(19) österreichisches patentamt

(10) **AT 500 858 B1 2006-04-15**

(12) **Patentschrift**

(21) Anmeldenummer: A 1383/2004

(22) Anmeldetag: 2004-08-17

(43) Veröffentlicht am: 2006-04-15

(51) Int. Cl.[8]: ***G06F 9/30*** (2006.01)
G06F 12/04 (2006.01)

(73) Patentanmelder:
SCHÖBERL MARTIN
A-1050 WIEN (AT)

(72) Erfinder:
SCHÖBERL MARTIN
WIEN (AT)

(54) INSTRUCTION CACHE FÜR ECHTZEITSYSTEME

(57) Für Echtzeitsysteme ist die Kenntnis der maximalen Ausführungszeit (engl. Worst Case Execution Time - WCET) von fundamentaler Bedeutung. Der Einfluss des Instruction Caches auf diese WCET ist bei konventionellen Designs schwer vorhersehbar und führt zu pessimistischen WCET Werten. Es wird ein Instruction Cache 103 für einen Prozessor 100 beschrieben, der eine genauere Vorhersagbarkeit des Echtzeitverhaltens ermöglicht. Der Instruction Cache wird mit kompletten Funktionen geladen und fasst alle 'cach misses' beim Funktionsaufruf und der Funktionsrückkehr zusammen. Die Analyse des Cacheverhaltens ist dadurch auf die Analyse des 'call trees' reduziert.

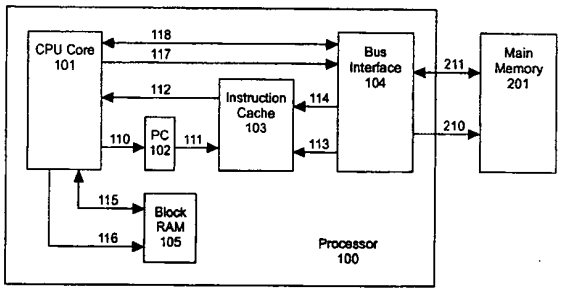

Fig. 1

AT 500 858 B1 2006-04-15

Die Erfindung bezieht sich auf einen echtzeitfähigen Instruction Cache.

In Echtzeitsystemen ist die Korrektheit eines Programms nur gegeben, wenn neben der algorithmischen Korrektheit auch zeitliche Bedingungen eingehalten werden. Um diese zeitlichen Bedingungen einhalten zu können muss die maximale Ausführungszeit von Programmen bekannt sein. Diese Werte sind die Basis jeder 'schedulability' Analyse.

Die maximale Ausführungszeit von Programmen muss durch Analyse der Programme und der dazu notwendigen Modellierung des Systems erfolgen. Eine Messung der Ausführungszeit ist nicht möglich, da nicht sichergestellt werden kann, dass alle Kombinationen von Ausführungspfaden durchlaufen wurden.

Cache Speicher sind ein wichtiger Bestandteil von Prozessoren um den Geschwindigkeitsunterschied zwischen Hauptspeicher und Prozessor auszugleichen. Die bekannten Cache Architekturen sind jedoch für die durchschnittliche Performanz und nicht für vorhersagbare Performanz optimiert. Dies führt zu schwer vorhersagbaren bzw. sehr pessimistischen WCET Werten. In (Proc. of the IEEE, 91(7): 1038-1054, Jul. 2003) werden Caches von realen Prozessoren analysiert. Architekturmerkmale führen dazu, dass nur 1/2 bzw. 1/4 des vorhandenen Cache Speicher modelliert werden können.

Ein Ansatz zur Lösung dieser Problematik besteht aus der Teilung des Instruciton Caches in einen Block für allgemeine Programme und einen Block für echtzeit relevanten Code (z.B.: EP 0 529 217 A1 oder US 5,913,224). Der Echtzeitcode wird vor der Ausführung in einen Cacheblock geladen und dieser Block gesperrt. Dieser Block enthält dann während der kompletten Laufzeit den echtzeit relevanten Programmteil. Diese Lösung ist jedoch sehr inflexibel und beschränkt die maximale Größe der Echtzeitprogramme auf die Cachegröße.

