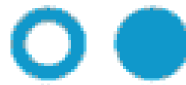
Parallelität in der Intel IA-64 Architektur

Arbeit im Rahmen der Veranstaltung Verteilte und Parallele Systeme

Fachhochschule Bonn Rhein Sieg



Fachhochschule
Bonn-Rhein-Sieg

Verfasser: Daniel Post

Ort, Datum: Bonn im Januar 2002

Inhaltsverzeichnis

Inhaltsverzeichnis **2**
1. Einführung in das Thema **3**
1.1 Begriffsdefinition 3
1.1.1 IA-64 Architektur 3
1.1.2 EPIC 3
1.1.3 Itanium 3
1.2 Grundsätzliche Funktionsweise eines Prozessors 3
2. Parallelität in modernen Prozessoren **4**
2.1 Datenabhängigkeiten 5
2.2 Sprungabhängigkeiten 5
3. Die IA-64 Architektur **6**
3.1 Das EPIC Konzept 6
3.1.1 Die Befehlsbündel 6
3.2 Der 64 Bit Adressbus 8
3.3 Die Verarbeitungseinheiten 8
3.4 Die Register 8
3.4.1 General Purpose Register 8
3.4.2 Floating Point Register 8
3.4.3 Predication Register 8
3.4.4 Branch Register 8
3.4.5 Application Register 9
3.4.6 Register Stack Engine 9
3.4.7 Register Rotation 9
3.5 Der Speicher 10
3.5.1 Virtuelle Adressierung 10
3.5.2 Translation Lookaside Buffer 10
3.6 Die Gleitkomma Berechnung 11
3.6.1 Berechnungsmodi 11
3.7 Branch Prediction 12
3.8 Spekulationen 13
3.8.1 Control Speculation 13
3.8.2 Data Speculation 13
4. Zusammenfassung **14**
Abbildungsverzeichnis **15**
Literaturverzeichnis **15**

1. Einführung in das Thema

Diese Ausarbeitung ist im Rahmen der Vorlesung Verteilte und Parallele Systeme entstanden. Diese Vorlesungsreihe beinhaltet Themen zur Parallelverarbeitung von Daten. Hierbei wird sowohl auf die Softwareseite (Java Threads, OpenMP und MPI) wie auch auf die Hardware (Prozessoren, Speicher, Netzwerke) engegegangen.

Intel beschreitet in der neuen IA-64 Architektur einen ähnlichen Weg. Hier wird versucht, eine optimale Kopplung der Aufgaben zwischen dem Compiler und der Hardware herzustellen. Dafür sind im Microcode von IA-64 extra Befehle vorgesehen, um Parallelitäten im Programmcode auszudrücken, die von der Hardware erkannt und umgesetzt werden kann.

Im weiteren Verlauf des ersten Kapitels bespreche ich zur Wiederholung die Grundfunktionen eines Prozessors.

Der zweite Abschnitt befaßt sich mit der parallelen Verarbeitung von Daten, und welche Vorraussetzungen dafür geschaffen werden müssen.

Das Kapitel 3 beschreibt die IA-64 Architektur, und geht näher auf die Methoden zur Parallelverarbeitung in dieser Architektur ein.

1.1 Begriffsdefinition

Bevor ich in die Thematik einsteige, möchte folgende Begriffe kurz erklären.

1.1.1 IA-64 Architektur

IA-64 steht für Intel Architecture. Die Architektur beschreibt ein grundsätzliches Prozessordesign, und legt die prinzipielle Arbeitsweise und die Komponenten innerhalb des Prozessors fest. Weiter wird in IA-64 das Instruction Set (alle Micro Befehle) definiert. Kapazität und Geschwindigkeit von IA-64 Prozessoren hängen dagegen nicht von der Architektur, sondern von der konkreten Implementierung eines Prozessors ab.

1.1.2 EPIC

EPIC – Explici Parallel Instuction Computing – ist ebenso eine Prozessorarchitektur wie z.B. RISC(Reduced Instruction Set Computing), oder CISC(Complex Instruction Set Computing). In der IA-64 findet das EPIC Modell Anwendung.

1.1.3 Itanium

Der Intel Itanium Prozessor ist der erste Prozessor, der nach der neuen IA-64 Architekture entwickelt wurde.

1.2 Grundsätzliche Funktionsweise eines Prozessors

Eine Central Processing Unit (CPU) beinhaltet im wesentlichen folgende Elemente:

Verarbeitungseinheiten: Sie führen die Rechenoperationen aus, und werden allgemein als Arithmetische logische Einheit (ALU) bezeichnet.

