**Стукало Виктор Александрович, к.т.н. 905-854-1294**

**stukalo.v@mail.ru**

**Аппаратные средства вычислительной техники**

Эндрю Таненбаум – «Архитектура компьютера».

**Процессоры**

**Архитектура системы команд. Классификация процессоров CISC и RISC.**

Термин архитектура системы употребляется в двух смыслах – узком и широком.

В узком смысле под архитектурой понимается совмещение с определенным набором команд и это наиболее частое употребление термина. Архитектура набора команд является границей между аппаратурой и программным обеспечением.

В широком смысле архитектура означает организацию вычислительной системы, включающую процессор, систему памяти, структуру системной шины, организацию ввода-вывода и т.п.

В мире существуют 2 основных архитектуры системы команд:

1. CISC (Complete Instruction Set Computer)
2. RISC (Reduced Instruction Set Computer)

Основоположником CISC архитектуры считается IBM с базовой архитектурой system 360. Ядро этой системы используется с 1960. Используется в современных операционных системах. Лидером в разработке микропроцессоров CISC считается компания Intel. Их линейка x86(x64) как раз относится к системам с полным набором команд (CISC). Для CISC процессоров характерно небольшое число регистров общего назначения (не больше 16).

**ДЗ: Повторить назначения регистров процессора фирмы Intel. Повторить систему команд.**

CISC процессоры характеризуются:

1. Большим количеством машинных команд (некоторые из них семантически нагружены аналогично операторам высокоуровневых языков программирования).
2. Большое количество методов адресации

**ДЗ: Повторить способы адресации микропроцессора Intel.**

1. Большое количество форматов команд

**ДЗ: Повторить форматы команд микропроцессора Intel.**

1. Преобладание двухадресного формата команд (поле команды и 2 поля адреса, один источника, другой приемника операции, результат заменяет первый адрес)
2. Наличие команд обработки типа регистров памяти

Альтернативой CISC является RISC архитектура. До недавнего времени считалось, что серверные процессоры должны строиться по RISC архитектуре. Современное понятие RISC сформировалось на базе трех исследовательских проектов. На интервале 2 года IBM выпустила процессор 801, университет Беркли выпустил процессор RISC, а Стэндфордский университет выпустил свой процессор. Машины имели много общего. В этой архитектуре команды обработки отделены от команд работы с памятью. Упор делался на конвейерную обработку команд. Система команд разрабатывалась таким образом, чтобы исполнение команды занимало один такт машинного времени. В CISC архитектуре огромное количество команд выполняется за несколько сотен тактов. С целью повышения производительности логика исполнения команд была аппаратной, а не микропрограммной. Для упрощения логики декодирования команд использовались команды фиксированных длинны и формата. Процессоры RISC архитектуры обладают большим регистровым файлов (набором регистров). От 32-х регистров. Это позволяет большему объему данных храниться в процессоре. Меньше обращаться к памяти. Упрощает работу компилятора по распределению регистров под переменные. Для обработки используется трехадресные команды. Что позволяет хранить большее число данных без перезагрузки.

Пример **A + B = C**

CISC

LD R1, A: [A] -> R1

ADD R1, B: [R1] + [B] -> R1

ST C, R1: [R1] -> C

RISC

LD R1, A: [A] -> R1

LD R2, B: [B] -> R2

ADD R3, R2, R1: [R1] + [R2] -> R3

ST C, R3: [R3] -> C

**Организация конвейера**

Ступени конвейера

1. IF – instruction fetch (выборка команды)
2. ID – instruction decode (декодирование команды)
3. Exe – executive (исполнение команды)
4. Mem – memory (обращение к памяти)
5. WB – write back (обратная запись)

Диаграмма конвейера (#1).

Отчет

1. Титульный лист
2. Постановка задачи
3. Описание алгоритма решения задачи (исходный текст с комментариями)
4. Таблица с результатами экспериментов
5. Заключение
   1. Констатация фактов (что наблюдал, какие данные получил)
   2. Почему получил именно такие данные, чем это вызвано, какими эффектами, работает или нет теория, изложенная в описании лабораторной работы

**Организация 8-bit процессора**

В команде нужно указать:

1. Код операции
2. Адрес операнда 1
3. Адрес операнда 2
4. Адрес результата
5. Адрес следующей команды

Сделан вывод, что адрес следующей команды можно убрать, потому как выполняется обычно следующая команда. Вместо него вводится (Program Counter) программный счетчик, который инкрементируется при выполнении каждой команды и указывает на следующую команду.

Add A, B, C, D – адрес следующей команды не нужен (используется PC)

Add A, B, C – адрес результата не нужен (используется первый операнд для вывода результата)

Add A, B – адрес первого операнда не нужен (используется аккумулятор)

Add B – получилась команда с одним операндом

Add Rn, B – получаем команду с операндом, указывающим на регистр общего назначения (с коротким адресом) и операндом со вторым значением

Вычисление по элементарной формуле:

B + C = D

Исходное положение процессора: PC {1000}

1 команда 1 фаза:

PC{1000/1001}

AR{1000}

MD{LD}

IR{LD}

1 команда 2 фаза:

PC{1001/1002}

AR{1001}

MD{LoB}

IR{LD||LoB}

1 команда 3 фаза:

PC{1002/1003}

AR{1002}

MD{HiB}

IR{LD|B}

1 команда 4 фаза:

PC{1003}

AR{B}

MD{100}

IR{LD|B}

1 команда 5 фаза:

MD{100}

A{100}

2 команда 1 фаза:

PC{1003/1004}

AR{1003}

MD{ADD}

LR{ADD}

2 команда 2 фаза:

PC{1004/1005}

AR{1004}

MD{LoC}

LR{ADD|LoC}

2 команда 3 фаза:

PC{1005/1006}

AR{1005}

MD{HiC}

LR{ADD|C}

2 команда 4 фаза:

PC{1006}

AR{B}

MD{200}

IR{ADD}

2 команда 5 фаза:

A{100/200}

3 команда 1 фаза:

PC{1006/1007}

AR{1006}

MD{ST}

IR{ST}

3 команда 2 фаза:

PC{1007/1008}

AR{1007}

MD{LoD}

IR{ST|LoD}

3 команда 3 фаза:

PC{1008/1009}

AR{1008}

MD{HiD}

IR{ST|D}

3 команда 4 фаза:

PC{1008}

AR{D}

MD{255}

{B+C}

255

**Системный таймер**

Раньше это была схема intel-8083 (PIT – programmable interval timer).

В составе таймера имеется 4 программно-доступных регистра, каждый имеет свой адрес:

1. 40h – регистр делитель
2. 41h – регистр делитель
3. 42h – регистр делитель звукового канала
4. 43h – регистр управляющего слова

61h – 8bit порт, многофункциональный. Младшие 2 бита этого порта используются, один для подачи сигнала запуска таймера, другой открывает электронный ключ. Остальные биты трогать не рекомендуется.

43h – описание битов:

1. binary coded decimal / binary.
2. Режим работы канала
3. Режим работы канала
4. Режим работы канала (011)
5. 1
6. 1
7. Канал
8. Канал (00, 01, 10)

B6 (управляющее слово для лабораторной)

Делитель = Входная величина/выходную величину(задается мелодией)

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования известным под общим названием совмещение операций, при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной операции. Этот общий метод включает 2 понятия:

1. Параллелизм
2. Конвейеризация

У них много общего, зачастую их трудно различить на практике, но эти термины отражают два совершенно различных подхода.

При параллелизме совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур осуществляющих решение различных частей задачи.

Конвейеризация – в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, ступени и выделение для каждой из них отдельного блока аппаратуры. Обработку любой машинной команды можно разделить на несколько этапов, организовав передачу данных от одного этапа данных к следующему. При этом конвейеризацию можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняется несколько команд. Конвейерная обработка такого рода широко применяется во всех современных процессорах.

Для иллюстрации основных принципов построения процессоров будем использовать простейшую пяти ступенчатую архитектуру, содержащую 32 регистра общего назначения от до , 32 регистра плавающей точки от до и программный счетчик. Набор команд процессора включает типичные арифметические и логические операции, операции плавающей точки, пересылки данных, управления и разного рода системные команды. В арифметических командах используется трех-адресный формат. Для обращения к памяти используются операции загрузки и записи содержимого регистров в память. Общее время выполнения команды на таком конвейере будет составлять 5 тактов. В каждом такте аппаратура выполняет в совмещенном режиме 5 различных команд.

При реализации конвейерной обработки возникает ситуация, препятствующая выполнению очередной команды из потока команд в предназначенном для неё такте. Такие ситуации называются конфликтами. Конфликты снижают реальную производительность конвейера, которая могла бы быть достигнута в идеальном случае. Существуют 3 класса конфликтов:

1. Структурные конфликты (которые возникают из-за конфликтов по ресурсов, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением)
2. Конфликты по данным (возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды)
3. Конфликты по управлению (которые возникают при конвейеризации команд переходов и других команд, изменяющих значение программного счетчика)

Конфликты приводят к необходимости приостановки выполнения команд. В простейших конвейерах, если останавливается какая-либо команда, то все следующие за ней также приостанавливаются. Предшествующие команды продолжают выполнение, но во время приостановки не выбирается ни одна новая команда.

**Способы минимизации конфликтов.**

Структурный конфликт.

Совмещенный режим выполнения команд требует конвейеризации функциональных устройств и дублирования ресурсов, для разрешения всех возможных комбинаций команд в конвейере. Если какая-либо комбинация команд не может быть принята из-за конфликта по ресурсам, то имеется структурный конфликт. Время работы такого устройства может составлять несколько тактов синхронизации конвейера. В этом случае команды, использующие данное функциональное устройство, не могут поступать на него в каждом такте. Допускается наличие структурных конфликтов по двум причинам:

1. Снижение стоимости
2. Уменьшение задержки устройства

Конвейеризация всех функциональных устройств может оказаться слишком дорогой. Машины допускающие 2 обращения к памяти в одном такте должны иметь удвоенную пропускную способность памяти. Однако, если структурные конфликты не будут возникать слишком часто, то может быть и не стоит платить за то, чтобы их обойти.

Конвейер может иметь один порт для записи в память, но если необходимо 2 записи – возникает структурный конфликт. При этом, например, исполнение уже выбранной команды продолжается, операнд записывается в память.

Для разрешения конфликтов по данным используются так называемые пути обхода.

Конфликты по управлению.

Смысл перехода с задержкой перехода длиной n:

Команда условного перехода

Команда 1

Команда 2

…

команда n

точка перехода

Задача перехода заключается в том, чтобы сделать команду следующей за командой перехода действительным и полезными.