Der Erfindung liegt die Aufgabe zugrunde einen Instruction Cache zu gestalten, dessen Echtzeitverhalten genauer modelliert werden kann ohne die Programmgröße einzuschränken.

Die Aufgabe wird dadurch gelöst, dass komplette Funktionen im Instruction Cache gespeichert werden. Das Laden des Instruction Caches erfolgt nur, wenn notwendig, bei einem Funktionsaufruf bzw. bei einer Funktionsrückkehr.

Da sichergestellt ist, dass eine Funktion bei der Ausführung komplett im Instruction Cache geladen ist, fallen keine Cache bedingten Wartezeiten während der Ausführung der Funktion an. Der Cache muss daher nur bei Funktionsaufruf und Funktionsrückkehr in der WCET Analyse berücksichtigt werden. Die Entscheidung ob ein 'cach hit' oder 'cache miss' vorliegt ist nur vom Aufrufbaum der Funktionen bestimmt und nicht von den Adressen der einzelnen Instruktionen.

Funktionen werden nur relativ adressiert. D.h. es sind innerhalb der Funktion nur relative Sprünge möglich. Diese Bedingung ist z.B. in dem Zwischenkode der Sprache Java erfüllt. Daher eignet sich dieser Instruction Cache sehr gut für einen echtzeitfähigen Java Prozessor. Java ist aber nur als Beispiel für die Anwendung dieses Instruction Caches zu verstehen. Auch andere Programmiersprachen, wie z.B.: C, lassen sich auf eine Weise Übersetzen, die nur relative Sprünge innerhalb von Funktionen enthält.

Durch die relative Adressierung ist es während der Funktionsausführung irrelevant an welcher Cacheposition die Funktion beginnt. Der Program Counter 102 muss nur beim Funktionsaufruf mit der Startadresse im Cache geladen werden.

Um mehr als nur eine Funktion im Instruction Cache halten zu können wird dieser in Blöcke eingeteilt. Eine Funktion kann sich über mehrere zusammenhängende Blöcke erstrecken. Wobei ein Zusammenhang auch vom letzen Block zum ersten Block besteht, da der Program

Counter auf die Cacheadressierung begrenzt ist und es dadurch zu einem automatisch korrekten Über- bzw. Unterlauf kommt.

Durch die relative Adressierung können der Programm Counter 102 und die zughörigen Busse und Multiplexer einfacher, da kleiner, realisiert werden. Auch die Adressübersetzung für die Implementierung eines virtuellen Speichers ist nur mehr beim Laden einer Funktion notwendig.

Die Feststellung eines 'cach hit' ist nur beim Funktionsaufruf bzw. bei der Rückkehr notwendig und wird durch Lesen des Block RAM 105 gelöst. Das in konventionellen Caches notwendig 'tag RAM', das bei jedem Cachezugriff gelesen werden und mit der Adresse verglichen werden muss, kann dadurch entfallen. Der Zugriff auf das 'tag RAM' und der Adressenvergleich liegen normalerweise im kritischen Pfad der Hardware und bestimmen dadurch die minimale Zugriffszeit auf den Cache. Ohne Vergleich bei jedem Zugriff, wie in dieser Erfindung, kann die Zugriffszeit auf den Cache bei gleicher Technologie verringert werden.

Das Laden kompletter Funktionen, und damit größerer Blöcke als bei einem konventionellen Cache, wirkt sich auch positiv bei Verwendung von dynamischen Speichern für den Hauptspeicher 201 aus. Diese Speichertechnologie zeichnet sich dadurch aus, dass das erste Wort erst nach einer beträchtlichen Verzögerung verfügbar ist, jedoch die folgenden nach kürzerer Zeit. Diese Initialverzögerung wirkt sich bei größeren Blocken weniger aus, als bei kleinen Blöcken.

In Fig. 1 wird die Architektur eines Prozessors dargestellt, der den Erfindungsgegenstand enthält. Fig. 2 zeigt exemplarisch die Belegung der Cache Blöcke bei der Ausführung des Programmfragments in Fig. 3.

