簡介電通所爪哇處理機的設計
Introducing to CCL Java Processor

馬瑞良
工研院電通所
新竹縣竹東鎮中興路四段 195-11 號 51 館 X200
840498@itri.org.tw

摘要

Java 在網際網路的熱潮中切入，成為網路上目前最受重視的語言。Java 處理機可以直接執行 Java bytecode 和 extended bytecode，在 Java 程式的執行速度上比起其它以解譯方式執行的處理機，有更佳的執行效能。 CC Java CPU 是一個支援 Java 物件導向處理及高效能堆疊執行的 Java 處理機，它具有管線式的執行架構。CC Java CPU I 是 1998 年 7 月完成的整數處理機核心，其執行效能約為同頻率 StrongAR 的 3 至 9 倍，CCL Jav CPU II 則是 1999 年 6 月完成的一個 full-set 的 Java 處理機。本文將說明它的管線式執行架構及其中各模組的特性及相互關係，其中智慧型堆疊管理和指令折疊是 Java 處理機的主要架構特色。

關鍵詞：堆疊處理機、堆疊快取、指令折疊

1 前言

網際網路及其相關產業近年來大幅成長，根據 IDC 估計，西元 2000 年時全球網際網路用戶的數量約為二億三千萬戶。迅速普及的網際網路風潮勢必會改變目前的電腦市場發展，電腦軟硬體和應用都會產生許多變化。Java 相關技術因其具跨平台、動態連接，及內建之安全性考量等特色，非常符合網際網路分散式運算特性[1]。電通所 Java 處理機是一個支援 Java 物件導向處理及高效能堆疊執行的 Java 處理機。它具有管線式的執行架構，能夠直接執行 Java bytecode 指令，並提供所需的系統功能及中斷處理能力。本文說明 CCL Java CPU 的管線式執行架構及設計方法，其中智慧型堆疊管理和指令折疊是 Java 處理機的主要架構特色。

本文之組織架構說明如下：第二節簡介 CCL Java CP 發展沿革；第三節介紹 CCL Java CPU I 之主要架構與特色；第四節就處理機內部之重要單元做進一步之說明；第五節總結計畫之執行成果與未來方向。

2 CCL Java CPU 發展沿革

Java語言所撰寫的程式是透過一個明確定義的虛擬系統架構來執行的，這個系統稱為Java虛擬機器(Java Virtual Machine， JVM)。Java虛擬機器包含一套230個指令的Java位元指令集(Java bytecode) 和一些系統暫存器和記憶體資源的定義[2]。執行Java的方法有三(亦即實現Java虛擬機器的方法有三)：第一種方法是以解譯(interpreting的方法來執行bytecode，即Java虛擬機器每次處理及解譯一個bytecode指令，這是目前最常用的方法。 它的優點是構建容易，缺點是解譯執行效能不佳。第二種方法是利用一個即時編譯器(just-in-time compiler)，每次執行時動態的將Java bytecod 編譯成目的處理機的目的碼(native code)。它的優點是執行效能約可比前者提昇五到十倍，但編譯器需要額外用到約2百萬位元組的記憶體空間。最後一種方法是以bytecode為目的碼，設計一個Java處理機，用它來直接執行bytecode 。這種方法的好處為執行效能最佳，也不需要額外的記憶體，但是設計一顆Java處理機有一定的複雜度，而且需要提供後續的系統軟體和發展環境等支援。

由於一般Java的執行是利用解譯的方式來解譯bytecode，所以大家普遍覺得目前Java的執行效能較差。SME(Sun Micro Electronics，Java的起源地)為解決此一問題，提出一套硬體解譯的方案，即設計一個能直接執行bytecode的硬體核心（CPU core）。目前SME提出的Java硬體解譯的產品主要包含PicoJava和MicroJava兩個系列。PicoJava為其第一個產品，是一個處理機核心(core)，SME將這個處理機核心免費授權給各大小廠商和學校

