 請簡述「記憶體管理」之功能。
3. 請描述指令快取、資料快取的架構
4. 試述 Intel XScale 核心相較於 ARM5，做了哪些的擴充？
5. 試述 Intel XScale 核心之事件的優先權順序。
6. 試述分支目的緩衝區的用途。
7. 何謂 Thumb 指令集？
8. 何謂多重資料中止？