Конфликты по данным не устраняемые с помощью аппаратуры обхода.

Современные компиляторы используют технику планирования команд для улучшения производительности конвейера. В простейшем алгоритме компилятор просто планирует распределение команд в одном и том же базовом блоке. Базовый блок представляет собой линейный участок последовательности программного кода, в котором отсутствуют команды перехода. За исключением начала и конца участка. Переходы внутрь участка, тоже должны отсутствовать. Планирование такой последовательности команд осуществляется достаточно просто. Компилятор строит граф зависимостей этих команд и упорядочивает их так, чтобы минимизировать остановки конвейера. Для простых конвейеров стратегия планирования на основе базовых блоков удовлетворительна. Однако при интенсивной конвейеризации и росте задержек конвейера требуются более сложные алгоритмы планирования. Эти методы получили общее название методов динамической оптимизации. Часто используется термин out of order execution, т.е. неупорядоченное выполнение. А также метод переименования регистров.

**Параллелизм уровня команд**

Зависимости и конфликты по данным

Потенциальный параллелизм заложенный в последовательности команд называется параллелизмом уровня команд или ILP (Instruction Level Parallelism). Степень параллелизма доступная внутри одного базового блока очень мала. Например частота перехода в целых программах составляет около 16%. Это означает, что между двумя переходами выполняется 5 команд, а каждая 6 – переход. Поскольку эти 5 команд возможно взаимозависимые, то степень перекрытия, которую можно использовать внутри базового блока, возможно, будет меньше чем 5. Чтобы получить существенное улучшение производительности необходимо использовать параллелизм уровня команд одновременно для нескольких базовых блоков. Самый простой способ увеличения степени параллелизма, доступного на уровне команд – использование параллелизма между итерациями цикла. Этот тип параллелизма часто называется параллелизмом уровня итеративного цикла. Например, цикл выполняющие сложение двух векторов является полностью параллельным. Каждая итерация цикла может перекрываться любой другой итерацией. Хотя внутри каждый итерации практическая возможность перекрытия небольшая. Имеется несколько методов для превращения такого параллелизма уровня цикла, в параллелизм уровня команд. Методы основаны на разворачивании цикла, статистически используя компилятор, либо динамически, с помощью аппаратуры.

Основы планирования загрузки конвейеров и разворачивания циклов

Для поддержания максимальной загрузки конвейера зависимые команды должны отделяться от исходных на расстояние в тактах равное задержкам конвейера для этих исходных команд. Способность компилятора выполнять подобное планирование определяется способностями функциональных устройств. Во всех дальнейших примерах будем предполагать задержки конвейера указанные в таблице.

|  |  |  |
| --- | --- | --- |
| Команда вырабатывающая результат | Команда использующая результат | Задержка в тактах |
| Операция АЛУ с плавающей точкой | Др. оп. АЛУ с ПТ | 3 |
| Оп. АЛУ с ПТ | Запись двойного слова | 2 |
| Загрузка двойного слова | Др. оп. АЛУ с ПТ | 1 |
| Загрузка двойного слова | Запись двойного слова | 0 |
| Оп. условного перехода | Др. оп. УП | 1 |

Предположим, что в регистре R1 вектор, А в регистре f2 скалярная величина, которая добавляется к каждому элементу вектора.

Loop:

LD F0, 0(R1) ; загрузка в F0 элемента вектора

1

ADDD F4, F0, F2 ; добавление скалярной величины к элементу вектора

2

SD 0(R1), F4 ; запись в R1 нового значения

0

SUBI R1, R1, #8 ; пересчет указателя элемента вектора

0

BNEZ R1, Loop ; переходим на метку loop, если R1 = 0

1

9 тактов на результат итерации

Loop:

LD F0, 0(R1)

1

ADDD F4, F0, F2

0

SUBI R1, R1, #8

0

BNEZ R1, Loop ; задержанный переход

0

SD 8(R1), F4 ; размещается в слоте задержанного перехода и выполняется перехода

0

6 тактов на результат итерации

В примере за одну итерацию выполняется запись одного элемента вектора на каждые 6 тактов (после оптимизации цикла). Действительная работа по обработке элемента вектора занимает только 3 такта (загрузка, сложение, запись), остальные 3 такта составляют накладные расходы на выполнение цикла (пересчет указателя и условный переход, приостановка). Для устранения этих трех тактов нужно иметь больше операций в цикле относительно команд создающих накладные расходы. Один из основных методов является метод разворачивания циклов. Разворачивание циклов выполняется путем многократной репликации тела цикла и коррекции соответствующего кода конца цикла. Разворачивание циклов может также использоваться для улучшения планирования. В этом случае можно устранить задержку вызванную загрузкой элемента вектора. Для использования нескольких итераций потребуется большее количество регистров.

Представляем цикл развёрнутым четырехкратно, предполагаем, что R1 первоначально кратен четырем. Любые излишние вычисления устраняем, повторно не используем никакие регистры.