設計各種的應用產品，以擴大Java產品的市場及應用領域，目前有100MHz的PicoJavaII core。另一方面SME本身亦同時開發整合型的Java處理機以滿足個人電腦市場和嵌入式控制器(embedded controller)的需求，其第一個產品是MicroJava 701處理機。由於Java CPU主要的市場是在嵌入式控制器的應用，所以SME決定專注在Java CPU core的設計，最新的消息指出SME將專注於發展PicoJava核心而放棄stand alone的CPU設計。

SME預測西元兩千年時全球的嵌入式控制器的總使用量達到七十億顆，而消費性電子產品如行動電話的銷售量也會突破一億。Java 處理機可以直接執行Java bytecode，在Java 程式的執行速度上比起一般以解譯方式執行的處理機，有更佳的執行效能。所以在網際網路等大量需求使用Java的環境中，它便具有生存之利基。Java 處理機除了發展成網路電腦和家用電腦之核心處理機外，還可以推廣至微控制器及消費性電子產品等相關產品的領域中使用，如PDA、多功能行動電話等等。

SME 在1996年左右開始開發及授權其Java處理機，當時其PicoJava的授權費用約需數百萬美金，其應用產品銷售時還要付一定比例的權利金。由於金額過於龐大、市場需求不明確等等因素，工研院電通所轉而自行開發Java處理機。工研院電通所從民國86年中起執行經濟部技術處「Java處理機核心技術」在87年中開發出第一代電通所Java 處理機(簡稱CCL Java CPU I)核心，並將在第二年擴充成為完整的 Java 處理機—CCL Java CPU II，構成Java網路資訊系統的核心機構。隨著未來大量的網際網路連線需求，由Java 處理機、Java 作業系統、和各種Java應用軟體組成的Java網路資訊系統，將會在未來的網際網路市場中佔有一席之地。由於Java語言在網際網路的普遍應用及大量Java軟體的出現，比起其他處理機設計的網路資訊裝置，Java 處理機可以更直接快速的執行Java軟體，擁有更佳的效能及價格優勢。最後我們可以再結合Sun的Jini或Microsoft的uPnP技術，把Java網路資訊整合系統推入家庭，構成所謂的Java Home和Java Town，讓Java家庭網路資訊整合系統成為家中兼具娛樂，通訊的系統中心。

3 CCL Java CPU II之主要架構與特色

目前我們完成了CCL Java CPU I 的設計，其架構設計主要承襲了CCL Java CPU I 之管線式設計。除此之外，為了開發更高的執行效能(Performance)與系統完整性，我們在這一代的Java CPU加入了動態分支指令預測(Dynamic Branch Prediction 機構、硬體除錯指令及機制、75個extended bytecod 指令、及微指令循序處理器(Sequencer)，這些機構我們將在4.6和4.7做介紹。配合原本的智慧型堆疊管理和指令折疊架 ，便構成了CCL Java CPU II。

CCL Java CPU I 共分六個管線階段(Pipeline Stage)，分別為指令擷取(Instruction Fetching)、解碼(Instruction Decoding)、堆疊讀取(Stack Cache Reading)、執行(Instruction Execution)、記憶體存取(Memory Reading)、和資料寫回(Write Back)。和傳統的5-stage 處理機最大的不同是Java處理機由於有一個複雜的智慧型堆疊管理機制，所以多了一個堆疊讀取的管線階段，它將在4.2中討論。

在指令擷取階段時，由PC(Program Counter)提供下一批指令的起始位址，接著透過匯流排介面單元 將指令從外部記憶體讀入，存入指令緩衝區。當遇到須改變程式執行流程時， 如出現分支指令、中斷等時，PC之值會重新被控制與中斷處理單元設定，指令緩衝區內容同時也要被清除，再重新進行指令擷取。接著執行指令解碼階段，指令緩衝區內之有效指令經指令折疊檢測(instruction folding check)，檢查指令是否可執行指令折疊，隨即被解碼成相關控制訊號，送往堆疊讀取階段處理。若此指令是跳躍指令，則會根據之前跳躍指令的執行歷史決定是否跳躍。若此指令是需要多個微指令(microinstruction 來執行，則會啟動微指令循序處理器(Sequencer)機構，發出適當的控制訊號並至微指令ROM(microinstruction ROM)擷取微指令，送至下一個管線級執行。

