Содержание

БЛАГОДАРНОСТИ

Авторы хотели бы поблагодарить
Карла Смита (Karl Smith),
Майка Коплэнда (Mike Copeland)
и Манфреда Чутку (Manfred Choutka)
из фирмы Infineon Technologies,
Хоакима Клейна (Joachim Klein)
из фирмы Hitex Development Tools GmbH,
а также Стива Брауна (Steve Brown)
и Венди Уокера (Wendy Walker)
и выразить свою благодарность
сотням пользователям
микроконтроллеров семейства 166
в Великобритании
за их вклад в написание этой книги.

Предисловие

Данное руководство содержит основные сведения, необходимые для разработки вашего первого устройства на основе микроконтроллера XC166. Это простые, но основополагающие сведения, знание которых, особенно на начальном этапе разработки, позволит сэкономить немало времени и денег. Цель этой книги – дополнить официальное руководство пользователя XC166 теми фактами, которые пригодятся вам на практике.

Кое-что из материала книги можно отыскать в спецификациях на семейство XC166, но большая часть информации получена в результате нашей практической работы и найти ее можно только в данной книге. Многие из поднятых в ней тем отнюдь не очевидны или вообще не рассматривались ранее. Если руководство пользователя на микроконтроллер XC166 достаточно полно освещает какой-либо вопрос, то в книге делается ссылка на него и информация не дублируется. Данное издание является далеко не полным и немало дополнительной информации можно найти на веб-сайте фирмы Infineon.

Достоверность информации, представленной в данной книге, подтверждена специалистами фирмы Hitex. Однако, несмотря на это, фирма Hitex не берет на себя ответственности за возможные неточности. Те или иные субъективные или неподтвержденные сведения, содержащиеся в книге, не всегда отражают официальную точку зрения фирм Hitex Development Tools Ltd. и Infineon Technologies AG.

Материал подготовили:
Майкл Бич (Michael Beach)
Дэвид Гринхилл (David Greenhill)
Дополнительный материал предоставили:
Карл Смит (Karl Smith), **Infineon Technologies UK**
Хоаким Клейн (Joachim Klein), **Hitex Development Tools**

Глава 1

RISC-архитектура для встроенных приложений

1.1. Введение

Популярность RISC-микроконтроллеров непрерывно растёт. В процессорном ядре C166S V2, на базе которого построена серия микроконтроллеров XC166, широко используются принципы архитектуры вычислений с сокращённым набором команд (Reduced Instruction Set Computer — RISC), благодаря чему удалось достичь очень высокой производительности процессора при весьма умеренной его стоимости. Чтобы понять, почему именно RISC-технологии больше подходят для встроенных приложений, работающих в режиме реального времени, полезно будет вспомнить, как развитие традиционной архитектуры микропроцессоров с полным набором команд (Complex Instruction Set Computer — CISC), пик популярности которой пришелся на период с конца 1980-х по середину 1990-х годов, привело к появлению RISC-архитектуры. Тем не менее, необходимо учитывать, что несмотря на множество достоинств, присущих RISC-технологии, не все RISC-микроконтроллеры, по определённым причинам, полностью их используют. В данной главе мы рассмотрим несколько таких «критических» случаев.

1.2. RISC-подобное ядро C166S V2: предпосылки создания

Причиной снижения интереса к микропроцессорам, использующим архитектуру с полным набором команд (CISC), стала необходимость существенно увеличить производительность микропроцессорных систем. В первую очередь, речь идёт о требованиях, предъявляемых графическими рабочими станциями и современными компьютерными видеоиграми. Традиционно, стандартные наборы ассемблерных команд (инструкций) микропроцессоров разрабатывались таким образом, чтобы максимально упростить работу программиста. Это достигалось путём широкого использования микрокодов, позволяющих создавать всё более мощные инструкции, такие, например, как команда перемножения трёх операн-

дов. Благодаря этому, программист, пишущий на ассемблере (или разработчик компилятора языка высокого уровня), может добиваться тех же результатов, что и с применением простых команд, но с меньшей затратой сил.

