Компьютеры с явным параллелизмом команд

1. Введение и терминология
2. Краткая характеристика Intel Itanium архитектуры
3. Принципы, лежащие в основе Itanium архитектуры

Знать*:*

Краткая характеристика вычислительных систем с явным параллелизмом команд.

Архитектура Intel Itanium – основные принципы:

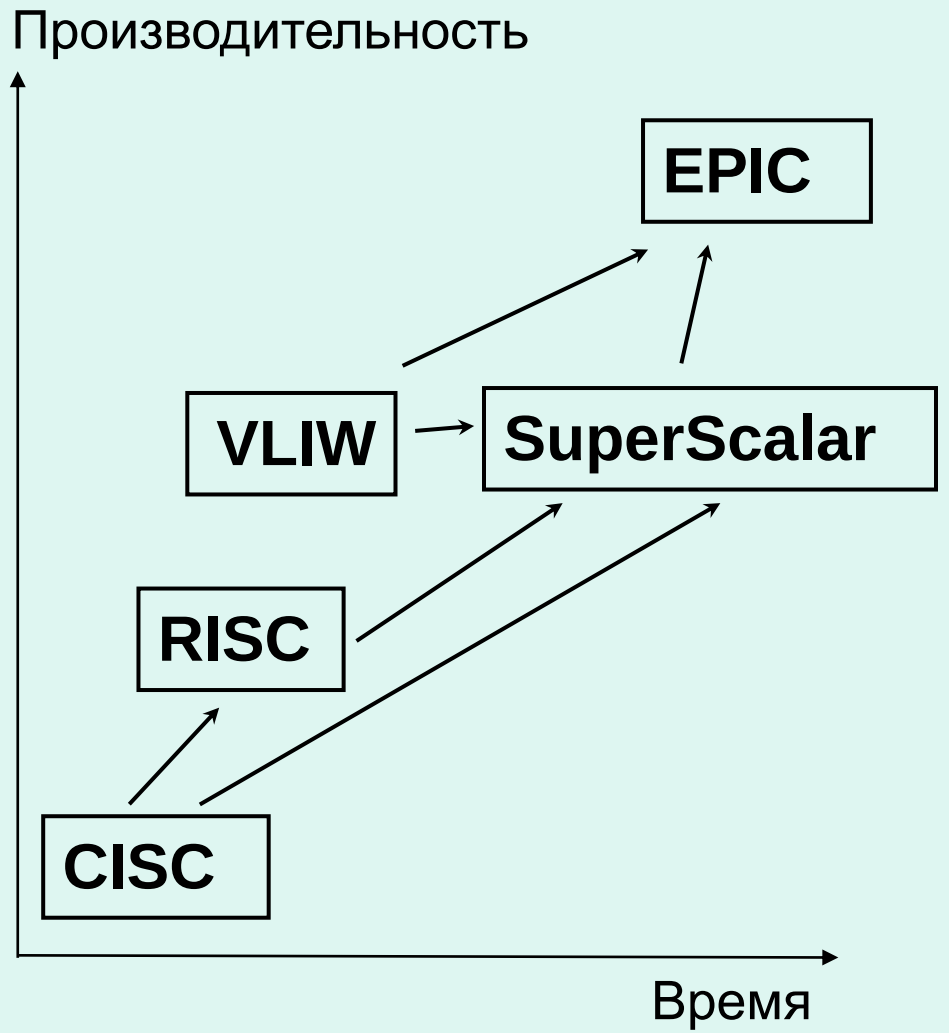
- спекуляция по управлению,
- спекуляция по данным,
- выполнение команд "под предикатами",
- использование регистрового стека при вызове процедур,
- вращение регистров.

Литература:

- Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем. Учебник для вузов. – СПб.: Питер, 2004. – 668 с. (с. 413-446)

1. Введение и терминология

Фирма Intel, первая выпустившая процессор с явным параллелизмом команд (EPIC), позиционировала его в координатах "производительность" – "время" (время выпуска) так, как показано на рисунке.