在堆疊讀取階段時，運算元位址轉換器(renamer)先將運算元的邏輯位址轉換為實際的位置，再從快取堆疊內讀出其值。另外資料危險(data hazard) 及資料前送(data forwarding)也在此偵測及處理，並產生適當控制訊號。

在指令執行階段時，依據上一級送下來的控制欄位(control field)來控制算術和邏輯單元(ALU)執行，運算之結果與執行時的例外中斷(exceptions)狀況在此載入此級的控制欄位。如執行的是分支指令，其目的位址及是否要執行分支跳躍的動作，也在此載入相關的暫存器。記憶體存取階段依據執行單元送來的控制欄位值來控制是否向匯流排介面單元 發出記憶體存、取要求，而所讀出之值則被存於資料暫存器。各種中斷及分支控制指令也在此級由控制與中斷處理單元處理，判斷之前的跳躍預測是否正確。寫回資料階段(write back stage)只是單純的將資料寫回快取堆疊，所以合併在堆疊暫存器單元中設計。

CCL Java CPU　主要架構特色和功能包括：

(1) 六階的管線式架構設計—指令擷取、解碼、堆疊資料取得、執行、記憶體存取、資料寫回。

*(2) 智慧型堆疊管理架構，具自動化堆疊資料spill/fill功能，使指令執行更流暢。

(3) 提供指令折疊(instruction folding)功能，發掘指令執行平行度，提高單位時間內所完成之指令數。

(4) 分離式(decoupled)之指令擷取與解碼，提高指令擷取頻寬。

(5) 1、2、4位元組之對齊(aligned)與位元未對齊(unaligned)記憶體存取機制，以便利各類型資料存取之需求。

(6) 資料危險(data hazard)之偵測與處理，由硬體直接處理，刪除了編譯器必須排序指令之限制。

(7) 中斷(interrupt)和例外(exception 處理機構及硬體除錯機制。

(8) 動態分支指令預測機構。

(9) 執行微指令之循序處理器。

* 本部份研究正申請美國和中華民國的專利中

4　各單元細部說明

CCL Java CPU I 架構設計由七個模組組成：指令擷取單元(IFU: Instruction Fetch Unit)， 解碼單元(DIU: Decoding Instruction Unit)， 堆疊暫存器單元(SRU: Stack Register Unit)， 執行單元 (EXU: Execution Unit) ，記憶體存取單元 (MAU: Memory Access Unit)， 控制與中斷處理單元 (PCU: Pipeline Control Unit)， 匯流排介面單元 (BMU: Bus Management Unit)。圖1. 為其連接示意圖，以下就其各個單元做進一步之說明。

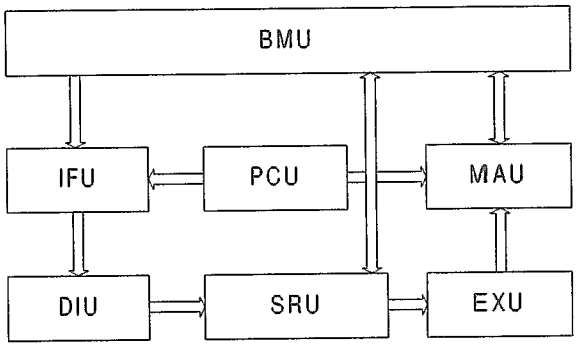


圖 1.　CCL Java Ch i 的架構方塊圖

4.1 指令擷取與解碼

指令擷取單元(Instruction Fetch Unit， IFU)和解碼單元(Decode Instruction Unit， DIU)，主要負責自指令記憶體中根據PC指定的位置讀取指令，再將指令解碼以產生適當的訊號，供管線中以下各級使用。指令擷取單元自指令記憶體一次讀取4位元組的資料，存於指令緩衝器中，再由控制邏輯記錄指令長度，和載入新的目標位置PC，以調整指令擷取及後續的執行順序。

本指令擷取單元的一個特色是分離式(Decoupled)的架構，亦即將有效指令送到解碼單元進行解碼的動作和從指令記憶體讀取指令的動作可各自獨立進行，只要仍有有效指令即可送至解碼單元；反之，若管線必須停止，但指令緩衝器中仍有空位，則指令讀取可繼續進行至指令緩衝器裝滿為止。