Чем большее количество различных инструкций необходимо распознавать и обрабатывать (декодировать) центральному процессору, тем сложнее электронные схемы, размещённые на его кристалле, и тем большее количество циклов тактовой частоты занимает процесс обработки команд. Увеличение площади полупроводникового кристалла повышает стоимость устройства и потребляемую им мощность. Поскольку максимальная тактовая частота работы электронных схем микропроцессора физически ограничена возможностями полупроводниковых технологий, очевидно, что максимальная производительность напрямую зависит от количества машинных циклов, требуемых для выполнения команд процессора.

RISC-технология переносит основную нагрузку на программистов, пишущих на ассемблере и разработчиков компиляторов языка высокого уровня, а не на программирование в микрокодах. Исследования, проведённые учёными и производителями полупроводниковой техники, показали, что при одной и той же тактовой частоте корректно запрограммированный RISC-процессор обеспечивает гораздо более высокую производительность, чем процессор, использующий технологию CISC.

Как ни странно, разработчики встроенных приложений средней производительности не спешат отказываться от CISC-микроконтроллеров. Если в сфере высокопроизводительных устройств RISC-процессоры, такие как ARM9, MIPS и Hyperstone, создают жёсткую конкуренцию традиционным CISC-процессорам PowerPC и Pentium, то для более распространённых задач встроенных приложений RISC-устройства используются относительно редко. Между тем, возрастающая сложность современных алгоритмов управления встроенными устройствами приводит к тому, что выполнение практически любых задач, кроме самых элементарных, требует использования больших вычислительных мощностей. Кроме того, в таких областях, в отличие от использования микропроцессоров в рабочих станциях, весьма критична величина максимального времени реакции на недетерминированные события — параметр, с которым у CISC-процессоров дело обстоит особенно плохо. Впрочем, большинство RISC-процессоров ARM также не обладают выдающимися характеристиками по времени реакции.

Многие современные 16-битные микроконтроллеры средней производительности построены на базе существующих CISC-архитектур (S12, H8, M16C и т. п.), которые, как и 8-битные устройства (например, 8051), разработаны более 20 лет назад. Производители микросхем обеспечивают потребителям возможность модернизации существующих систем, поэтому микропроцессоры новых серий часто разрабатываются на базе уже существующих. Использование тех же или близких архитектуры и набора команд избавляет потребителя от необходимости тратить деньги на переписывание уже существующих ассемблерных программ.

Микроконтроллеры, как и рабочие станции, могут быть запрограммированы на языках высокого уровня, что позволяет сократить время написания программ и улучшить удобство обслуживания системы. К сожалению, использование даже

самых лучших компиляторов приводит к потерям быстродействия[1)], что ещё раз подчеркивает необходимость увеличения производительности процессоров.

Помимо прямой обработки данных, микроконтроллеры также должны получать информацию от периферийных устройств, таких как АЦП, ШИМ, таймеров, портов, ФАПЧ и т. п. Все это требует обработки информации в режиме реального времени и быстрой реакции на прерывания.

Ниже приведены основные недостатки, свойственные CISC-микроконтроллерам:

1. Длительное и непредсказуемое время ожидания прерывания

Во время выполнения сложных «снижающих трудозатраты программиста» ассемблерных команд все вычислительные ресурсы процессора оказываются задействованы, что не позволяет обслуживать поступающие извне прерывания. Это приводит к тому, что длительность ожидания обработки прерывания оказывается непредсказуемо велика, что может создать серьезные проблемы в быстродействующих системах реального времени. Одним из методов уменьшения времени реакции CISC-микроконтроллеров является интеграция в ядро «блока временной обработки» или использование внешнего процессора для разгрузки критичных по времени операций. Однако, это повышает сложность проектов и требует использования очень компактных микрокодов, в дополнение к куда более распространённому языку C и ассемблеру для программирования самого CISC-ядра.

2. Медленное декодирование вследствие слишком большого набора команд

Загруженные команды нужно распознать среди нескольких сотен или даже тысяч возможных, поэтому процесс декодирования замедляется и усложняется.