EPIC - Explicitly Parallel Instruction Computer

Терминология:

- **Instruction Set Architecture (ISA)** – (архитектура системы команд) – определяет применение и ресурсы системного уровня. Эти ресурсы включают команды и регистры.
- **Itanium Architecture** – архитектура **Itanium** – новая **ISA** с возможностями 64-битных команд, с новыми улучшенными рабочими параметрами, и с поддержкой системы команд IA-32 архитектуры. Иначе эта архитектура называется **IA-64** архитектурой.
- **IA-32 Architecture** – 32-битная и 16-битная Intel архитектура, как она определена в фирменном описании.

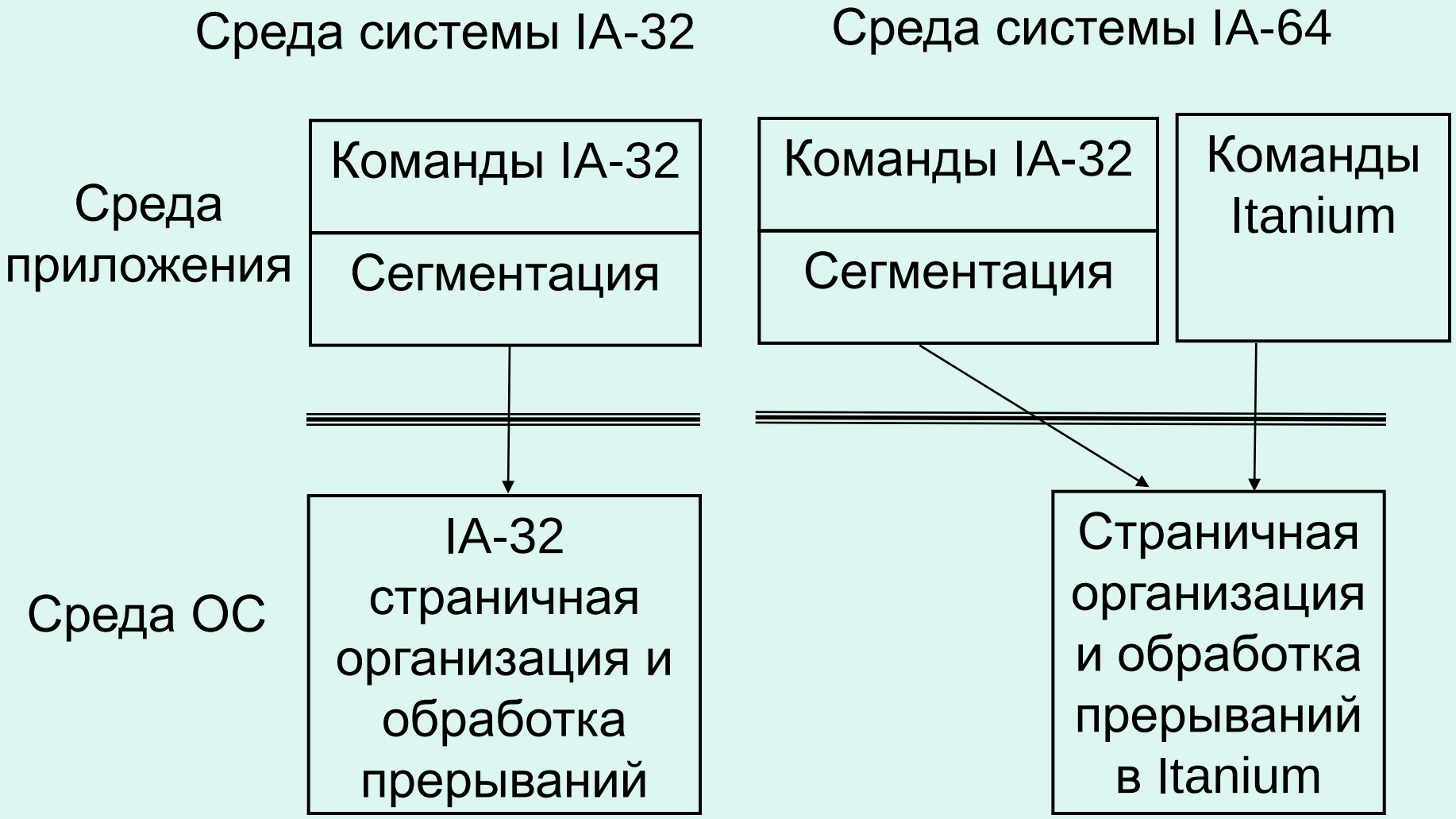
2. Краткая характеристика Intel Itanium архитектуры