解碼單元除了根據指令碼(Opcode)產生相對應的訊號存於管線暫存器中，還可根據指令擷取單元傳過來的指令及其排列順序，進行*指令折疊*[3]。Java虛擬機器是一個堆疊機器，堆疊機器的一大缺點是當所需運算元不是剛好在堆疊頂端時，系統便必須插入一個移動(move)指令將資料由區域變數(local variable)儲存區中移往堆疊頂端，使得系統效能大幅下降。指令折疊便是用來降低此一額外負擔的。當解碼器發現例如有三個指令分別是：

(1) 取出區域變數資料a放入堆疊頂端

(2) 取出區域變數資料b放入堆疊頂端

(3) 由堆疊中取出a和b兩筆堆疊資料做運算時，解碼器便可以把他們轉譯成一個新的指令-- 從區域變數的暫存器中中直接把資料送至運算單元做運算，利用暫存器不需像堆疊循序存取的特性，將原本單純做移動動作的指令去除掉。指令折疊的效用，對某些程式來說，可將堆疊的運算自43%降至29% [4]，不僅可有效地減少指令執行的數量，更能進而提升系統的效能。

除此之外，解碼單元還能處理需要多個週期才能完成的指令，這個機制稱為微指令循序處理器 (Microcode Sequencer)。微指令產生器針對需要多個週期才能完成的指令(如Swap)產生一連串的控制訊號。其主要的功能在於將多個週期指令轉譯成多個微指令(microinstructions)傳送到下一個管線級。微指令產生器主要由微指令ROM(microinstruction ROM)、微指令位址(uPC)選擇電路、及微指令產生器控制電路組成。

微指令ROM依照uPC順序儲存微指令，微指令循序處理器會依照uPC值至微指令ROM擷取相對應的微指令控制訊號。uPC選擇電路根據系統狀態、測試條件，決定下一個微指令的uPC。多週期指令控制電路負責檢測多週期指令(由IFU傳下的資訊得知)，當目前在解碼單元的指令為多週期指令時，啟動微指令循序處理器，同時停住指令擷取單元，將解碼控制權交由微指令循序處理器執行。

4.2 智慧型堆疊管理

電通所 Java 處理機執行的Java虛擬機器是以堆疊機器為基礎的。堆疊機器是假設系統的執行過程中，資料利用一個"先入後出"(first-in-last-out) 的資料堆疊來做儲存的機構，指令執行時系統會自動將堆疊最前端的運算元取出作相對應的運算，並將運算結果放回堆疊去。先前之設計，此一資料堆疊多半是以記憶體堆疊的方式來設計，亦即在記憶體中找一段連續的空間來存放此一堆疊。由於記憶體存取的速度及頻寬有限，一個堆疊機器的系統效能便大受影響。為了解決此一問題，在系統中可以加入一個新的架構，它是由一塊大的暫存器檔(register file)所組成的，稱為快取堆疊(stack cache)。

快取堆疊是堆疊暫存器單元的核心。把原本記憶體堆疊的最頂端放在快取堆疊內，由於資料堆疊的存取是以它最前端的幾個資料為主，所以加入快取堆疊的第一個好處是-- 資料堆疊的存取動作由記憶體存取改變成為暫存器存取，可以提昇系統執行的效能。Java虛擬機器本身也是一個物件導向的系統，每個執行的物件 (object) 在系統中會變成一個動態產生的資 料單位稱為frame，frame執行時除了用到資料堆疊外，還有許多其它動態產生的執行資訊，快取堆疊一併把其它相關的frame執行資訊也放入其中，這些資訊在執行的過程中也常被存取，將他們和資料堆疊一同放在快取堆疊中能大量降低記憶體的存取，提昇整體的系統效能。

由於frame的執行資訊和資料堆疊都放在快取堆疊中，所以又能夠配合解碼單元得到第二種好處 --指令折疊。

指令折疊已在解碼單元中做過說明，此處不再解釋。需注意的是：由於快取堆疊儲存了堆疊和其它frame資訊，我們才可以利用解碼單元來完成指令折疊的功能，相關的詳細Java CPU堆疊管理細節請參考 "Java硬體加速器中快取堆疊之設計與考慮"[3]。