Loop:

LD F0, 0(R1)

1

ADDD F4, F0, F2

2

SD 0(R1), F4

0

LD F6, -8(R1)

1

ADDD F8, F6, F2

2

SD -8(R1), F8

0

LD F10, -16(R1)

1

ADDD F12, F10, F2

2

SD -16(R1), F12

0

LD F14, -24(R1)

1

ADDD F16, F14, F2

2

SD -24(R1), F16

0

SUBI R1, R1, #32

0

BNEZ R1, Loop

1

14 команд + 13 тактов = 27

27/4=6.75 такта на результат

Loop:

LD F0, 0(R1)

LD F6, -8(R1)

LD F10, -16(R1)

LD F14, -24(R1)

ADDD F4, F0, F2

ADDD F8, F6, F2

ADDD F12, F10, F2

ADDD F16, F14, F2

SD 0(R1), F4

SD -8(R1), F6

SD -16(R1), F10

SUBI R1, R1 #32

BNEZ R1, Loop

SD 8(R1), F16

14 тактов на все

14/4=3.5 такта на 1 результат

9 т/рез

после оптимизации

6 т/рез

после разворачивания

6.75 т/рез

после оптимизации разворачивания

3.5 т/рез

Одновременная выдача нескольких команд для выполнения и динамическое планирование

Методы минимизации приостановок конвейера из-за наличия в программах логических зависимостей по данным и по управлению, нацеленные на достижение идеального коэффициента CPI (Cycle-per-instruction). Идеальный CPI равен единице. Чтобы еще больше повысить производительность необходимо сделать CPI меньше единицы. Этого нельзя сделать, если в одном такте на выполнение выдается только одна команда. Следовательно, необходима параллельная выдача команд в каждом такте. Существует 2 типа подобного вида машин.

1. Супер скалярные машины
2. Машины с очень длинным машинным словом

Супер скалярные машины могут выдавать на исполнение в каждом такте переменное число команд. Работа их конвейера может планироваться как статически, с помощью компилятора, так и динамически с помощью средств оптимизации.

В отличие от супер скалярных машин, машины с длинным машинным словом выдают на исполнение фиксированные количество команд, которые сформатированы либо как одна большая команда, либо как пакет команд фиксированного формата. Планирование работы таких машин всегда осуществляется компилятором.

Супер скалярные машины используют параллелизм уровня команд, путем посылки нескольких команд из обычного потока команд в несколько функциональных устройств. Чтобы снять ограничение последовательного выполнения команд в этих машинах используются механизмы внеочередной выдачи и внеочередного завершения команд. Прогнозирование переходов. Кеши целевых адресов переходов. И условное, т.е. по предположению, выполнение команд. В типичной, супер скалярной машине, аппаратура может выдавать до 8 команд в одном такте. Обычно это должны быть независимые команды, удовлетворяющие некоторым ограничениям. Например, в каждом такте может выдавать не более одной команды обращения к памяти, если какая либо команда в потоке является зависимой, логически или не удовлетворяет критериям выдачи, на выполнение будут выданы только команды предшествующие данной, поэтому скорость выдачи команд в супер скалярных машинах является переменной, это и отличает их от машин с очень длинным машинным словом, в которых пакет команд, выполняемых одновременно, формирует компилятор, а аппаратура не принимает решений, относительно параллельной выдачи нескольких команд.

Первые супер скалярные компьютеры выдавали на выполнение 2 команды в одном такте. Одной из таких команд была загрузка регистров из памяти, запись регистра в память, команда перехода, операция целочисленного АЛУ. Другой командой могла быть команда с плавающей точкой. Параллельная выдача команд одной целочисленной и одной с плавающей точкой намного проще, чем выдача двух произвольных команд. В процессорах Pentium, например, применялся именно такой подход. Выдача двух команд в каждом такте требует одновременной выборки и декодирования по крайней мере 64 бит. Для упрощения декодирования команды можно располагать парами и выравнивать их по 64 битовым границам, в противном случае необходимо анализировать команды в процессе выборки и менять их местами в момент пересылки в ЦЧ устройство и ПТ. При этом возникают дополнительные требования к схемам обнаружения конфликтов. В любом случае вторая команда может выдаваться, если только на исполнение может быть выдана первая. Аппаратура обеспечивает выдачу второй команды, если условия выполнения первой не соблюдаются.

Чтобы спланировать цикл добавления скалярной величины, к вектору памяти для работы без задержек, необходимо развернуть его пятикратно.

|  |  |  |
| --- | --- | --- |
| № такта | Целочисленная команда | Команда с плавающей точкой |
| 1  2  3  4  5  6  7  8  9  10  11  12 | Loop:  LD F0, 0(R1)  LD F6, -8(R1)  LD F10, -16(R1)  LD F14, -24(R1)  LD F18, -32(R1)  SD 0(R1), F4  SD -8(R1), F8  SD -16(R1), F12  SD -24(R1), F16  SUBI R1, R1, #40  BNEZ R1, Loop  SD 8(R1), F20 | ADDD F4, F0, F2  ADDD F8, F6, F2  ADDD F12, F10, F2  ADDD F16, F14, F2  ADDD F20, F18, F2 |

12/5 = 2.4

2.4 на такт текущая скорость, в старом примере 3.5.

Команд ПТ недостаточно для полной загрузки конвейера ПТ, первоначальный, неразвернутый цикл, выполнялся со скоростью 6 тактов на результат, мы получили ускорение в 2.5 раза. Больше половины которого произошло за счет разворачивания цикла. Чистое разворачивание дало улучшение в 1.5 раза.

**Архитектура машин с длинным командным словом (VLIW – Very Long Instruction Word)**

VLIW-архитектура позволяет сократить объем оборудования и потенциально, чем большее количество команд выдается параллельно, тем больше эта экономия.