Der Instruction Cache 103 liegt zwischen dem Prozessorkern 101 und dem Bus Interface 104. Instruktionen werden über den Bus 112 vom Instruction Cache 103 geholt. Der Instruction Cache 103 wird über den Program Counter 102 adressiert. Da dieser nur den Cache adressiert muss dieser und die zugeörigen Busse 110 und 111 log2 (Cachegröße) Bits breit sein.

Der Instruction Cache 103 wird vom Bus Interface 104 aus dem Hauptspeicher 201 mit kompletten Funktionen gefüllt. Die Busse 113 und 114 sind die Adress- bzw. Datenbusse zwischen dem Bus Interface 104 und dem Instruction Cache 103. Über den Adressbus 117 und dem Datenbus 118 werden Lade- und Speicheranforderungen des Prozessorkerns 101 abgewickelt.

Das Bus Interface 104 wickelt den Datenaustausch und das Laden des Instruction Caches 103 mit dem Hauptspeicher 201 über den Adressbus 210 und dem Datenbus 211 ab. Da das Laden des Instruction Caches 103 nur bei einem Funktionsaufruf oder einer Rückkehr aus einer Funktion passiert, kommt es zu keinen Konflikten mit den Lade- und Speicheranforderungen des Prozessorkerns 101.

Der Block RAM 105 dient dem Prozessor zur Speicherung welche Blöcke des Instruction Caches 103 von welchen Funktionen belegt sind. Er wird über den Adressbus 116 und den Datenbus 115 angesprochen.

Fig. 2 zeigt die Belegung von Cache Blöcken während der Ausführung des in Fig. 3 skizzierte Programms. Die Anzahl der Blöcke und die Strategie welche Blöcke ersetzt werden ist nur exemplarisch. Die Ersetzungsstrategie kann komplexer als bei herkömmlichen Instruction Caches ausfallen, da die Entscheidung seltener (nur beim Laden einer kompletten Funktion) anfällt. Die Belegung der Blöcke wird in Block RAM 105 gespeichert. Dieser muss gelesen werden um festzustellen ob ein 'cach hit' vorliegt und geschrieben werden, wenn eine Funktion neu in den Instruction Cache geladen wird.

Das Beispiel in Fig. 2 besteht aus 4 Funktionen, wobei die Funktionen A() und D() klein genug sind um in einen Block zu passen. Funktionen B() und C() sind größer und belegen zwei Blöcke.

301 zeigt den Zustand nach dem Aufruf der Funktion A(). Der erste Block ist belegt, die restlichen drei sind frei. Der Aufruf der Funktion B() innerhalb von A() führt zur Belegung wie in 302 gezeigt. Es ist nur mehr ein Block frei. Die Funktion C(), die von B() aufgerufen wird benötigt jedoch zwei Blöcke. Wie in 303 gezeigt wird C() in Block 4 und Block 1 geladen, wodurch Funktion A() nicht mehr im Cache ist.

Die Adressierung der Funktion C() über das Cacheende (Block 4) zum Cacheanfang (Block 1) geschieht implizit durch die Begrenzung vom Program Counter 102 auf die Cachegröße. Die Addition bzw. Subtraktion über die Cachgrenze hinaus ergibt implizit den korreten Überlauf bzw. Unterlauf des Program Counters 102.

Bei der Rückkehr von Funktion C() zur Funktion B() ist kein Laden des Caches notwendig, da Sich Funktion B() zu diesem Zeitpunkt noch im Cache befindet. Der Aufruf von Funktion D() führt zur Belegung wie in 304 gezeigt. Obwohl D() nur einen Block belegt und damit einen Teil B() verdrängt, ist Block 3 als unbelegt markiert. Dies ist Notwendig, da nur komplette Funktionen gültig sind.

Die Entscheidung ob D() Funktion B() oder Funktion C() aus dem Cache verdrängt ist abhängig von der Ersatzstrategie. In diesem Beispiel wird jeweils der nächste Block nach einer geladenen Funktion als Startblock für eine neue zu ladende Funktion verwendet. Dies ist aber nur eine Möglichkeit von vielen (z.B.: 'last recently used' oder 'best fit'). Ebenfalls ist die Einteilung in vier Blöcken nur examplarisch zur Vereinfachung der Illustration.