快取堆疊存放執行時動態產生的frames，隨著程式不斷的執行，有限大小的快取堆疊 中存放的frames可能產生堆疊過溢或不足的情形。在一般的系統中，當發生此一情形時，系統便停止執行，而由堆疊過溢或不足的處理機構將一個或多個frames 存放(spill)到記憶體中或從記憶體中擷取(fill)進來，spill/fill的過程可視為快取堆疊的額外負擔。為了降低系統的額外負擔，可以使用背景(background) 執行 spill/fil 配合資料高水位(high water mark)/ 低水位(low water mark)的設計。首先，系統必須先設定快取堆疊的高水位和低水位，當快取堆疊的使用量超過一定數目的暫存器時(這個數目稱為高水位)，就會啟動一個背景執行的spill系統，本系統會將目前快取堆疊中最舊的資料以背景執行處理的方式丟到記憶體堆疊中去(利用另一個獨立的輸出入管道，將資料送出，不影響原本指令的執行)，直到快取堆疊的有效資料量小於高水位。同理，當快取堆疊的的使用量小於低水位時，系統會自動的將記憶體堆疊的資料以背景執行的方式擷取到快取堆疊中，直到快取堆疊的有效資料量大於低水

位。以上快取堆疊相關的設計和管理方法便是Java處理機和一般處理機最主要的不同點。

此外堆疊暫存器單元還負責做資料前送(forwarding)和資料危險(hazard) 的檢查。資料前送的檢查是在管線執行中檢查執行單元所要的運算元，其最新的數值是否尚未寫回快取堆疊，檢查結果將影響執行單元取得運算元的資料路徑，決定是否做資料前送。資料危險偵查則是用來檢查記憶體讀取指令(load)的下一個指令是否會使用到此讀取值。若是，由於管線架構的限制，控制與中斷處理單元會在此讀取指令後動態插入一個NOP(n operation)指令以防止資料危險。

4.3 指令執行單元

指令執行單元(Execution Unit，EXU)主要是負責執行各種算術及邏輯的運算、旗標的設定、判別條件式跳躍(Branch)之條件、跳躍位址(Branch Address)的計算及當算術指令執行時發生不合理之情形產生中斷(除零(divide by zero) 、無效指標(null point) 、超過陣列邊緣(array out of bound))。

指令執行單元主要包含算術邏輯單元(ALU)、位址加法器、跳躍條件判別電路(Branch Decision Circuit)、旗標暫存器(Flag Registers)。算術邏輯單元包含了加減法器、乘法器、除法器、移位器、邏輯運算單元以及提供算術邏輯單元內部控制訊號的解碼器。由於ALU中有許多的控制訊號，若是由外部的解碼單元來控制，則會耗費許多的匯流排面積，也使得解碼單元的負擔更重，為使解碼單元得以簡化，所以在ALU中設計一個解碼器，專責ALU中各個控制訊號的產生，如此解碼單元只需利用少數的匯流排來通知ALU該做的動作。在算術邏輯單元中，除了除法器為需要多個週期(multi-cycle) 執行的單元，其他皆為只需一週期時間的單元。

由於Java的乘法指令為32位元乘以32位元，最後只取結果的低32位元，我們選擇了Booth's Algorith 及Wallace tree的架構，使得運算過程電位的變換次數減少，乘法器的閘層(gate level)降低，以減少乘法器執行所需的執行時間及消耗的功率。在除法器方面，由於32位元的除法器需要多個週期來完成，我們將他分為三個階段，第一階段為初始化，第二階段為運算，第三階段為調整。初始化是將除數及被除數正規化(normalization)。在運算階段是利用SRT model及Sequential Parallel原理，每次只做8位元減法，產生8個位元的商數及餘數，利用暫存器儲存部份結果，全部除法需花費1到4個週期來完成所有位元的運算。調整(correct)階段是將商數及餘數還原回正確的位置。