- Явный параллелизм.
  - Механизм для взаимодействия (совмещения) компилятора и процессора.
  - Огромные ресурсы для достижения параллелизма на уровне команд.
  - 128 регистров для целочисленных данных и 128 регистров для данных с плавающей запятой, 64 однобитных предикатных регистра, 8 регистров для ветвлений.
  - Поддержка многих исполнительных блоков и портов памяти.

- Особенности, расширяющие параллелизм на уровне команд.
  - Спекулятивное выполнение команд.
  - Предикаты (которые позволяют удалить ветвления).
  - Программная конвейеризация циклов (снизу до верху).
  - Предсказание переходов, минимизирующее затраты времени на ветвления.

- Ориентация на повышение эффективности выполнения программ.
  - Специальная поддержка модульности программного обеспечения.
  - Высокопроизводительная архитектура средств обработки данных с плавающей запятой, ориентированная на работу с регистрами, а не со стеком.
  - Специфические мультимедиа команды, которые обращаются к общему регистру как к восьми 8-разрядным, четырем 16-разрядным или двум 32-разрядным регистрам и обрабатывают каждый элемент регистра параллельно, независимо от других.

Операционные системы и приложения



3. Принципы, лежащие в основе Itanium архитектуры

3.1. Параллелизм на уровне команд.

3.2. Информирование процессора компилятором (поиск зависимостей между командами осуществляет компилятор, а не процессор).

3.3. Спекуляция.

- 3.3.1. Спекуляция по управлению.
- 3.3.2. Спекуляция по данным.

3.4. Предикаты.

3.5. Регистровый стек.

3.6. Ветвления.

3.7. Регистровое вращение.

3.1. Параллелизм на уровне команд

- Параллелизм на уровне команд (инструкций) (ILP) – это способность выполнять множество команд в одно и тоже время.
- Архитектура Itanium позволяет выдавать независимые команды в связках (три команды в связке) для параллельного выполнения и может выдавать множество связок за такт.
- Благодаря большому числу параллельных ресурсов, таких как большой регистровый файл и множество исполнительных блоков архитектура Itanium позволяет компилятору управлять работой по развитию и планированию параллельных нитей вычислений.

3.2. Информирование процессора компилятором

- Предусмотрены: шаблоны команд, предсказатели переходов и предсказатели для кэш-памяти, позволяющие компилятору сообщать информацию, полученную во время компиляции, процессору.
- Скомпилированный код позволяет управлять аппаратурой процессора, используя информацию, получаемую во время исполнения. Затраты на переходы минимизируются благодаря поступлению упреждающей информации о фактическом направлении переходов.
- При каждом чтении и записи в память используется 2-битное поле кэш предсказателя, в котором компилятор кодирует свое предсказание о пространственно-временной локальности области памяти, используемой для доступа. С помощью этой информации определяется местоположение кэш строк в иерархии кэш-памяти для повышения эффективности работы с памятью.

3.3. Спекуляция

- Используется два вида спекуляции: по управлению и по данным. В обоих видах спекуляции компилятор выявляет параллелизм на уровне команд, выдавая операцию заранее. Компилятор будет выдавать операцию спекулятивно, если действительно будет уверенность, что спекуляция выгодна.
- Спекуляция является одним из первичных механизмов для компилятора, использующего статистический параллелизм на уровне команд, чтобы организовать последующую работу с перекрытием (совмещением).