3. Замедление работы устройств памяти, вызываемое частым доступом к ним

Данные обычно берутся процессором из внешней памяти, и помещаются в регистры типа аккумулятора. С ними производятся математические или логические операции, затем результат снова записывается в память. Часто при выполнении процедуры то или иное значение может потребоваться несколько раз, что требует дополнительных обращений к памяти для записи или чтения.

4. Медленный вызов процедур

Если при вызове подпрограммы используются параметры (что чаще всего и происходит при качественном программировании на языке высокого уровня), то они по отдельности помещаются в стек. Затем эти параметры обрабатываются в регистре (регистрах) аккумулятора, и лишь после этого через стек возвращаются вызывающей подпрограмме.

5. Выполнение за один раз только одной операции

Каждое периферийное устройство или источник прерывания должны иметь специализированную обслуживающую подпрограмму, которая, как минимум, потребует помещения в стек и извлечения из него слова состояния программы и счётчика команд, а также передачи данных от периферийного устройства или к периферийному устройству.

[1)] По сравнению с программированием на языке ассемблера. (прим. ред.)

6. Пользовательская программа должна быть структурирована таким образом, чтобы соответствовать аппаратным возможностям микроконтроллера

Программирование встроенных приложений часто представляет собой множество отдельных задач, выполняемых в режиме реального времени и связанных в единую систему. В типовых CISC-процессорах переключение между задачами происходит довольно медленно. Часто при поступлении новой задачи оказывается, что многие регистры заняты и данные из них предварительно нужно поместить в стек. Эта проблема усугубляется при использовании компиляторов высокого уровня, характерным свойством которых является наличие в библиотечных функциях большого количества подлежащих сохранению локальных переменных.

7. Избыточные инструкции и режимы адресации

Почти повсеместное использование языков высокого уровня заставляет производителей микроконтроллеров обеспечивать совместимость с различными компиляторами путём введения в набор команд процессора соответствующих дополнительных инструкций. На практике, компиляторы чаще всего используют лишь несколько режимов адресации, а большое количество других инструкций остаются незадействованными, что усложняет процесс декодирования команд.

8. Несогласованные наборы команд процессора

Наборы команд становится всё труднее использовать из-за большого количества доступных типов данных и разнообразия поддерживаемых режимов адресации.

9. Неполная загруженность шины

Во время выполнения сложных команд внутренняя шина микроконтроллера простаивает.

1.3. Использование RISC-архитектуры во встроенных системах управления

Чтобы продемонстрировать, как использование RISC-архитектуры позволяет увеличить производительность микроконтроллеров, в качестве примера, мы взяли процессорное ядро C166S V2.

Основные определения:

- 1 такт = 1/частота генератора — базовая единица времени, используемая при рассмотрении процессорных систем.
- 1 машинный цикл = такт — минимальное время, необходимое процессору для выполнения наиболее простой операции. Поскольку машинный цикл позволяет абстрагироваться от тактовой частоты, это понятие можно использовать для сравнения RISC- и CISC-процессоров.
- Все отсчёты машинных циклов как для S12X, так и для C166 M2 приводятся в однокристальном режиме работы.

1.3.1. Шинный интерфейс

Характерной особенностью RISC-процессоров является высокий уровень их конвейеризации, что позволяет увеличить скорость выполнения команд. За счет перекрытия различных стадий, в любом машинном цикле могут одновременно выполняться до 4 команд. Упрощённо эти стадии можно представить следующим образом:

FETCH — выборка кода операции из памяти программ
DECODE — идентификация кода операции из небольшого списка и полученных операндов выборки
EXECUTE — выполнение операции, соответствующей найденному коду
WRITE-BACK — результат выполнения операции возвращается в память по заданному адресу

Таким образом, хотя инструкция требует четырёх машинных циклов, реально она выполняется всего за один машинный цикл (1 такт). Конвейеризация значительно ускоряет выполнение последовательного (линейного) кода, поскольку шина гарантированно будет полностью загружена.