Register: Register sind 1 – 8 Wort Speicher und zeichnen sich durch eine sehr schnelle Zugriffszeit aus. Die Recheneinheiten beziehen ihre Operanden aus den Registern, und schreiben auch das Ergebnis in ein Register zurück.

Es kann zwischen Control und Datenregistern unterschieden werden, wobei die Datenregister ausschließlich für die ALU Einheiten bereitstehen, und die Controlregister z.B. Systemzähler wie den Stackpointer oder Befehlsregister darstellen.

Systembus: Alle Einheiten sind innerhalb der CPU über einen Systembus verbunden. Dieser kann je nach Architektur der CPU zwischen 8 und 128 Bit breit sein.

I/O Einheiten: Die Ein/ Ausgabeeinheiten steuern alle Datenflüsse von und zu externen Systemen wie Speicher, Seriellen Schnittstellen oder externen Bussystemen wie PCI.

CPU Steuerung: Hier findet die Befehlsdecodierung, die Flusssteuerung sowie das Interrupt Management für die CPU statt.

Das nachfolgende Beispiel zeigt die 4 wesentlichen Schritte, um eine Instruktion durch die CPU ausführen zu lassen:

1. Die Instruktion muss aus dem Befehlsregister geladen werden (*Instruction fetch: F*)

2. Die Operanden der Instruktion werden in entsprechende Datenregister geladen und der Befehl wird dekodiert (*Decode: D*)

3. Die Instruktion wird durch die Recheneinheit ausgeführt (*Execute: E*)

4. Das Ergebnis wird in ein Register geschrieben (*Write: W*)

Den oben beschriebenen Ablauf nennt man auch einen Befehlszyklus. Jedes Element eines Befehlszyklus nennt man eine Stufe.

2. Parallelität in modernen Prozessoren

Alle modernen Prozessoren, CISC wie auch RISC, setzen auf die sequentielle Verarbeitung von Befehlen auf. Alle Befehle werden durch die einzelnen Stufen des Befehlszyklus abgearbeitet. Parallelität kann durch das Instruktion Pipelining (Fließbandverarbeitung) erzeugt werden. Hier werden die Befehle pro Taktstufe in die Pipeline eingestellt, so dass jede Stufe immer ausgelastet ist. Das folgende Beispiel verdeutlicht das Prinzip:

1. sequentielle Befehlsausführung

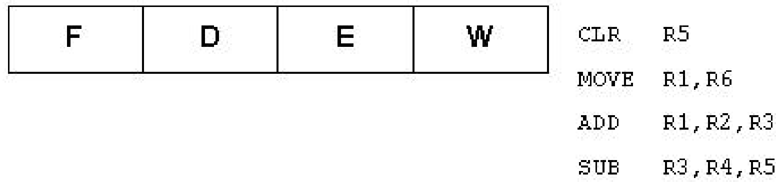


Abbildung 1: sequentielle Befehlsausführung

2. Instuction Pipelining

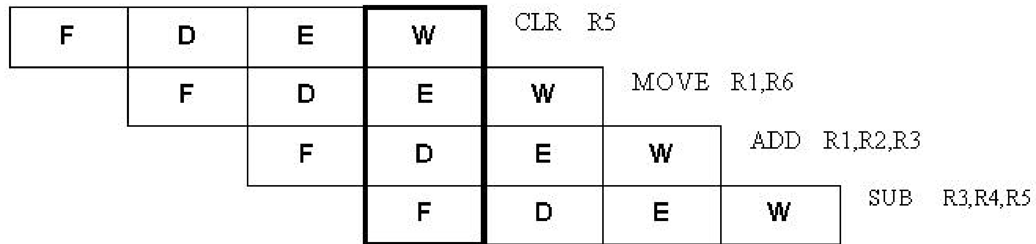


Abbildung 2: Instruction Pipelining

Bei der Instruction Pipeline bezeichnet man den Bereich in dem die Pipeline noch nicht gefüllt ist als Prolog, den ganz gefüllten Mittelteil der Pipeline nennt man Kernel und die Phase der Leerung der Pipeline Epilog.
Diese Art von Pipelining findet auch als Software Pipelining Anwendung.

2.1 Datenabhängigkeiten

Eines der beiden Hauptprobleme bei der Ausführung von Instruction Pipelining ist die Datenabhängigkeit. Wenn das Ergebnis eines datenverarbeitenden Befehls durch den darauf folgenden Befehl weiterverarbeitet wird, kommt es zu einem Konflikt. Der erste Befehl kann sein Ergebnis nicht rechtzeitig für den zweiten Befehl zur Verfügung stellen. Ein Konfliktfall tritt immer dann auf, wenn die Zielregisteradresse eines Befehls gleich der Quellregisteradresse eines der beiden nachfolgenden Befehle ist.