3.3.1. Спекуляция по управлению

- Спекуляция по управлению представляет собой выполнение операции, прежде чем будет вычислено значение условия перехода к ней.
- Рассмотрим следующую последовательность команд:

if (a>b) load (ld_addr1, target1)

else load (ld_addr2, target2)

- Если команда load (ld_addr1, target1) будет выполнена, прежде чем определится значение условия (a>b), то команда будет управляться спекулятивно связанным с ней управляющим условием (a>b). При нормальном выполнении команд, команда load (ld_addr1, target1) может быть выполнена, а может быть, и не выполнена.
- Если новая спекулятивно управляемая загрузка вызовет исключение, то исключение будет обслужено только при истинном значении условия (a>b). Когда компилятор использует спекуляцию по управлению, он вводит проверочную команду (scheck) в первоначальном местоположении контролируемой операции.

Пример спекуляции по управлению

- Последовательность команд,

if (a>b) load (ld_addr1, target1)

else load (ld_addr2, target2)

транслируется следующим образом:

/* от критического пути */

sload (ld_addr1, target1)

sload (ld_addr2, target2)

/* другие операции, включая использование target1/ target2 */

if (a>b) scheck (target1, recovery_addr1)

else scheck (target2, recovery_addr2)

[recovery – восстановление]

3.3.2. Спекуляция по данным

- Спекуляция по данным представляет собой выполнение загрузки из памяти перед записью ("чтение перед записью", "опережающая загрузка").
- Процесс определения во время компиляции отношения между адресами памяти (проверка на совпадение адресов) называется устранением противоречий.
- Если адреса памяти совпадут во время исполнения, то спекулятивная загрузка данных, идущая перед записью, даст величину, отличную от обычной загрузки, следующей после записи.
- Поэтому аналогично спекуляции по управлению, когда компилятор обрабатывает загрузку со спекуляцией по данным, он вводит проверочную команду в первоначальном местоположении команды загрузки.
- Проверочная команда выясняет, имело ли место совпадение адресов, и, если необходимо, организует повторение загрузки.

Пример спекуляции по данным

- Рассмотрим следующую последовательность команд:

store (st_addr, data)

load (ld_addr, target)

use (target)

- Последовательность команд транслируется следующим образом:

/* от критического пути */

aload (ld_addr, target)

/* другие операции, включая использование target */

store (st_addr, data)

acheck (target, recovery_addr)

use (target)

3.4. Предикаты

- Предикаты являются условиями выполнения команд. В архитектуре Itanium для реализации ветвлений используются команды под предикатами.
- Иллюстрация команды без предиката: r1=r2+r3.
- После введения предиката она примет следующую форму: if (p5) r1=r2+r3.
- В этом примере p5 – управляющий предикат, который решает, будет или нет выполняться команда, и обновляет состояние.
- Если предикат имеет значение "истина", то команда изменяет состояние. Иначе она обычно ведет себя как команда "NOP". Значения предикатам устанавливаются командами сравнения.

3.4.1. Пример программы с предикатами

- Рассмотрим следующую последовательность команд:

```
if (a>b) c=c+1
else d=d*e+f
```

- Ветвления по условию (a>b) можно избежать путем преобразования приведенной программы в программу с предикатами:

```
pT, pF = compare (a>b)
if (pT) c=c+1
if (pF) d=d*e+f
```

- Предикат pT устанавливается в единицу, если условие истинно и в ноль, если – ложно. Предикат pF представляет собой инверсию предиката pT.

3.4.2. Преобразования зависимости по управлению в зависимость по данным

- Использование предикатов позволяет избежать ветвлений и упрощает компилятору оптимизацию кода путем преобразования зависимости по управлению в зависимость по данным.
- Управляемая зависимость команд c=c+1 и d=d*e+f от ветвления по условию (a>b) была преобразована в зависимость по данным от результата сравнения (a>b) через предикаты pT и pF (ветвление исключено).
- Дополнительная выгода заключается в том, что компилятор может теперь планировать параллельное выполнение команд под предикатами pT и pF.
- Достоинством также является наличие нескольких типов команд сравнения, которые определяют значения предикатов различными способами, включая безусловные сравнения и параллельные сравнения.