Супер скалярная машина, обеспечивающая выдачу двух команд одновременно, требует реализацию параллельного выполнения 2-х команд, 6 регистров, а так же динамического анализа возможности выдачи одной или двух команд и распределения этих команд по функциональным устройствам. Требования по объему аппаратуры для параллельной выдачи двух команд считаются умеренными. Увеличением количества выдаваемых параллельно команд для выполнения приводит к вырастанию сложности реализации из-за определения порядка следования команд и существующих между ними зависимостей.

Архитектура VLIW базируется на множестве изолированных друг от друга устройств. Вместо того, чтобы выдавать в эти устройства независимые команды, в таких машинах несколько операций упаковывается в одну очень длинную команду. При этом ответственность за выбор параллельно выдаваемых на выполнение операций полностью ложится на компилятор, а аппаратные средства необходимые для реализации супер скалярной обработки просто отсутствуют.

Рассмотрим работу цикла инкрементирования элементов вектора на VLIW машине в предположении, что одновременно могут выдаваться две операции обращения к памяти, 2 операции с плавающей точкой, одна целочисленная, либо одна команда перехода. Для устранения всех возможных приостановок конвейера цикл разворачивается семикратно.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Обращение 1 | Обращение 2 | Плавающая 1 | Плавающая 2 | Целое/переход |
| 1 | LD F0, 0(R1) | LD F6, -8(R1) |  |  |  |
| 2 | LD F10, -16(R1) | LD F14, -24(R1) |  |  |  |
| 3 | LD F18, -32(R1) | LD F22, -40(R1) | ADDD F4, F0, F2 | ADDD F8, F6, F2 |  |
| 4 | LD F26, -48(R1) |  | ADDD F12, F10, F2 | ADDD F16, F14, F2 |  |
| 5 |  |  | ADDD F20, F18, F2 | ADDD F24, F22, F2 |  |
| 6 | SD 0(R1), F4 | SD -8(R1), F8 | ADDD F28, F26, F2 |  |  |
| 7 | SD -16(R1), F12 | SD -24(R1), F16 |  |  | SUBI R1, R1, #56 |
| 8 | SD 24 (R1), F20 | SD 16 (R1), F24 |  |  | BNEZ R1, Loop |
| 9 | SD 8(R1), F28 |  |  |  |  |

Один проход по циклу осуществляется за 9 тактов и вырабатывает 7 результатов. На вычисление каждого результата расходуется 1.28.

**Д/З:** Концепция машин с явным параллелизмом (EPIC). Изучить микроархитектуру процессора Itanium 2 и концепцию EPIC.

Для машин с VLIW архитектурой был разработан новый метод планирования команд. При использовании этого метода из последовательности исходной программы генерируются длинные команды, путем просмотра программы за пределами базовых блоков. С точки зрения архитектурных идей, машину с длинным командным словом можно рассматривать как расширение RISC архитектуры. Как и в RISC архитектуре аппаратные ресурсы VLIW машины предоставлены компилятору и ресурсы планируются статически. К этим ресурсам относятся конвейерные функциональные устройства, шины, банки памяти. Для поддержки высокой пропускной способности используется большее количество регистров, предпочтение отдается простой логике управления. В отличие от традиционных машин регистры и шины не резервируются, а их использование определяется во время компиляции. Калькулятор с трассировочным планированием определяет участок программы без обратных дуг, т.е. без переходов назад, который становится кандидатом для составления расписания. Обратные дуги, обычно, имеются в программах с циклами. Для увеличения размера тела цикла, используется раскрутка цикла, что приводит к появлению больших фрагментов программы без обратных дуг. Если в программе имеются только переходы вперед, то компилятор выполняет эвристическое предсказание путей. Используется для оптимизации проводимой с учетом зависимостей между командами и ограничений аппаратуры. Во время планирования генерируется длинное командное слово. Все операции длинного командного слова выдаются одновременно и выполняются параллельно. Процесс продолжается до тех пока не будет оптимизирована вся программа. Ключевым условием достижения эффективной работы VLIW машины - является корректное предсказание переходов.

**Буферы прогнозирования условных переходов.**

Простейшей схемой динамического прогнозирования направления условных переходов является буфер прогнозирования или таблица истории условных переходов (branch prediction buffer). Буфер прогнозирования УП представляет собой небольшую память, адресуемую с помощью младших разрядов адреса команды перехода. Каждая ячейка этой памяти содержит один бит, который говорит о том, был ли предыдущий переход выполняемым или нет. В действительности не известно, является ли прогноз корректным. Т.к. значение буфера могла установить совсем другая команда перехода, которая имела тоже самое значение младших разрядов адреса, но это не имеет значения, прогноз – предположение, которое рассматривается как корректное и выборка команд начинается по прогнозируемому направлению. Если же предположение окажется неверным, бит (вид) прогноза инвертируется. Производительность буфера зависит от частоты его применения и точности предсказания.

**Пример:** команда условного перехода в цикле, которая являлась выполняемым переходом выполнялась 9 раз подряд, а затем не выполнилась. При однобитовой схеме прогноза направление перехода будет предсказано неверно дважды. На первой и последней итерации цикла. На последней итерации неправильный прогноз неизбежен, поскольку перед этим переход был 9 раз подряд выполняемым. Неправильный прогноз происходит из-за того, что бит прогноза инвертируется, поскольку на последней итерации переход был невыполняемым. Таким образом, точность прогноза, который выполнялся в 90% случаев составляет 80%. 2 некорректный и 8 корректных. В общем случае для команд условного перехода, переход является выполняемым много раз подряд, а затем является невыполняемым. Следовательно однобитовая схема предсказаний будет предсказывать неверно на двух итерациях, на первой и на последней.