在JavaCPU II除法器的設計中，我們用8位元的減法取代所有的32位元減法，使得電路面積減小且運算時間減少。

4.4 記憶體存取與匯流排介面

記憶體存取單元的工作主要是根據執行單元的要求，向匯流排介面單元提出使用匯流排的請求，作記憶體的讀寫動作。另外也提供資料路徑，接受控制與中斷處理單元的控制作中斷的處理。

CCL Java CPU I 對外界匯流排的寬度是 32 位元，所以 讀寫指令對記憶體作存取一次最多可存取 4 個位元組資料。不論從記憶體讀取 1 或 2 或 4 個位元組的資料，讀取時都是一次讀取 4 個位元組資料。假使此次是讀取 1 或 2 位元組資料的動作時會先作 zero extension 或是 sign extension 的動作，之後才會傳回給處理機。若要寫入資料到記憶體時，會有一組訊號線用來指示現在在匯流排上的 4 個 位元組 資料有幾個 位元 是有效的。

對 2 或 4 位元組 資料作讀寫動作時可能會發生存取位元未對齊(unaligned access) 的情形，即當讀寫的位址不是在 4 個 位元組 的邊界 (boundary) 時就是 存取位元未對齊。當發生 存取位元未對齊 的情形時，不論讀或是寫外界記憶體都需發動兩次的記憶體讀或是寫的動作。在這兩次讀寫的過程中，執行管線是暫停不動的，等到對記憶體兩次讀寫完成之後，執行管線才會恢復動作。

匯流排介面單元負責 CCL Java CPU I 與外界輸出入裝置的溝通工作，所有的 處理機 控制或資料訊號都是透過匯流排介面單元，在特定的時脈週期時作輸出或輸入。

CCL Java CPU I 的對外匯流排只有一條，而卻有三種存取需求(request) 可能同時需要讀寫外界記憶體，這三

微指令循序處理器(Microcode Sequencer)主要是在指令解碼單元(DIU)級負責多週期指令 (multi-cycle instruction) 的解碼，CCL JAVA CPU I 多週期指令的定義為此指令需要由多個微指令來完成。

微指令循序處理器(Microcode Sequencer)主要由微指令ROM(microinstruction ROM)、微指令位址(uPC)選擇電路、及循序處理器控制電路組成。微指令依uPC順序存放於微指令ROM(microinstruction ROM) ，微指令位址(uPC)選擇電路根據測試條件及微指令控制資訊選擇下個微指令的微指令位址(uPC) ，循序處理器控制電路負責整個DIU中有關循序處理器的控制。

當指令擷取單元(IFU)通知指令解碼單元(DIU)目前指令為多週期指令時，循序處理器控制電路負責啟動Sequencer，並將解碼控制權由原有單週期指令的解碼表(decoder table)交與Sequencer，同時發出暫停訊號，將前一管線級IFU的指令緩衝區的內容停住(halt)。Sequencer於下個時序週期將擷取到的第一個微指令傳往下一管線級，同時根據微指令的控制資訊決定下個微指令的uPC值，若遇到branch類微指令時，再根據測試條件由微指令位址選擇電路選擇uPC值。多週期指令的最後一個微指令的控制資訊會通知循序處理器控制電路此微指令為最後一個微指令，DIU會將暫停訊號拉下， IFU的指令緩衝區會在下個時序週期更新至新的指令，DIU也會在下個時序週期將解碼控制權交回DIU中的單週期指令的解碼表。

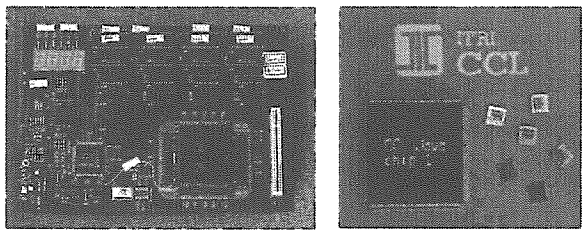
當外界中斷在多週期指令的中斷任何一個微指令時，JAVA CPU II微指令的控制資訊的設計能提供系統必要的控制資訊，啟動相關的電路，使能維護指令的精確中斷(precise interrupt) 。

