**Аппаратные оптимизации**

Оптимизация – модификация программ с целью улучшения их характеристик без изменения функциональности (логики работы).

Оптимизация может выполняться не только по отношению к программе.

Оптимизация:

* На уровне алгоритма:
  + Выбор более быстрого алгоритма
* На уровне машинного кода
  + Выбор более быстрого сочетания команд при компиляции
* На аппаратном уровне
  + Ускорение выполнения машинных команд

Оптимизация может быть

* Не только по скоросте выполнения

Но и по другим свойствам

* По объему потребляемой памяти
* По размеру получаемого машинного кода

Выполнения команды можно разделить на несколько этапов:

* Чтение инструкции (Instruction Fetch)
* Декодирование инструкции (Instruction Decode)
* Выполнение инструкции (Execute)
* Запись результатов (Wride-Back)

Количество и назначения этапов могут быть и другими

За каждый этап отвечают каждый части процессора

IF ID EX WB

Идея:

* Объединить устройства в конвейер

Плюсы:

* В зависимости от количества этапов и размера программы ее выполнение может происходить в разы быстрее
* Более эффективно используются устройства, входящие в состав процессора

Конфликты:

* По данным
* По управлению
* Структуры

При возникновении конфликтов:

* Эффективность конвейера падает
  + Часть конвейера приходится приостанавливать
  + Или весь конвейер перезапускать

Проблема:

* Чтобы эффективно загрузить конвеер, нужно в каждый момент времени, какая инструкция будет выполнена следующей
* Если в программе есть условные переходы, адрес следующей команды становится известен после выполнения (ЕХ)
* ….

Идея

* Попробовать предсказать, будет или не будет выполнен без(-условный) переход
* Пример:
  + Безусловные переходы выполняются
  + Условные переходы выполняются
  + Условные переходы вперед **не** выполняются
  + Косвенные переходы **не** выполняются

Предсказания переходов:

* Статическое
* Динамическое
  + Запоминать результаты выполнения
  + Нескольких последних условий перехода
  + Принимать решения исходя из этой информации
  + Информация для динамического предсказания переходов сохраняется в так называемом BTB (Branch Transfer Buffer)
  + Размеры BTB различны для различных моделей процессоров

Вызов подпрограммы – это безусловный переход

Возврат из подпрограммы (RET) – это косвенный безусловный переход

Оптимизация подпрограмм:

* Для оптимизации вызовов подпрограмм используются т.н. Return Stack
* Емкость Return Stack в большинстве современных моделей процессоров Intel составляет 16 вложенных вызовов

Конфликт по данным

**Чтение после записи**

Mov bx, si

Mov cx, 10

Mov ax, cx ; Конфликт, тк нужно дождаться, пока предыдущая команда запишется

Решение: вставить в середину первую строку

**Запись после записи**

Mov ax, 10

Mov ax, 15

Решение – внеочередное исполнение (Тк команды могут меняться местами)

**Конфликт «Запись после чтения»**

add ax, bx

mov cx, ax ; После этого содержимое ax уже не важно

mov ax, 10

Идея (Прием – переименование регистров):

* Разместить в процессоре больше регистров, чем их существует в процессоре с точки зрения программиста
* При необходимости (и по возможности) для различных команд одно и то де имя регистра (Например, EAX) связывать с различными физическими устройствами

**Внеочередное исполнение**

* Процессор может выполнять инструкции не в том порядке, в котором они идут в программе
  + Поступающие инструкции помещаются в RS (Reservation Station)
  + Когда все операнды какой-либо инструкции готовы, она выполняется
    - Данные, для которых уже готовы операнды, могут выполняться без очереди, а остальные ожидают, пока не будут готовы операнды
  + Результат помещается в ROB (ReOrder Buffer)
  + Логика программы не изменяется
  + Результаты выполнения инструкций применяются в порядке их появления в коде программы

Результаты выполнения инструкции:

* Записываются в операнд-приемник
* Передаются в RS, чтобы запустить зависимые от этих результатов инструкции
  + Т.е. зависимая инструкция может начать выполняться раньше, чем нужные ей данные будут записаны в источник

Структурные конфликты

Например: устройство, отвечающее за деление может быть занято, а идет два деления подряд

Идея!

* Организовать несколько конвееров и подавать в них инструкции параллельно
* В современных процессорах Intel есть 2 конвеера на каждое ядро:
  + U-pipe, V-pipe (Умеет только простые команды)

**Hyper-threading**

* Для каждого физического ядра процессор содержит:
  + N наборов регистров
  + N контроллеров прерываний
* Одному физическому ядру соответствует N логических ядер
* ОС видит логические ядра
  + Например, 4-ядерный процессор с HT выглядит как 8-ядерный процессор
* Выполняется программа для одного из логических ядер
* Возникает задержка:
  + Конфликт по данным
  + Кэш-промах
  + Ошибка предсказания перехода
* Вместо простого ожидания ядро переключается на выполнение программы для другого логического ядра.

СPU

M  
M  
U

Внешние устройства

I/O

КЕШ

RAM

S

Регистровый

файл

АЛУ

Instruction Prefetch

Branch Prediction Unit

ROB

RS

Decoder ICache

…

**Instruction Prefetch**

* Считывать команды не по одной, а сразу небольшими блоками (по 16 или 32 байта)

**Decoder ICache**

* Появился в процессорах линейки Sandy Bridge
  + Идея: сохранять результаты декодирования инструкций, которые уже выполнялись
  + Позволяет оптимизировать выполнение циклов, состоящих не более чем из 500 инструкций.
    - В некоторых случаях – до 1000

**Кэширование**

* Идея: сохранять результаты вычислений на случай, если они понадобятся повторно
* В современных процессорах:
  + Кэш данных
  + Кэш инструкций
  + И др.
* Кэш состоит из кэш-линий
  + Каждая кэш-линия позволяет хранить значения некоторого количества смежных байт
  + ОЗУ условно делится на области такого же размера
  + При обращении к любому байту какой-либо области ей может быть поставлена в соответствие кэш-линия.
    - Содержимое области копируется в кэш.
* При обращении к памяти возможны две ситуации:
  + Данные уже есть в кэше – cache hit
  + Данных еще нет к кэше – cache miss
* В случаях попадания данные берутся из кеша
* В случае промаха происходит обращение к более медленному ОЗУ

Кеш может быть реализован:

* Внутри ядра процессора
* Общий для всех ядер процессора
* Общий для всех процессоров
* И т.д.

Архитектура гарантирует когерентность ядер

**Архитектуры процессоров:**

* RISC (Reducer Instruction set computer);
* CISC (Complex instruction set computer);
* И др.

Современные процессоры Intel имеют гибридную архитектуру

* Набор команд – как у CISC
* Исполнительное устройство - RISCT