Для повышения точности прогнозирования используется схема двух битового прогноза. В двух битовой схеме ошибка должна быть сделана дважды, прежде чем он изменится на противоположное значение. Диаграмма двух битовой схема предсказания.

выполняется

да

Не выполн.

нет

Не выполн.

Выполн.

Выполн.

Не выполняется

Двух битовая схема прогнозирования в действительности является частным случаем более общей схемы, которая в каждой строке буфера прогнозирования имеет n-битовый счетчик. Этот счетчик может принимать значение от нуля до , при этом схема прогноза будет следующей.

1. Если значение больше либо равно в точке на середине интервала , то переход объявляется как правильный, если переход выполняется – то к значению добавляется единица, если только это значение не достигло максимальной величины. Если прогноз был неверный, из счетчика вычитается единица.
2. Если значение счетчика меньше , то переход прогнозируется как невыполняемый, если направление перехода оказывается правильным, то от значения счетчика вычитается единица, если не правильным, то добавляется единица.

Исследование n-битовых схем прогнозирования показывают, что она работает почти так же хорошо, поэтому почти во всех случаях используют двух битовые схемы.

**Коррелированные схемы прогнозирования переходов.**

Двух битовая схема прогнозирования использует информацию о недавнем поведении команды условного перехода для прогноза будущего команды этой же команды. Вероятно, можно улучшить точность прогноза, если учитывать не только поведение того перехода, который пытаемся предсказать, но рассматривать также недавнее поведение других команд перехода.

**Рассмотрим пример:** фрагмент из тестового пакета SPEC92.

If (a==2) { a=0; }

If (b==2) { b=0; }

If (a!=b) {некие действия }

A и B размещены в регистрах r\_1 и r\_2, а r\_0 равен нулю.

SUBI R3, R1, #2 ; R3 = a-2

BNEZ R3, L1 ; переход в L2 (a != 2)

ADD R1, R0, R0 ; a = 0

L1: SUBI R3, R2, #2 ; R3 = b-2

BNEZ R3, L2; переход в L2 (a != 2)

ADD R2, R0, R0 ; b=0

L2: SUB R3, R1, R3 ; R3 = a-b

BEQZ R3, L3 ; переход в L3 (a==b)

…

…

L3: …

Схемы прогнозирования, которые для предсказания перехода используют поведение других команд перехода называются коррелированными. Схема прогнозирования называется прогнозом 1,1 если она используется поведение одного последнего перехода для выбора из пары однобитовых схем прогнозирования на каждый переход. Схема прогнозирования n,m использует поведение последних m переходов, для выбора из схем прогнозирования, каждая из которых представляет собой n-битовую схему прогнозирования для каждого отдельного перехода. Привлекательность коррелируемых схем прогнозирования переходов заключается в том, что они могут давать больший процент успешного прогнозирования, чем обычная двух битовая схема и требуют небольшого объема дополнительной аппаратуры. Глобальная история последних m переходов может быть записана в m-битовом сдвиговом регистре. Каждый разряд которого запоминает, был ли переход выполняемым или нет. Тогда буфер прогнозирования переходов может индексироваться конкатенацией младших разрядов адреса перехода с n-битовой глобальной историей.

**Д/З:** попытаться представить схему 2,2.

**Буфер прогнозирования коррелированной схемы прогнозирования 2,2.**

Рисунок #1.

На рисунке буфер типа 2,2 с общим числом строк, равным 64, где 4 младших разряда адреса команды перехода и 2 бита глобальной истории коррелированных переходов формируют 6-ти битовый индекс, который может использоваться для обращения к 64 счетчикам.

Сравнение простой двух битовой схемы прогнозирования с 4К строк и схемы прогнозирования 2,2 с 1к строк показывает, что схема 2,2 не только превосходит двух битовую схему с тем же количеством состояний, но часто превосходит даже двух битовую схему прогнозирования с бесконечным количеством строк.

Рассмотрим ситуацию при которой на стадии выборки находится команда перехода. На следующей стадии будет выполняться её дешифрация, для сокращения потерь необходимо знать по какому адресу необходимо выбирать следующую команду, для этого надо выяснить, что еще не дешифрованная команда является командой перехода, для этого нужно знать счетчик. Если знать все это, то потери на команду перехода могут быть сведены к нулю. Специальный аппаратный кэш прогнозирования переходов, который хранит прогнозируемый адрес следующей команды, называется буфером целевых адресов перехода (branch target buffer). Каждая строка этого буфера включает программный адрес команды перехода, прогнозируемый адрес следующей команды и предысторию команды перехода.

Рисунок #2.

Существуют вариации данного метода. Основная суть вариаций заключается в том, чтобы хранить в процессоре одну или несколько команд из прогнозируемой ветви перехода. Метод может применяться вместе с буфером целевых адресов перехода и без него. При этом появляется дополнительное преимущество. Буферизация самих целевых команд позволяет использовать метод свертывания переходов (branch folding). Свертывание переходов может использоваться для реализации нулевого времени выполнения для реализации самих команд перехода и в некоторых случаях нулевого времени выполнения условных переходов. Рассмотрим буфер команд перехода, который буферизует команды из прогнозируемой ветви. Пусть к нему выполняется обращение по адресу команды без условного перехода. Единственной задачей этой команды безусловного перехода является замена текущего значения программного счетчика. В этом случае, когда буфер адресов регистрирует попадание и показывает, что переход безусловный, конвейер просто может заменить команду, которая выбирается из кэш памяти, т.е. саму команду безусловного перехода на команду из буфера. Таким образом в некоторых случаях удается убрать потери для команд условного перехода, если код условия установлен заранее.