Da Datenabhängigkeiten recht häufig auftreten, müssen sie durch Hardware aufgelöst werden. Zum einen kann man die ersten beiden Stufen der Pipeline stoppen bis das Ergebnis ins Zielregister geschrieben wurde. Diese Methode unterbricht die Pipeline und senkt die Performance des Prozessors. Zum anderen könnte das Ergebnis des ersten Befehls direkt auf einen Eingang der Verarbeitungseinheit geschrieben werden, und steht so einen Takt vorher zur Verfügung. Diese Methode nennt man Bypassing.

2.2 Sprungabhängigkeiten

Das zweite Problem bei der Fleißbandverarbeitung wird durch die Sprungbefehle (Branches) erzeugt. Wenn ein Sprungbefehl in einer Instruction Pipeline einen neuen Programmzweig einschlägt (z.B. durch eine true /false Bedingung), dann wird dieser Programmzweig von der Instruction Pipeline ausgeführt. Die Befehle oberhalb des Sprungbefehls müssen aus der Pipeline entfernt werden, und die Befehle im neuen Programmzweig der Pipeline zugeführt werden. All diese Maßnahmen „bremsen" die Effizients der Pipeline erheblich.

Um das Problem zu lösen, wird durch Sprungvorhersagen versucht Sprungentscheidungen zu erkennen und daraufhin den Befehlsstrom anzupassen.

3. Die IA-64 Architektur

Die IA-64 Architektur sieht eine massive Unterstützung für die parallele Verarbeitung von Daten vor. Durch Anweisungen, die in der IA-64 Instruction Set Architecture (ISA) vorgesehen sind, können Bündle von mehreren Befehlen parallel für die Verarbeitung bereitgestellt werden. Die Parallelisierung auf der Instruktionsebenen – Instuction Level Parallism(ILP) erfordert viele Hardware Ressourcen. Hierfür sind in IA-64 eine große Anzahl von verschiedenen Registern, Verarbeitungseinheiten vorgesehen.

3.1 Das EPIC Konzept

Im Konzept von EPIC ist vorgesehen, das der Compiler den zu berechnenden Code möglichst so optimal vorbereitet, das alle Instruktionen eines Bündels datentunabhängig sind und somit parallel ausgeführt werden können. Dem Programmierer steht es offen, ob er selbst Anweisungen zur parallelen Ausführung von Instruktionen in den Code einbringen will. Zur Fehlererkennung bzgl. Daten und Sprungabhängigkeiten steht dem Programmierer IA-64 Code Checker Software zur Verfügung.

EPIC basiert auf der Kombination von Softwarefunktionen und Hardware. Ziel ist es einen möglichtst hohen Grad an paralleler Verarbeitung zu erreichen. Hierfür übernehmen der Compiler und die Hardware verschiedene Aufgaben.

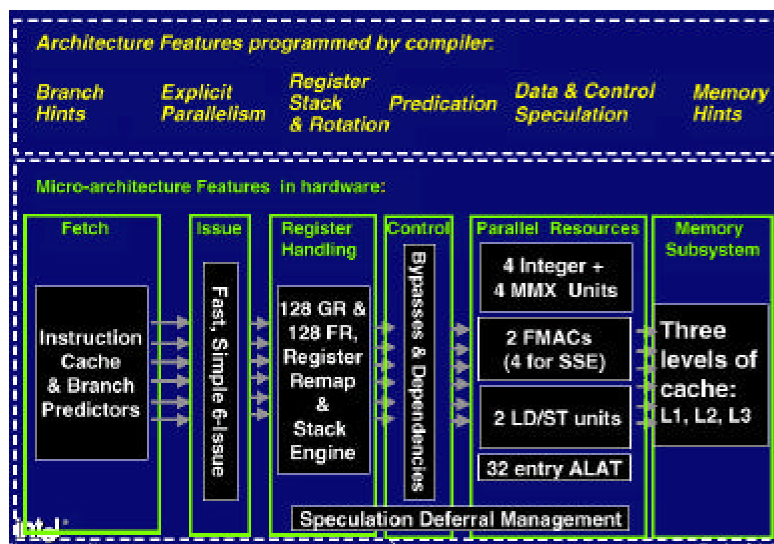


Abbildung 3: Das EPIC Design

3.1.1 Die Befehlsbündel

Bei EPIC sind Befehle zu Bündeln von 3 Befehlen zusammengefaßt.