Ряд наиболее современных RISC-процессоров (например, C166S V2) имеют 5-ступенчатый конвейер исполнения команд (добавлена стадия ADDRESS), а также 2-ступенчатый блок выборки команд с упреждением PREFETCH:

Высокопроизводительный блок выборки команд (IFU)

PREFETCH — выборка команд из памяти в предсказанном порядке; обнаруживаются любые ветвления, их выполнение или невыполнение обусловливаются логикой предсказания
FETCH — адрес следующей команды для выборки вычисляется с использованием правил предсказания ветвления

Конвейерный блок

DECODE — команды декодируются и, если требуется, производится доступ к регистровому файлу для считывания GPR (регистра общего назначения), используемого в режиме косвенной адресации
ADDRESS — вычисляются все адреса операндов
MEMORY — из памяти RAM и регистров извлекаются все необходимые операнды
EXECUTE — выполняется операция, соответствующая коду
WRITE BACK — результат операции возвращается в память по заданному адресу

Такая организация интерфейса теоретически обеспечивает удвоение производительности при выполнении линейного участка программы. Однако, даже при наличии блока предсказания ветвлений, который минимизирует снижение производительности, вызванное имеющимися в реальной программе ветвлениями, удвоения производительности достигнуть не удаётся.

1.3.2. Реакция на прерывание RISC-процессора

Процессорное ядро C166S V2 производит обработку прерываний с помощью встроенных инструкций, и поэтому для векторизации на программу обслуживания прерывания требуется 5 машинных циклов. Время реакции увеличивается из-за наличия сложных, но необходимых команд, таких как `MUL` и `DIV` (1 и 21 машинный цикл соответственно), но необходимо отметить, что в C166S V2 команда `DIV` является частично-прерываемой. В течение первых 4 циклов все прерывания блокируются, но в течение остальных 17 циклов прерывания могут восприниматься.

Высокая скорость обслуживания прерываний имеет особое значение в наиболее высокопроизводительных системах, таких, как блоки управления двигателями, сервоприводы и радары. В этих системах информация, поступающая в режиме реального времени, обрабатывается цифровым сигнальным процессором (DSP), выходные данные которого используются для замыкания контура управления системой. В этом случае, неустойчивость времени ожидания обработки прерываний проявляется в виде нежелательных флуктуаций параметров выходных сигналов контроллера.

1.3.3. Регистры и многозадачность

Обычные (CISC) микроконтроллеры имеют один или несколько специальных регистров, которые могут использоваться для математических, логических или булевых операций. В микроконтроллере 8051 имеется один регистр-аккумулятор и 8 других регистров, которые могут использоваться для операций с локальными переменными или с промежуточными результатами при сложных вычислениях. Эти дополнительные регистры используются также для доступа к памяти путем косвенной и/или индексной адресации.

Как было указано выше (см. пункты 3 и 4 раздела1.2), обычные процессоры (CPU) тратят много времени на перемещение данных из медленной внешней памяти в активные регистры. RISC-процессоры имеют очень много регистров общего назначения, которые можно использовать для хранения локальных переменных, параметров и промежуточных результатов. Процессорное ядро C166S V2 содержит 16 32-битных регистров общего назначения (GPR), каждый из которых может использоваться в качестве аккумулятора, указателя косвенного адреса или индексного регистра. Таким образом, вполне реальным становится хранение всех локальных переменных и промежуточных результатов во время выполнения большого количества подпрограмм в CPU, что значительно увеличивает скорость вычислений.

Следующим важным преимуществом RISC-архитектуры является наличие регистрового окна. Как только что было сказано, пользовательской программе доступны до 16 регистров. Однако, решать проблему подлинной многозадачности в системах реального времени значительно проще, имея возможность свободно

перемещать активный блок регистров в пределах адресного пространства встроенной оперативной памяти (RAM).