5 總結

電通所 Java 處理機的整體設計完全是從無到有，在電通所中自行研發的。Java虛擬機器的規格在1996年中才定案，我們以它們為基礎，自行定義電通所 Java處理機的架構，而後進行IC的設計、開發及驗證。第一代的Java處理機核心-- CCL Java CPU ，已授權給世界先進。圖2.為CCL Java CPU I的驗證板和IC，其晶片面積約為9.5mm^2，使用TSMC 0.5um的製程。CCL Java CPU 實際測量獲得的效能資料和其它處理機的比較如表一，在執行Java時CCL Java CPU I的效能約為其它處理機的2至9倍。

今年六月我們完成了第二代的完整的電通所 Java 處理機—CCL Java CPU I 的製作與設計，我們依然使用TSMC 0.5um的製程，它共有27萬個電晶體，晶片面積約為25mm^2，除了約20%的效能提昇外，CCL Java CPU II支援完整的Java bytecod 直接執行，並有75個extended bytecode以支援OS、JVM、Compiler、chip set、和系統板的需要。電通所Java 處理機未來可能的發展方向包括：高頻低耗電之處理機設計、支援語音及媒體處理單元之設計、及整合無線和各類輸出入週邊的單晶片處理機功能等等。

圖 2. CCL Java CP U 的驗證板和 IC



表一. Java 測試程式之執行時間比較

(以相同頻率來計算)

CPU Name		CCL JAVA CPU I	PENTIU M II	PowerPC 603e	StrongARM SA1100
測試程式	nxn.j	1	1.92	2.8	3.35
	push.j	1	2.04	2.8	2.71
	many.j	1	4.44	6.15	9.42
	jump.j	1	2.52	2.8	3.25
	sort30.j	1	1.68	2.61	2.61

本文係工研院電通所執行經濟部委託之「Java處理機核心技術」計畫的成果之一。

參考文獻

[1] 汪自強、尚希聖、許家彰，"網路電腦的明日之星--JAVA處理器 ，" CCL Technical Journal ，57期，1997， pp.3-12.

種存取需求分別是記憶體存取需求、指令存取需求以及快取堆疊存取需求。假設這三種存取需求同時發生，同時需要去讀取記憶體，這時就需要一個匯流排仲裁機構來決定哪一種存取需求有較高優先權，可以先使用匯流排。在不同的狀態時，三種存取需求有不同的優先權，一般而言同一時間發生多個存取需求時優先權是記憶體存取需求> 指令存取需求 > 快取堆疊存取需求，三者依序執行完後才接受下一個存取需求，以避免發生永遠搶不到bus資源的問題。

4.5 控制與中斷處理單元

控制與中斷處理單元，負責各級管線間之工作協調與處理執行過程中所發生之中斷情形。其在處理機之角色，相當於中樞控制神經，目的是使得處理機內部平行操作之單元和諧工作。其控制主要包括各管線級之進行或停止，中斷狀況仲裁，及指令執行流程控制。

控制單元在每週期控制各級管線之前進或停止，在正常執行狀況下，每一週期各級管線會前進一級以便處理一個新的指令，但是例如當指令緩衝區內之資料不足無法解碼，或記憶體存取尚未完成，或有資料危險發生時，管線級會被暫時停止，直至狀況已被處理不再存在為止。

在對執行指令各階段所產生之中斷狀況加以處理上，一般而言，中斷狀況產生之來源，可區分成處理機內部發生，或處理機外部發生。處理機內部所產生之中斷狀況，又可區分為解碼、快取堆疊存取、或執行時發生。解碼時所產生之中斷狀況，包括錯誤的運算碼(invalid opcode)、軟體產生的中斷(software trap)、等。快取堆疊存取之中斷狀況，主要來源是因設定堆疊 spill/fill功能時發生不合理之情形。執行指令之中斷狀況，包括除零(divide by zero) 、無效的指標(null pointer)、超過陣列邊緣(array out of bound)等。處理機外部發生之中斷，包括硬體中斷(hardware interrupt 、不可遮斷的中斷(non-maskable interrupt)等。因各類之中斷狀況可以同時發生，此控制單元負責仲裁決定中斷處理之優先次序。