**Д/З:** представить такой случай.

Схемы прогнозирования условных переходов ограничены, как точностью прогноза, так и потерями в случае неправильного прогноза. Типичные схемы прогнозирования достигают 80-95% точности, в зависимости от типа программы и размера буфера. Кроме увеличения точности можно пытаться уменьшить потери при неверном прогнозе, обычно это делается с помощью выборки команд из обеих ветвей условия, т.е. по предсказанному и не предсказанному условию. Это требует двух портовой памяти, кэш памяти с расслоением или осуществления выборки в начале по одному направлению, а затем по другому. Хотя подобная организация увеличивает стоимость организации, возможно, это единственный способ снижения потерь на условные переходы ниже определенного уровня. Другие решения заключаются в кэшировании адресов и команд в другом буфере. В современных микропроцессорах используется около двух десятков различных алгоритмов.

**Кэш память**

В основе реализации иерархии памяти современных компьютеров лежат 2 основных принципа:

1. Принцип локальности обращений
2. Соотношение стоимость/производительность

Принцип локальности обращений говорит о том, что большинство программ не выполняют обращений ко всем своим командам и данным равновероятно, а оказывают предпочтения некоторой области адресного пространства. Иерархия памяти современных компьютеров строится на нескольких уровнях, причем более верхний уровень меньше по объему, быстрее и имеет большую стоимость в пересчете на один байт хранения, чем более низкий. Уровни иерархии взаимосвязаны. Данные некоторого уровня могут быть найдены и на более низком уровне. Сверху вниз до конца иерархии, от начала до конца. Иерархия памяти может состоять из многих уровней, но в каждый момент времени рассматриваются только 2 близ лежащих. Минимальная единица информации, которая может присутствовать, а может отсутствовать в двух уровневой реализации называется блоком. Размер блока может быть, как фиксированным, так и переменным. Если размер блока фиксированный – то объем памяти кратен размеру блока. Успешное или не успешное обращение к высшему уровню называется попаданием или промахом. Попадание – есть обращение к объекту в памяти, который найден на более высоком уровне. Промах означает, что он не найден на этом уровне. Коэффициент попаданий есть доля обращений, найденных на более высоком уровне. Может представляться в процентах. Доля промахов есть доля обращений, которые не найдены на более высоком уровне. Коэффициенты попаданий и промахов являются важными характеристиками. Чтобы описать некоторый уровень иерархии памяти надо ответить на следующие 4 вопроса:

1. Где может размещаться блок на верхнем уровне иерархии?
2. Как можно найти блок, когда он находится на верхнем уровне?
3. Какой блок должен быть замещен в случае промаха?
4. Что происходит во время записи?

**Организация кэш памяти**

Концепция кэш памяти возникла раньше, чем архитектура IBM 360. Сегодня кэш память имеется во всех классах компьютеров, и даже во множественном числе.

Типовые значения ключевых параметров для кэша памяти рабочих станций и серверов.

* Размер блока до 128 байт.
* Время попадания – один такт синхронизации.
* Потери при промахе – до 10-ти тактов синхронизации.
* Время пересылки – до 20-ти тактов.
* Доля промахов – от 1 до 20%.
* Размер кэш памяти – до 16МБ.

Детальная организация кэш памяти.

**Размещение блока в кэш памяти.**

По типу размещения определяют 3 основных типа организации:

Если каждый блок памяти имеет только одно фиксированное место, на котором он может появиться, то такая память называется кэшем с прямым отображением. Это наиболее простая организация кэша, при которой для отображения адресов блоков основной памяти на адреса кэш памяти используются младшие разряды адреса блока. Таким образом все блоки основной памяти, имеющие одинаковые младшие разряды в своем адресе попадают в один блок кэш памяти. **Д.З. проблема aliasing кэш памяти.**

Direct Mapped Cache (DMC)

Адрес проц.

|  |  |  |
| --- | --- | --- |
| Tag | Line | Offset |
| 0010 | 001 | 100 |
| 2 | 1 | 4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |  |
| 1 | 2 |  |  |  |  | X |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |

Процессор выберет заданную ячейку, если указаны нужные line и offset и tag равен значению, заданному в таблице.

**Следующий принцип организации кэш памяти.**

Если некоторый блок памяти может располагаться в любом месте кэш памяти – то такой кэш называется полностью ассоциативным.