Подобная возможность реализуется с помощью контекстного указателя (CP), который определяет текущий абсолютный базовый адрес активного блока регистров в адресном пространстве памяти программ. Таким образом, ссылка «R0» означает регистр с адресом, указанным в CP (типовое значение 0xFD00). Соответственно, доступ к 16 регистрам, адрес первого из которых задан в CP, обеспечивается быстрым 4-битным смещением.

Возможно, лучшим примером использования CP является одновременная реализация фоновой задачи и прерываний в реальном времени. При обнаружении прерывания вместо того, чтобы переписывать все регистры GPR в стек, в него помещается значение CP текущего блока регистров и происходит переключение к определённому во время компоновки новому значению CP, задающему «свежий» блок регистров. Таким образом, сложное контекстное переключение осуществляется всего лишь за время выполнения одной команды, но при этом исключается возможность использования рекурсии (повторного запуска подпрограммы).

Комбинированный метод, допускающий повторный вход, использует указатель стека для динамического вычисления нового CP. В этом случае, при входе в подпрограмму-обработчик прерывания количество требуемых регистров вычитается из текущего указателя стека, а полученный результат помещается в CP, при этом старый CP помещается в стек. Таким образом, новый блок регистров располагается в адресном пространстве на вершине старого стека, а сразу за ним располагаются старый CP и новый стек. При выходе из обработчика прерывания, происходит переключение к исходному блоку регистров путем извлечения старого значения CP из стека. Указатель стека восстанавливается путём сложения текущего указателя стека с числом, соответствующим размеру нового блока регистров.

Следующим усовершенствованием RISC-процессора стала возможность использования перекрывающегося регистрового окна, благодаря чему при вызове новой подпрограммы часть регистров нового блока, заданного в новом CP, совпадает с исходно заданными в старом CP:

Регистры			Назначение
		R3'	Регистр локальных переменных подпрограммы и промежуточных результатов
		R2'	Регистр локальных переменных подпрограммы и промежуточных результатов
	R7	R1'	Общий регистр, R7 == R1'
CP'	R6	R0'	Общий регистр, R6 == R0'
	R5		Регистр для вызывающей программы и промежуточных результатов
	R4		Регистр для вызывающей программы и промежуточных результатов
	R3		Регистр для вызывающей программы и промежуточных результатов
	R2		Регистр для вызывающей программы и промежуточных результатов
	R1		Регистр для вызывающей программы и промежуточных результатов
CP	R0		Регистр для вызывающей программы и промежуточных результатов

Назначение символических имён регистрам общего назначения (GRP)

```
Модуль 1

;*** Назначение регистров общего назначения (GPR) локальным переменным -
 Подпрограмма ***
x_var       LIT       'R0'     ; Локальная переменная
y_var       LIT       'R1'     ; Локальная переменная

parm1       LIT       'R6'     ; Передаваемый параметр 1
parm2       LIT       'R7'     ; Передаваемый параметр 2

result      LIT       'R6'     ; Значение, возвращаемое подпрограммой
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Модуль 2
;*** Назначение регистров общего назначения (GPR) локальным переменным -
 Подпрограмма ***

a_var       LIT       'R2'     ; Локальная переменная
a_var       LIT       'R3'     ; Локальная переменная

input1      LIT       'R0'     ; Получаемый параметр 1
input2      LIT       'R1'     ; Получаемый параметр 2
ret1        LIT       'R0'     ; Окончательный результат, возвращаемый в R0
```

Значения, которые требуется передать подпрограмме, должны находиться в общей (перекрываемой) области. Предусмотрев это, программист может избежать лишних операций загрузки и выгрузки параметров. Этот приём можно использовать как в случае абсолютной адресации блока регистров, так и в режиме относительной адресации через указатель стека.

Возможность переключения к новому блоку рабочих регистров посредством изменения регистра контекстного указателя полезна при обработке прерываний, так как это устраняет необходимость сохранения содержимого текущего блока регистров перед выполнением команд служебной программы. Ведь процесс сохранения содержимого всех 16 регистров может занять значительное время. Обычно одна команда (здесь переключение контекста «SCXT») в начале программы обработки прерывания служит для того, чтобы сменить базовый адрес блока регистров, и вторая команда в конце подпрограммы восстанавливает исходный базовый адрес.