3.5. Регистровый стек

- В архитектуре Itanium исключается необходимость в сохранении и восстановлении регистров при взаимодействии процедур при вызове и возврате благодаря управляемому компилятором переименованию регистров.
- В точке вызова новый фрейм регистров становится доступным вызываемой процедуре без необходимости перераспределения и заполнения регистров (вызывающей или вызываемой процедурой).
- Доступ к регистрам организуется путем переименования идентификаторов виртуальных регистров, указанных в командах, в физические регистры (в номера физических регистров) с помощью базового регистра (логический номер регистра из команды, складывается с базовым адресом).

3.5.1. Распределение регистров регистрового стека

- Вызываемая процедура свободно может использовать имеющиеся в распоряжении регистры без сохранения и восстановления регистров вызываемой процедуры.
- Вызываемая процедура распределяет указанные в командах регистры в предположении, что число доступных физических регистров достаточно.
- Если необходимое число регистров недоступно (регистровый стек переполнен), то распределение приостанавливается и выполняется освобождение регистров за счет регистров вызывающей процедуры, которое продолжается до тех пор, пока не освободится необходимое число регистров.

3.5.2. Восстановление содержимого регистров регистрового стека

- В точке возврата состояние базового регистра восстанавливается – в него записывается значение, которое он имел до выделения регистров вызываемой процедуре.
- При этом может оказаться, что содержимое некоторых регистров вызывающей процедуры было ранее аппаратно сохранено и еще не восстановлено. В этом случае (отрицательное переполнение стека) возврат приостанавливается до тех пор, пока процессор не восстановит содержимое соответствующего числа регистров вызывающей процедуры.
- Аппаратные средства процессора могут использовать информацию о складывающемся распределении регистров стека и производить сохранение содержимого регистров в памяти и восстановление содержимого регистров из памяти, используя наиболее удобные моменты независимо от вызывающих и вызываемых процедур.

3.6. Ветвления

- В дополнение к устранению ветвлений с помощью предикатов, несколько механизмов обеспечивают уменьшение доли ложных предсказаний ветвлений и затраты времени на остающиеся ложно предсказанные ветвления.
- Эти механизмы обеспечивают передачу информации об условиях ветвления от компилятора процессору.
- Для передачи заблаговременных указаний о целевом адресе (адресе перехода) и локализации ветвления могут быть использованы команды предсказания ветвлений. Компилятор будет пытаться показать, будет ли предсказываемое ветвление статическим или динамическим.
- Процессор может использовать эту информацию для инициализации предсказываемых структур ветвлений, обладающих хорошим качеством предсказания, даже если первое время при ветвлениях возникают трудности.
- Это дает выигрыш для безусловных переходов или в ситуациях, когда компилятор имеет информацию о подобных развитиях вычислительного процесса.

Условные ветвления и циклы

- При условных ветвлениях для сохранения адреса перехода используется регистр ветвления.
- Команды предсказания ветвлений обеспечивают указание того, какой регистр будет использован в момент ветвления, а сам адрес перехода может быть вычислен заранее.
- Команды предсказания ветвлений могут также сигнализировать об условных ветвлениях при возврате из процедур, повышая эффективность использования предсказываемых структур, связанных с работой стека при вызовах и возвратах из процедур.
- Специальные заключительные ветвления в циклах обеспечивают ускорение выполнения простых и вложенных циклов.
- Эти ветвления и связанные с ними команды предсказания ветвлений обеспечивают процессор информацией, которая позволяет произвести безупречное предсказание окончания цикла, таким образом исключаются дорогостоящие ложные предсказания и уменьшается время выполнения циклов.

3.7. Регистровое вращение

- Циклы аналогичны аппаратурным конвейерам из функциональных блоков с учетом того, что следующая итерация цикла стартует, прежде чем завершится предыдущая. Итерация разделяется на стадии подобно стадиям в работе конвейера.
- Компилятор получает возможность представить выполнение итераций цикла скорее в параллельной, чем в последовательной форме.
- Одновременное выполнение многочисленных итераций традиционно требует разворачивания цикла и программного переименования регистров.
- В архитектуре Itanium предусмотрено переименование регистров, которое обеспечивает каждой итерации свой собственный набор регистров и исключает необходимость в разворачивании цикла.
- Этот способ переименования регистров назван регистровым вращением (вращением регистров).
- В результате программной конвейеризации циклов значительно сокращается время их выполнения.