Fully Associative Cache (FAC)

|  |  |
| --- | --- |
| Tag | Offset |
| 0010001 | 4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |  |
| 1 | 2 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |
| 3 | 17 |  |  |  |  | X |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |
| 6 | 45 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |

**N-way Set Associative Cache (NSAC)**

|  |  |  |
| --- | --- | --- |
| Tag | Set | Offset |
| 00100 | 01 | 100 |
| 7 | 1 | 4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 0set |  |  |  |  |  |  |  |  |  |
| 1 0set | 2 |  |  |  |  |  |  |  |  |
| 2 1set | 4 |  |  |  |  | X |  |  |  |
| 3 1set | 17 |  |  |  |  |  |  |  |  |
| 4 2set |  |  |  |  |  |  |  |  |  |
| 5 2set |  |  |  |  |  |  |  |  |  |
| 6 3set | 45 |  |  |  |  |  |  |  |  |
| 7 3set |  |  |  |  |  |  |  |  |  |

Обычно множество представляет собой группу из двух или большего числа блоков в кэше. Если множество состоит из n блоков, то такое размещение называется множественно-ассоциативным с n каналами. Для размещения блока прежде всего необходимо определить множество. Множество определяется младшими разрядами адреса блока памяти.

Диапазон возможных организаций кэш-памяти очень широк: кэш-память с прямым отображением есть просто одноканальная множественно-ассоциативная кэш-память, а полностью ассоциативная кэш-память с m блоками может быть названа m-канальной множественно-ассоциативной. В современных процессорах как правило используется либо кэш-память с прямым отображением, либо n-канальная множественно-ассоциативная кэш-память.

У каждого блока в кэш-памяти имеется адресный тег, указывающий, какой блок в основной памяти данный блок кэш-памяти представляет. Эти теги обычно одновременно сравниваются с выработанным процессором адресом блока памяти.

Кроме того, необходим способ определения того, что блок кэш-памяти содержит достоверную или пригодную для использования информацию. Наиболее общим способом решения этой проблемы является добавление к тегу так называемого бита достоверности (valid bit).

Адресация множественно-ассоциативной кэш-памяти осуществляется путем деления адреса, поступающего из процессора, на три части: поле смещения используется для выбора байта внутри блока кэш-памяти, поле индекса определяет номер множества, а поле тега используется для сравнения. Если общий размер кэш-памяти зафиксировать, то увеличение степени ассоциативности приводит к увеличению количества блоков в множестве, при этом уменьшается размер индекса и увеличивается размер тега.

**Какой блок кэш памяти должен быть замещен при промахе?**

При возникновении промаха, контроллер кэш-памяти должен выбрать подлежащий замещению блок. Польза от использования организации с прямым отображением заключается в том, что аппаратные решения здесь наиболее простые. Выбирать просто нечего: на попадание проверяется только один блок и только этот блок может быть замещен. При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти имеются несколько блоков, из которых надо выбрать кандидата в случае промаха. Как правило для замещения блоков применяются две основных стратегии: случайная и LRU.

В первом случае, чтобы иметь равномерное распределение, блоки-кандидаты выбираются случайно. В некоторых системах, чтобы получить воспроизводимое поведение, которое особенно полезно во время отладки аппаратуры, используют псевдослучайный алгоритм замещения.

Во втором случае, чтобы уменьшить вероятность выбрасывания информации, которая скоро может потребоваться, все обращения к блокам фиксируются. Заменяется тот блок, который не использовался дольше всех (LRU - Least-Recently Used).

Достоинство случайного способа заключается в том, что его проще реализовать в аппаратуре. Когда количество блоков для поддержания трассы увеличивается, алгоритм LRU становится все более дорогим и часто только приближенным.

**Что происходит во время записи?**

При обращениях к кэш-памяти на реальных программах преобладают обращения по чтению. Все обращения за командами являются обращениями по чтению и большинство команд не пишут в память. Обычно операции записи составляют менее 10% общего трафика памяти. Желание сделать общий случай более быстрым означает оптимизацию кэш-памяти для выполнения операций чтения, однако при реализации высокопроизводительной обработки данных нельзя пренебрегать и скоростью операций записи.

К счастью, общий случай является и более простым. Блок из кэш-памяти может быть прочитан в то же самое время, когда читается и сравнивается его тег. Таким образом, чтение блока начинается сразу, как только становится доступным адрес блока. Если чтение происходит с попаданием, то блок немедленно направляется в процессор. Если же происходит промах, то от заранее считанного блока нет никакой пользы, правда нет и никакого вреда.

Однако при выполнении операции записи ситуация коренным образом меняется. Именно процессор определяет размер записи (обычно от 1 до 8 байтов) и только эта часть блока может быть изменена. В общем случае это подразумевает выполнение над блоком последовательности операций чтение-модификация-запись: чтение оригинала блока, модификацию его части и запись нового значения блока. Более того, модификация блока не может начинаться до тех пор, пока проверяется тег, чтобы убедиться в том, что обращение является попаданием. Поскольку проверка тегов не может выполняться параллельно с другой работой, то операции записи отнимают больше времени, чем операции чтения.

Очень часто организация кэш-памяти в разных машинах отличается именно стратегией выполнения записи. Когда выполняется запись в кэш-память имеются две базовые возможности:

* сквозная запись (write through, store through) - информация записывается в два места: в блок кэш-памяти и в блок более низкого уровня памяти.
* запись с обратным копированием (write back, copy back, store in) - информация записывается только в блок кэш-памяти. Модифицированный блок кэш-памяти записывается в основную память только когда он замещается. Для сокращения частоты копирования блоков при замещении обычно с каждым блоком кэш-памяти связывается так называемый бит модификации (dirty bit). Этот бит состояния показывает был ли модифицирован блок, находящийся в кэш-памяти. Если он не модифицировался, то обратное копирование отменяется, поскольку более низкий уровень содержит ту же самую информацию, что и кэш-память.

**Д.З. сравнение подходов к организации записи.**