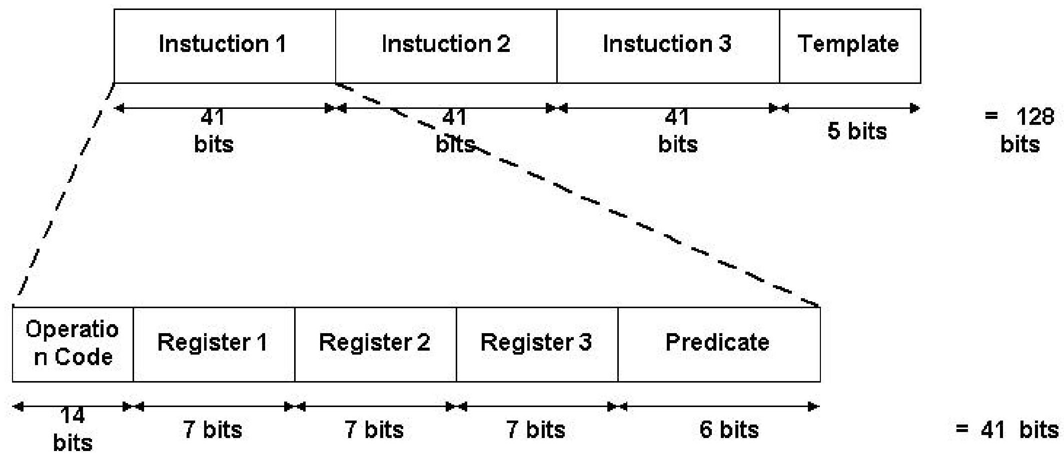


Abbildung 4: EPIC Befehlsbündel

Das Template Feld im Befehlsbündel gibt an, um welche Befehlsarten es sich in den einzelnen Instruktionen handelt, und welche davon parallel ausgeführt werden können. Durch die 5 Bit in diesem Feld können max. 32 Kombinationen an Instruktionen von 00 bis 1F gebildet werden. 24 dieser Kombinationen sind bereits definiert.

Befehlstyp	**Abkürzung**	**Beschreibung**
Memory	M	M-Unit
Non Integer ALU	I	I-Unit
Floating Point Unit	F	F-Unit
Branch	B	B-Unit
64 Bit immediate	L+X	I-Unit/ B-Unit
Integer ALU	A	I-Unit/ M-Unit

Tabelle 1: Befehlsarten in IA-64

Die Tabelle 3.1 zeigt alle definierten Befehlsarten in IA-64. Jeder Befehlstyp spricht eine Verarbeitungseinheit (Unit) in IA-64 an. So können verschiedene Befehlsbündel parallel abgearbeitet werden. Eine Kombination könnte z.B. MII lauten. Hier würden sich dann ein Memory Befehl und 2 Integer Befehle im Befehlsbündel parallel verarbeiten lassen. Das MII Befehlsbündel wird im Template Feld durch 00 oder 01 dargestellt. Wenn es zwischen den beiden Integer Befehlen zu einer Datenabhängigkeit kommt (keine parallele Verarbeitung möglich), dann kommt die MII Template Kombination 02 oder 03 zum tragen, in der zwischen der ersten und zweiten Integer Anweisung ein Wartetakt eingebaut ist, um das Ergebnis von der ersten an die zweite Instruktion zu übergeben.

Nicht alle Kombinationen von Befehlsarten sind erlaubt. Jedes Befehlsbündel kann z.B. nur eine Gleitkomma Operation enthalten. Durch den Befehl LX wird ein 64 Bit Wert an ein Zielregister übergeben, wofür 2 Befehlsplätze innerhalb des Befehlsbündels genutzt werden. Diese LX Anweisung wird nur im Zusammenhang mit einem Memory Befehl ausgeführt. Die beiden gültigen Kombinationen im Template für diese Operationen lauten 04 und 05. Beide führen das Befehlsbündel MLX aus. Die Operation wird genutzt, um Integer und Branch Registern 64 Bit Speicheradressen zuzuweisen (indirekte Adressierung).

3.2 Der 64 Bit Adressbus

Ein wesentliches Merkmal der IA-64 Architektur ist der 64 Bit breite Adressbus. Er ermöglicht eine Adressierung von 18500 Terabyte an Hauptspeicher. Auch wenn diese Massen an Speicher in der nahen Zukunft wohl noch nicht benötigt werden, sind sie in der Architektur vorgesehen.

3.3 Die Verarbeitungseinheiten

Zur parallelen Ausführung von Programmcode werden mehrere parallele Verarbeitungseinheiten(Units) benötigt:

- Integer/ Multimedia Einheiten
- Gleitkomma Einheiten
- Programmsprung Einheiten

Alle Einheiten sind mehrfach implementiert und werden über sog. Issue Ports mit Instruktionen aus der Instruction Queue versorgt.

