5. Векторизация, многопоточность, внеочередное исполнение инструкций.

Векторные вычисления – это вид параллельных вычислений с параллелизмом на уровне данных (SIMD – Single Instruction- Multiple Data = «одна команда – несколько элементов данных»). В скалярных микропроцессорах общего назначения векторные вычисления поддерживаются с помощью векторных расширений архитектур: MMX (Multimedia Extensions), SSE (Streaming SIMD Extensions, потоковое SIMD-расширение процессора), AVX (Advanced Vector Extensions). Векторные расширения включают:

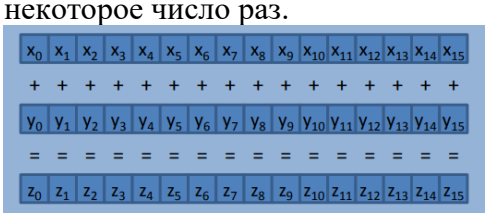
- векторные регистры – хранят множества скалярных значений;
- векторные команды – для работы с векторными регистрами.

Скалярная программа – программа, оперирующая отдельными числами, векторная программа – программа, оперирующая векторами, **векторизация** (вид распараллеливания) – преобразование скалярной программы в векторную. Кандидаты для векторизации - это самые внутренние циклы программы. Для фрагмента типа

```
Do i = 1, n
  Z(i) = X(i)+Y(i)
End Do
```

в скалярном режиме потребуется: прочитать элемент X(i), прочитать элемент Y(i), выполнить сложение, записать результат в Z(i), увеличить параметр цикла, проверить условие цикла.

В векторном режиме: загрузить порцию массива X, загрузить порцию массива Y (эти две операции будут выполняться со сдвигом в один такт, т.е. практически одновременно), векторное сложение, запись порции массива в память, если размер массивов больше длины векторных регистров, то повторить эту последовательность некоторое число раз.



Больше о векторизации: здесь и здесь.

Многопоточность - вариант реализации вычислений, при котором для решения задачи запускаются и выполняются несколько независимых потоков вычислений в одном процессе. Все потоки находятся в одном адресном пространстве, делят одну и ту же виртуальную память и имеют те же права доступа, что и процесс.

Внеочередное исполнение - исполнение машинных инструкций не в порядке поступления, а по мере готовности к их выполнению. Процессор помещает поступающие команды в очередь по порядку и начинает ожидать выполнения всех условий, для обработки этих команд. При этом для исполнения одной команды в очереди все условия могут быть выполнены раньше, чем для другой. Это означает, что команда, поставленная в очередь последней, благодаря внеочередному исполнению может быть выполнена раньше всех остальных. Благодаря реализации этого алгоритма время простоя процессоров при ожидании выполнения условий для исполнения команд в очереди существенно сокращается.

Больше о внеочередном исполнении.

6. CISC и RISC процессоры, примеры реализаций.

Подходы к конструированию процессоров:

1) придумать как можно больше разных команд и предусмотреть как можно больше разных режимов адресации: **CISC**= Complex Instruction Set Computers. Примеры: Motorola семей 6800, 6809 и 68000; Intel 8080, iAPX432 и x86 (он же IA-32); Zilog Z80, Z8 и Z8000; National Semiconductor 32016 и NS320xx линии; MOS технология 6502 серии; семейство Intel 8051, IBM 360, x86_64 (он же AMD64)

2) реализовать минимальное множество команд и режимов адресации: **RISC**= Reduced Instruction Set Computers. Примеры: DEC Alpha, Power PC, Intel Itanium, MIPS, SPARC, серия архитектур ARM (ARM7, ARM9, ARM11, Cortex)

Особенности **CISC**:

1) команды выполняются несколько тактов
2) небольшое число регистров общего назначения
3) большое количество машинных команд, некоторые из них реализуют сложные операторы языков высокого уровня (ЯВУ)
4) различные форматы команд с разной длиной (Pentium – длина команды 1-10 байт)
5) большое количество режимов адресации
6) наличие команд, где обработка совмещается с обращением к памяти
7) широко использование микропрограммирования (микропрограмма — программа, реализующая набор инструкций процессора; в процессоре, использующем микрокод, каждая машинная инструкция реализуется в виде серии микроинструкций — микропрограммы, микрокода)

Особенности **RISC**:

1) команда выполняется за такт
2) большой (не менее 32) процессорный файл регистров общего назначения, это позволяет большему объёму данных храниться в регистрах на процессорном кристалле большее время и упрощает работу компилятора по распределению регистров под переменные
3) небольшое количество команд (не более 128) фиксированной длины
4) и их форматов
5) малое количество режимов адресации
6) работа с памятью ограничена специальными командами
7) сложные команды (функции аппаратуры) выполняются набором простых команд=отсутствие микропрограммирования
8) развитый конвейер команд: он эффективно загружен за счёт унификации форматов команд

Для технологии RISC характерна сравнительно простая структура устройства управления. Площадь, выделяемая на кристалле микросхемы для реализации УУ, существенно меньше. Как следствие, появляется возможность разместить на кристалле большое число регистров ЦП. Кроме того, остается больше места для других узлов ЦП и для дополнительных устройств: кэш-памяти, блока арифметики с плавающей запятой, части основной памяти, блока управления памятью, портов ввода/вывода.

Унификация набора команд, ориентация на конвейерную обработку, унификация размера команд и длительности их выполнения, устранение периодов ожидания в конвейере — все эти факторы положительно сказываются на общем быстродействии. Простое устройство управления имеет немного элементов и, следовательно, короткие линии связи для прохождения сигналов управления. Малое число команд, форматов и режимов приводит к упрощению схемы декодирования, и оно происходит быстрее. Высокой производительности способствует и упрощение передачи параметров

между процедурами. Таким образом, применение RISC ведет к сокращению времени выполнения программы или увеличению скорости за счет сокращения числа циклов на команду.

Многие современные CISC-машины имеют средства для прямой поддержки функций ЯВУ, наиболее частых в этих языках (управление процедурами, операции с массивами, проверка индексов массивов, защита информации, управление памятью и т. д.). RISC также обладает рядом средств для непосредственной поддержки ЯВУ и упрощения разработки компиляторов ЯВУ, благодаря чему эта архитектура в плане поддержки ЯВУ почти равна CISC.

Недостатки RISC прямо связаны с некоторыми преимуществами этой архитектуры. Принципиальный недостаток — сокращенное число команд: на выполнение ряда функций приходится тратить несколько команд вместо одной в CISC. Это удлиняет код программы, увеличивает загрузку памяти и трафик команд между памятью и ЦП. Исследования показали, что RISC-программа в среднем на 30% длиннее CISC-программы, реализующей те же функции.

Хотя большое число регистров дает существенные преимущества, само по себе оно усложняет схему декодирования номера регистра, тем самым увеличивается время доступа к регистрам. УУ с аппаратной логикой, реализованное в большинстве RISC-систем, менее гибко, более склонно к ошибкам, затрудняет поиск и исправление ошибок, уступает при выполнении сложных команд.

Больше о: CISC и RISC
Больше о CISC и RISC: здесь

Здесь же можно будет написать наборы инструкций из следующего вопроса.