在決定指令執行流程上，有四種可能情形：(1)指令循序執行；(2)預測有分支情形發生；(3)分支預測發生錯誤時；(4)有中斷情形發生要處理；及(5)處理機電源啟動時。當指令循序執行時，此單元要決定下一次指令擷取之起始位址。當預測有分支情形發生或是分支預測發生錯誤時，此單元除要將指令擷取之起始位址設定為正確之目標位址外，也要清除管線中之未完成指令及調整處理機狀態(processor state)，以便執行正確位址之指令。當有中斷情形發生時，處理機內部之處理與分支情形發生時類似，只不過分支目標位址由存於系統記憶體中之向量表(vector table)決定。當電源啟動時，此單元讓指令從一固定位址開始擷取與執

4.6 動態分支指令預測

CCL Java CPU I 之分支預測機構(branch prediction mechanism)的規格是動態(dynamic)預測，也就是說在程式執行時依照跳躍指令執行歷史來預測程式執行順序有無改變。但是CCL Java CPU I 規格上並無類似跳躍目的暫存區(BTB, branch target buffer)的裝置用來儲存個別跳躍指令的執行歷史，因此只能記錄跳躍指令的共同歷史(global history)，也就是以前幾個跳躍指令的執行結果來預測目前跳躍指令的行為。共同歷史是以兩個位元(bit)的計數器(counter)來記錄，計數器的輸出值會分別定義成跳躍或是不跳躍，當要預測跳躍指令的行為時，便會去參考此計數器的輸出值來作決定。當此跳躍指令已確實知道是否跳躍時再依實際結果去改變計數器的值。

由於JVM指令集所定義之跳躍指令的目的位址絕大部份是程式計數器相對定址(PC relative)方式，因此在指令解碼階段才可判斷出目前指令是否為跳躍指令以及得出跳躍指令的目的位址。而控制與中斷處理單元依照指令解碼階段所傳來的資訊，並根據前幾個跳躍指令的執行歷史來推斷程式執行順序有無改變。跳躍指令的相關資訊會隨著指令流過管線到達指令執行階段，在指令執行階段即可確定目前跳躍指令會不會改變程式執行順序。而控制與中斷處理單元根據指令執行階段執行的結果與之前控制與中斷處理單元針對此跳躍指令所作的預測做比較，若預測不正確則必須執行跳躍錯誤預測回復(branch misprediction recovery)動作，重新由此跳躍指令的另一分支開始執行。控制與中斷處理單元並根據指令執行階段的執行結果更改其跳躍指令共同歷史，使得下一跳躍指令可以根據此共同歷史做跳躍的分支預測。

4.7 微指令循序處理器

[2] Tim Lindholm and Frank Yellin ， The Java Virtual 機器 Specification， Addison Wesley， 1996.

[3] 馬瑞良、尚希聖，Java硬體加速器中快取堆疊之設計與考慮，工研院電通所技術資料，文件編號：113862274，1997。

[4] J. Michael O'Connor， Marc Tremblay， "picoJava-I: The Java Virtual 機器 In Hardware，" IEEE Micro， March/April， 1997， pp. 45-53.

[5] 汪自強、馬瑞良、汲世安、尚希聖，"CCL Java Chip I 設計驗證方法與模擬環境 ，" CCL Technical Journal ，67期，1998， pp.9-14.

[6] P. Wayner， "Sun Gambles on Java Chips ，" *byte*， Nov. 1996， pp. 79-88.

[7] 馬瑞良、汪自強、汲世安、黃弘一、尚希聖，"CC Java Chip I 微架構設計 ，" CCL Technical Journal ，67期，1998， pp.3-8.

[8] 賴襄治，Java原始程式安裝及內部資料追蹤報告書，工研院電通所技術資料，文件編號：113862344，1997.

[9] Han-Min Tseng， Lung-Chang Chang， Lee-Ren Ton ， Min-Fu Kao ， Shi-Sheng Shang and Chung-Ping Chung "Performance Enhancement by Instruction Folding Strategies of a Java Processor ，" computer workshop， NCTU， 1997.