Другой способ, применяемый в некоторых RISC-микроконтроллерах, состоит в том, что при активировании предопределённого заранее источника прерывания открывается доступ к «альтернативному» комплекту регистров, благодаря чему сохранять содержимое текущего регистра не требуется. RISC-процессоры серий ARM7-TDMI обычно замещают 8 верхних регистров блока регистров по сигналу FIQ (запрос на быстрое прерывание), а у микроконтроллера C166S V2 (см. **Рис. 1.1.**) при инициализации любого из двух источников прерывания стано-

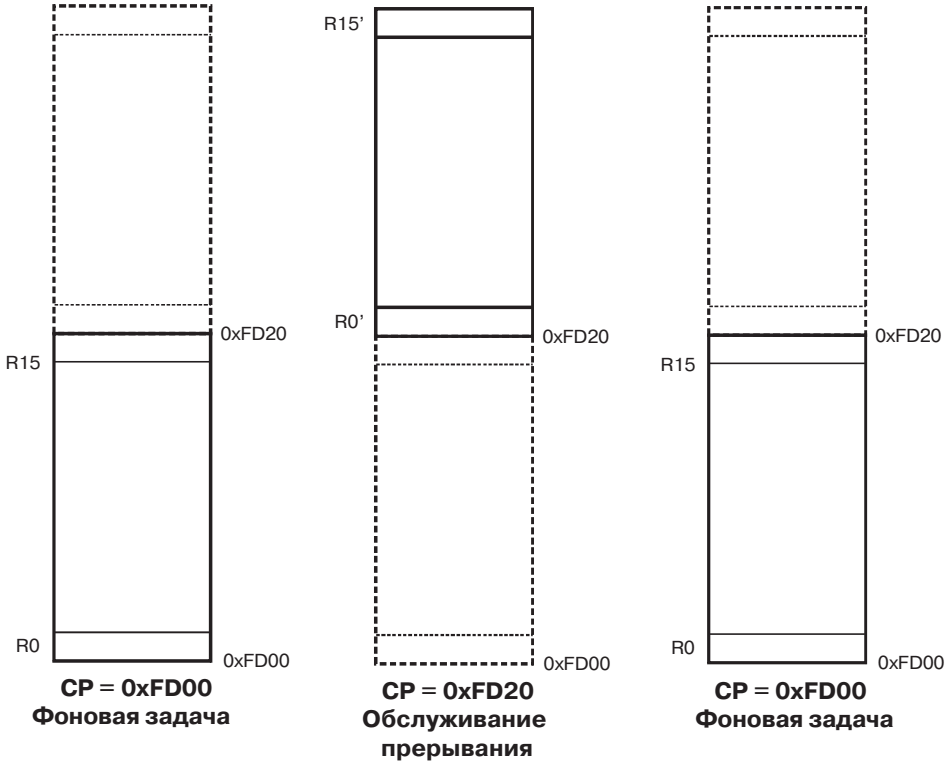

Рис. 1.1. Использование «альтернативного» комплекта регистров в микроконтроллере C166S V2

вятся доступны два полных «локальных» блока из шестнадцати регистров. В отличие от обычных блоков регистров, в нормальном режиме работы эти регистры не определены в адресном пространстве CPU, поэтому контекстное переключение не требуется, что позволяет сэкономить время[1)].

Чтобы максимально реализовать возможности RISC-регистров, необходимо тщательно учитывать, где конкретно находятся необходимые данные. Возможно, вам доставят определённое неудобство ограниченность и противоречивость некоторых режимов адресации, предусмотренных, например, для команд `MUL` и `DIV`. К счастью, большинство задействованных при их выполнении операндов уже будет находиться в регистрах, что исключает использование различных способов адресации. Как и следовало ожидать, наибольший выбор способов адресации предусмотрен для самых простых команд — команд перемещения данных. Тот факт, что RISC-процессоры очень тщательно анализируют факторы, влияющие на быстроту выполнения команд, становится очевидным уже с первого знакомства с ними!