3.4 Die Register

Den einzelnen Verarbeitungseinheiten sind jeweils Register zugeordnet. Zusätzlich gibt es Register für Anwendungen und Vorhersagen.

3.4.1 General Purpose Register

Es gibt 128 sog. General Purpose Register (GR), die Werte für Integer und Multimedia Berechnungen vorhalten. Sie sind 64 Bit breit, und verfügen über ein zusätzliches Bit, das Not a Thing (NaT) genannt wird. Das NaT Bit wird gesetzt, wenn sich in dem Register spekulativ geladenen Daten befinden, die bei einer Exception ungültig werden können. Mehr zu Datenspekulationen später im Text.
Das erste GR Register r0 ist dauerhaft auf null gesetzt, und als read only angelegt. Weiterhin sind die ersten 32 Register statisch, wohingegen Register r32 bis r127 „stacked" sind. Die oberen 96 Register lassen sich also dynamisch verwalten und rotieren. Die Rotation von Registern wird ebenfalls später in diesem Kapitel erläutert.

3.4.2 Floating Point Register

Die Floating Point Register (FP) sind 82 Bit breit und genau so wie die General Purpose Register in 32 statische und 96 dynamische Register aufgeteilt. Die Register 32 bis 127 können rotieren. Das Register r0 ist permanent auf null bzw. Register r1 auf 1 gesetzt. Beide Register (f0, f1) sind schreibgeschützt.

3.4.3 Predication Register

Die Predication Register p0-p63 sind nur ein Bit breit, und dienen zur Kontrolle von bedingten Anweisungen und bedingten Programmverzweigungen.
Das Register p0 ist schreibgeschützt und dauerhaft auf 1 (true) gesetzt.

3.4.4 Branch Register

In den 8 Branch Registern b0-b7 werden Zieladressen von indirekten Programmverzweigungen gespeichert. Die Register sind 64 Bit breit und werden von der Programmsprung Einheit (Branch Unit) genutzt.

3.4.5 Application Register

Für zukünftige Anwendungen sind in IA-64 128 Applikationsregister ar0-ar127 vorgesehen.

3.4.6 Register Stack Engine

Die Register Stack Engine (RSE) sorgt für die Registerverwaltung. Sie teilt jedem Prozess Register aus dem dynamischen Registerpool zu. Die zugeteilten Register stehen dann, wie z.B. lokale Variablen in höheren Programmiersprachen, lokal den Prozessen zur Verfügung. Alle Register eines Typs, die einer Prozedur zugewiesen sind, nennt man Register Frame. Die statischen Register(z.B. die GR Register r0-r31) können von allen Prozessen genutzt werden (vgl. statische Variablen in höheren Sprachen). Die Inhalte der statischen Register müssen durch die jeweiligen Prozeduren explizit gesichert werden.

3.4.7 Register Rotation

Um Gegen und Ausgabeabhängigkeiten (beides sind Datenabhängigkeiten) von Registern zu vermeiden, wird ein Verfahren namens Register Renaming angewendet. In der IA-64 Architektur wird es als Register Rotation bezeichnet. Ziel ist es, z.B. Programmschleifen und deren Iterationen in parallelen Einheiten auszuführen. Hierfür werden zusätzliche Register zur Verfügung gestellt. Wenn eine Programmschleife innerhalb einer Softwarepipeline parallel berechnet werden soll, dann muss sichergestellt sein, dass das Ergebnis der ersten Iteration der zweiten zur Verfügung steht.
Diese Anforderung kann entweder durch sog. Loop Unrollment, oder durch Software Pipelinig erfüllt werden. Beim Software Pipelining sind allerdings zusätzliche Instruktionen im Sourceode notwendig. IA-64 entgeht diesem Code Overhead, indem die Register Rotation automatisch durch die Angabe eines Befehls erfolgt. Danach werden die Iterationen einer Schleife im Pipeline Modes auf verschiedenen Einheiten ausgeführt.

Beispiel einer Register Rotation

Memcopy in C:

```
for (i = 0; i < n; i++)
{
      *a++ = *b++
}
```

Anweisungen innerhalb der Schleife in ASM64:

```
L1:   ld8 r35 = [r4], 8
      st8 [r5] = r37, 8
      swp_branch L1 ;;
```

Ld8 lädt ein 8 Byte Wort in GR r34 aus dem Speicherplatz, der durch r4 adressiert ist. Die nachfolgende 8 inkrementiert die Speicheradresse von r4 um 8. Dies geschieht nach der Werteübergabe an r35.

St8 speichert den Wert von r37 in den Speicher, der durch r5 adressiert ist.

Swp_branch ist eine Befehl, der eine Software Pipelined Loop kennzeichnet.