Patentansprüche:

1. Verfahren zum Betrieb eines Instruction Caches, *dadurch gekennzeichnet*, dass komplette Funktionen gespeichert werden, die nur bei einem Funktionsaufruf oder einem Rücksprung aus einer Funktion geladen werden.

2. Verfahren nach Anspruch 1, *dadurch gekennzeichnet*, dass Funktionen innerhalb des Caches nur relativ adressiert werden und dadurch mehrere Funktionen jeweils in aufeinanderfolgenden Blöcken gehalten werden können.

3. Verfahren nach Anspruch 1 oder 2, *dadurch gekennzeichnet*, dass das "tag memory" durch einen "Block RAM" (105) ersetzt ist, welches nur bei Funktionsaufruf bzw. Funktionsrückkehr vom Prozessorkern (101) gelesen bzw. geschrieben wird.

4. Instruciton Cache, *dadurch gekennzeichnet*, dass er nach einem der Verfahren nach Ansprüchen 1-3 betrieben wird.

Hiezu 2 Blatt Zeichnungen

Int. Cl.[8]: ***G06F 9/30*** (2006.01)
G06F 12/04 (2006.01)

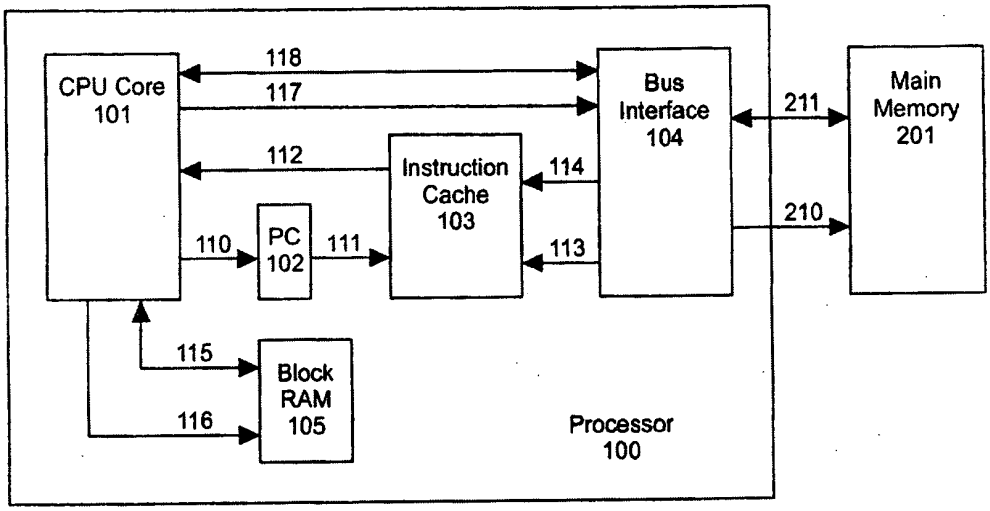

Fig. 1

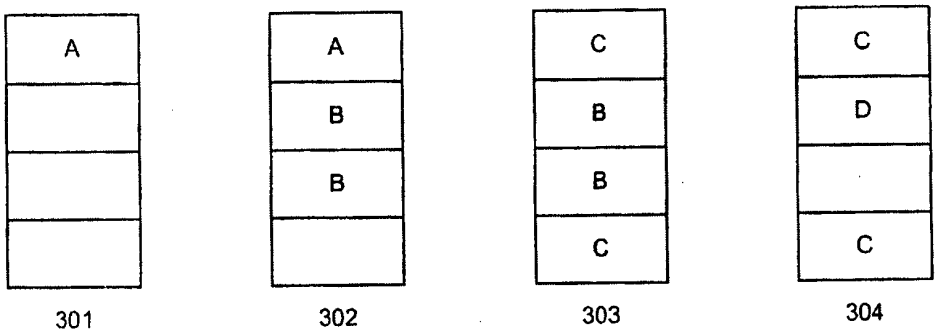

Fig. 2

Int. Cl.[8]: ***G06F 9/30*** (2006.01)
G06F 12/04 (2006.01)

```
A() {
     ...
     B();
     ...
}

B() {
     ...
     C();
     ...
     D();
     ...
}
```

Fig. 3