[1)] Фактически, текущему локальному блоку регистров можно дать физический адрес, и он может стать видимым при установке соответствующих битов в поле BANK регистра PSW.

1.3.4. Использование сокращенного набора команд RISC-процессора

Когда большинство команд выполняется за один цикл, некоторые типовые «быстрые» команды, такие как CLEAR, INC и DEC, оказываются «лишними». Тогда, чтобы свести суммарное количество команд к минимуму, RISC-процессоры просто-напросто пропускают (не используют) их. Ниже приводится пример:

```
Команда[1)]              80C196     Такты    X166S V2          Такты
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Очистить слово         CLR        4        AND Rn, #0        2
Декремент слова        DEC        4        SUB Rn, #01       2
Инкремент слова        INC        4        AND Rn, #01       2
```

3-операндовые команды являются обычными для CISC-процессоров, но в RISC-процессорах они не используются. Несмотря на то, что в таком случае требуется задействовать дополнительные команды, общее количество тактов всё равно оказывается меньшим, нежели в 3-операндовом CISC-эквиваленте, кроме того, более короткие RISC-команды обеспечивают большее удобство в обслуживании прерываний.

Рассмотрим приведенный ниже пример:

Выполнить: z = x + y

80C196 (CISC)

z, x и y являются ячейками памяти, к которым производится прямая адресация

```
x           DW      1
y           DW      1
z           DW      1

            ADD     z, x, y      ;5 тактов - прерывание невозможно
```

C166S V2 (RISC)

z, x и y являются ячейками памяти, Rw — регистр общего назначения (GPR)

```
x           DW      1
y           DW      1
z           DW      1

            MOV     Rw, x              ; 2 такта
                                       ; * Здесь возможно прерывание
            ADD     Rw, y              ; 2 такта
                                       ; * Здесь возможно прерывание
            MOV     z, Rw              ; 2 такта
                                       ; -----
                                       ; 6 тактов
```

[1)] Используется режим прямой адресации

При использовании RISC-процессора требуется один дополнительный такт.

Однако, если переменные назначаются с учетом того, что программа будет запущена на RISC-процессоре:

x и y — ячейки памяти, z — регистр общего назначения (GPR)

```
x           DW        1
y           DW        1

z           LIT       'R0'   ; z присваивается GPR R0 путем определения LIT
            MOV       z, x   ; 2 такта
                             ; * Здесь возможно прерывание
            ADD       z,  x  ; 2 такта
                             ; -----
                             ; 4 такта
```

По сравнению с CISC экономится один такт. При этом, приведенный выше пример представляет собой худший вариант для RISC- и лучший вариант для CISC-систем.

Для выполнения обычной 2-операндовой команды ADD CISC-процессор использует четыре такта, а RISC — только два, что даёт 50% экономии.

Назначение имён переменных всем регистрам общего пользования, вероятно, имеет смысл в контексте реальной программы.

Этот простой пример показывает, как, зная методы RISC-программирования, можно увеличить производительность системы.

1.4. Связь RISC-процессора с внешними устройствами

В собранных на базе RISC-процессора рабочих станциях или настольных компьютерах часто используется суперскалярная архитектура, которая обеспечивает параллельное выполнение команд. Это достигается за счёт наличия отдельных модулей сложения, умножения, сдвига, а также других специализированных блоков, каждый из которых имеет свой собственный конвейер.

Ни один из RISC-микроконтроллеров пока не может обеспечить ничего подобного (разве что, с большой натяжкой, 32-разрядный процессор Tricore), но некий аналог такой структуры используется при обслуживании встроенных периферийных устройств, например, аналогово-цифровых преобразователей (АЦП).

Вот типичная для обычных (CISC) микроконтроллеров ситуация — когда некое повторяющееся событие требует от центрального процессора загрузки и выгрузки данных. Как правило, АЦП периодически считывает сигналы с нескольких каналов, вызывая прерывание по завершении этой операции или просто ожидая, пока центральный процессор опросит его состояние. В конечном итоге,