In diesem Beispiel steht der Wert in Register r35 nach 2 Iterationen(und 2 Register Rotationen) in Register r37 für den Speicherbefehl zur Verfügung. In der Zwischenzeit können bereits 2 weitere Load Iterationen gestartet werden. Wegen der Register Rotation kommt es zu keinen Überschneidungen zwischen den Load und Store Befehlen.

3.5 Der Speicher

Die Speicherverwaltung in IA-64 erfolgt ausschließlich über die indirekte Adressierung von Speicherplätzen durch die GR Register.
In der IA-64 Architektur ist eine intelligente Speicherverwaltung vorgesehen, die das spekulative Laden von Daten aus dem Hauptspeicher in die Register unterstützt. Weiterhin wird der Speicherplatz in Hierarchien unterteilt, um einen schnellen Zugriff zu gewährleisten.
Wie bei allen modernen Prozessoren gibt es zum einen den physikalischen Speicher, der als Cache oder RAM zur Verfügung steht. Zum anderen gibt es den virtuellen Speicher, dessen Pages je nach Bedarf dem physikalischen Speicher zugeordnet werden. Dieses Verfahren erlaubt es den physikalischen Speicher effizienter zu nutzen, und allen Prozessen virtuell komplette Speicherbereiche anzubieten.

Um virtuelle Adresse in physikalische umzusetzen werden Translation Lookaside Buffer (TLB) eingesetzt.

3.5.1 Virtuelle Adressierung

Der virtuelle Adressraum wird bei IA-64 in acht virtuelle Regionen von je 2^{61} Byte unterteilt. Innerhalb dieser Regionen können Pages verschiedener Größe angelegt werden. Um die einzelnen Regionen ansprechen zu können, sind 8 Region Register mit einer Datenbreite von 64 Bit vorgesehen. Diese Register beinhalten je einen 24 Bit lange Region Identifier (RI) einer Region, und zusätzliche Informationen für den TLB und das Paging. Eine virtuelle 64 Bit Adresse hat folgendes Format:

Abbildung 5: Aufbau der virtuellen Adresse

Mit den 3 Bits der Virtual Region Number (VRI) kann eines der acht Region Register angesprochen werden. Dadurch wird festgelegt, in welcher Region sich die virtuelle Adresse befindet. Je nach Größe einer Page ergibt sich die weitere Zusammensetzung der virtuellen Adresse. Die Virtual Page Number (VPN) kennzeichnet eine Page in der durch die VRI adressierten Region, der Offset einen Speicherstelle innerhalb dieser Page.

3.5.2 Translation Lookaside Buffer

Der Translation Lookaside Buffer führt eine Hash Tabelle, in der virtuelle auf physikalische Adressen gemapped werden. Für den Zugriffsschutz, von privatem Speicher einer Applikation, besitzt der TLB zusätzliche Datenfelder für jede Page, in denen Zugriffsinformationen festgehalten werden.

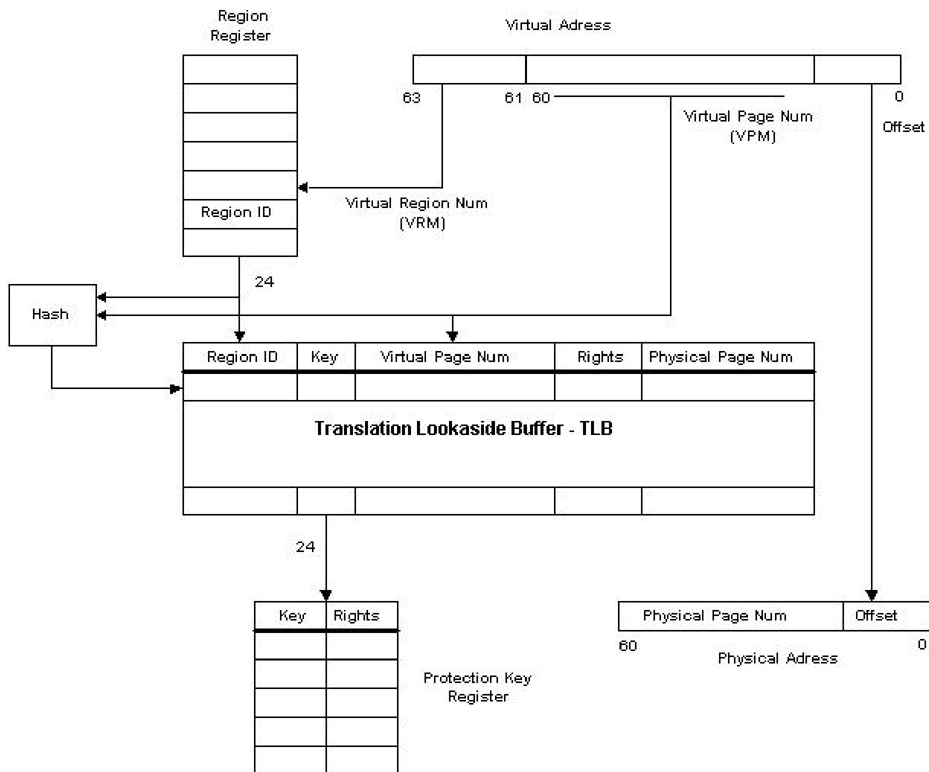


Abbildung 6: Prinzip der virtuellen Adressauflösung

Die Prozessorarchitektur hat 2 TLBs vorgesehen:

1. Den Daten Translation Lookside Buffer (DTLB)
2. Den Instruction Translation Lookaside Buffer (ITLB)

Der ITLB arbeitet mit der Instruktion Fetch Engine zusammen, und übersetzt Adressierungen in den Programmspeicher.
Der DTLB übersetzt virtuelle Speicheradressen für den Datenspeicher.

Um die Speicherzugriffe weiter zu beschleunigen, stehen für die TLBs erweiterte Hashtable zur Verfügung. Die Virtual Hash Page Table (VHPT) wird nach einer erfolglosen Suche im TLB nach der passenden physikalischen Adresse befragt.

3.6 Die Gleitkomma Berechnung

Die 128 FP Register mit einer Datenbreite von 82 Bit sind gegenüber IA-32 um 2 Bit erweitert. Für diese 82 Bit hat Intel ein neues Gleitkommaspeicherformat entwickelt, das gleich 128 Bit im Speicher ablegen kann. IA-64 unterstützt das Single (SP) und Double Precision (DP) Format, sowie Single Instruction Multiple Data (SIMD).

3.6.1 Berechnungsmodi

Logarithmus, Exponential, Tangens oder Sinusfunktionen sind im Instruktion Set von IA-64 nicht definiert. Der Compiler muss diese Funktionen bei Bedarf aus Softwarebibliotheken bereitstellen. Ebenso verhält es sich mit der Integer Division, die ebenfalls in der Floating Point Unit ausgeführt wird.

Durch die rotierenden FP Register ist es möglich, Divisionen pipelined auszuführen. Diese Verfahren dauert in der Throughput optimierten Version im Mittel 5 Takte. Andere Prozessoren kommen auf Werte um 29 Takte
(Pentium III).

Für wissenschaftliche Anwendungen steht der Multiply Add (MAC) Befehl zur Verfügung. Mit dem MAC Befehl können pro Takt und Gleitkommaeinheit zwei Double Precision Operationen ausgeführt werden: Multiplikation und Addition.

3.7 Branch Prediction

Die Branch Prediction – Sprungvorhersagen – sollen bedingte Programmverzweigungen vermeiden, da diese große Verzögerungen im Programmablauf hervorrufen. Je besser Sprungvorhersagen getroffen werden können, desto störungsfreier und schneller können die Befehle in der Instruktionspipeline abgearbeitet werden. Falsche Sprungvorhersagen erzeugen in Systemen mit langen Instruktionspipelines (z.B. 10 Zyklen), eine durchschnittliche Verzögerung von ca. 3 Zyklen pro Instruktion (30% * 10 Zyklen).

Beispiel einer Branch Prediction

Pseudo Code:
```
if(r1)
      r2 = r3 + r4
else
      r7 = r6 – r5
```

Durch den Einsatz von Sprung Registern (BR) und Predication Registern (PR) entsteht aus dem o.a. Code folgender ASM 64 Code:

ASM64 Code:

```
            cmp.eq     p1, p2 = r1, r0
(p2)        br.cond    else_clause
            add r2 = r3, r4
            br.endif
else_clause:
            sub r7 = r6, r5
endif:
```

cmp.eq vergleicht die Register r1 und r0. Wenn sie gleich sind, wird das Predication Register p1 auf eins gesetzt, Wenn r1 ungleich r0, dann wird p2 true.

br.cond führt eine Sprungverzweigung an die gekennzeichnete Stelle „else_clause“ aus, wenn das Register p2 auf 1 gesetzt ist.

Die Befehle add und sub addieren/ subtrahieren Registerinhalte voneinander.

Dieses Programmbeispiel ist nicht optimiert, da die Sprungverzweigung im false Fall ausgeführt wird.

Optimierter Code:

```
     cmp.eq     p1, p2 = r1, r0
(p1)       add r2 = r3, r4
(p2)       sub r7 = r6, r5
```

Jetzt wird die Anweisung ausgeführt, die durch das Predication Register angezeigt wird. Auf die Sprunganweisung kann hierdurch verzichtet werden.

3.8 Spekulationen

Wenn innerhalb eines Programms Daten/ Kontrollfluss Abhängigkeiten auftreten, besteht für den Prozessor die Möglichkeit, die benötigten Informationen vorab in den Cache bzw. in die Register zu laden. Diese spekulativ geladenen Daten werden speziell gekennzeichnet, und deren Adressen in einem Advanced Load Adress Table (ALAT) verwaltet. Wenn eine Adresse, die im ALAT hinterlegt ist, durch eine weitere Instruktion beschrieben wird, müssen die spekulativ geladenen Daten erneuert werden.

3.8.1 Control Speculation

Control Speculation wird angewendet, um Daten z.B. vor einer Sprunganweisung spekulativ in Register/ Cache zu laden.

Beispiel für Control Speculation

Pseudo Code ohne Control Specualtion:

```
if (a>b)   load(ld_addr1,target1)
else load(ld_address2,target2)
```

Pseudo Code inclusive Control Speculation:

```
sload(ld_addr1,target1)
sload(ld_addr2,target2)
.
./* weitere Instruktionen, die u.a. traget1, target2 verwenden
.
if(a>b)    scheck(target1,recovery_addr1)
else scheck(target2,recovery_addr1)
```

Beide load Anweisungen werden spekulativ ausgeführt, und als solche durch das s vor dem load Befehl gekennzeichnet. Wenn die Bedingung der if Schleife wahr ist, wird geprüft, ob die spekulativ geladenen Daten noch gültig sind. Wenn das nicht der Fall ist werden die Daten über die Recovery Adresse neu geladen. Die Recovery Adresse wird im ALAT verwaltet.
Im beschriebenen Beispiel ist der zweite spekulative Datensatz unnötig geladen worden, da er nicht gebraucht wird.

3.8.2 Data Speculation

Um Speicherkonflikte zu vermeiden, kann Data Speculation verwendet werden.

Beispiel für Data Specualtion

Pseudo Code ohne Data Speculation:

```
store(st_addr,data)
load(ld_addr,target)
use(target)
```

Zur Compiler Zeit können noch keinen Abhängigkeiten zwischen den Lade und Speicherstellen der Instruktionen festgestellt werden. Durch einen „advanced load" kann der load Befehl vor dem store Befehl ausgeführt werden. Advanced Loads sind immer spekulativ.

Pseudo Code mit Data Speculation:

```
//Advanced Load gekennzeichnet durch das a vor dem load-Befehl
aload(ld_add,target)

//andere Operationen des Programms

store(st_addr,data)
acheck(target,recovery_addr)
use(target)
```

acheck prüft hierbei die Gültigkeit des spekulativ durch aload geladenen Wertes.

4. Zusammenfassung

IA-64 unterstützt durch eine Vielzahl von Funktionen die parallele Bearbeitung von Daten. Die Architektur ist genau auf diese Ziel ausgerichtet und wird dadurch den zukünftigen Anforderungen an Prozessoren gerecht. Der Intel Itanium stellt eine erste Implementierung von IA 64 dar. Die Roadmap von Intel sieht weitere IA-64 Prozessoren mit erweiteter Hardware und höheren Taktraten vor. Ausschlaggebend für den Erfolg des Konzeptes ist meiner Ansicht nach die effiziente Aufgabenteilung zwischen Hardware und Complier. Je ökonomischer der Compiler den Code parallelisiert, desto schneller arbeitet die Hardware. Und bei der Hardware kann im Vergleich zum Itanium noch einiges getan werden.

Abbildungsverzeichnis

Abbildung 1: sequentielle Befehlsausführung 5
Abbildung 2: Instruction Pipelining 5
Abbildung 3: Das EPIC Design 6
Abbildung 4: EPIC Befehlsbündel 7
Abbildung 5: Aufbau der virtuellen Adresse 10
Abbildung 6: Prinzip der virtuellen Adressauflösung 11

Literaturverzeichnis

[1] Flik, Liebig: *Mikroprozessortechnik*. 5. Aufl. Berlin u.a.: Springer, 1998

[2] Stiller, A.: *Architektur für echte Programmierer: IA-64,EPIC.Itanium*. c´t. 2001, Nr. 13

[3] Intel Cooperation: *Intel IA-64 Software Developer´s Manual: Volume 1-3* Revision 1.1, July 2001

[4] Sharangpani,H.: *Intel Itanium Processor Microarchitecture Overview*. Intel Microprozessor Forum, Oktober 1999