МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук
Кафедра «Прикладной математики и информатики»

Утверждено на заседании кафедры
«Информационная безопасность»
« 14 » января 2020 г., протокол № 6

Заведующий кафедрой
_____ В.И. Иванов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к самостоятельной работе студента
по дисциплине (модулю)
«*Операционные системы*»

основной профессиональной образовательной программы
высшего образования – программы бакалавриата

по направлению подготовки
01.03.02 Прикладная математика и информатика

с направленностью (профилем)
Прикладная математика и информатика

Форма обучения: очная

Идентификационный номер образовательной программы: 010302-01-20

Тула 2020 год

Разработчик методических указаний

Скобельцын С.А., доцент каф. ПМиИ, к.ф.-м.н. _____

(*ФИО, должность, ученая степень, ученое звание*) *(подпись*)

Тема 1. Архитектура фон Неймана

Теоретические сведения

Основы учения об архитектуре вычислительных машин были заложены Джон фон Нейманом.

Основы учения об архитектуре ЭВМ заложил выдающийся американский математик Джон фон Нейман.

Первая ЭВМ "Эниак" была создана в США в 1946 г. В группу создателей входил фон Нейман, который и предложил основные принципы построения ЭВМ:

- переход к двоичной системе счисления для представления информации
- принцип хранимой программы: программу вычислений предлагалось помещать в запоминающем устройстве, что обеспечивало бы автоматический режим выполнения команд и, как следствие, увеличение быстродействия.

Заметим, что ранее все вычислительные машины хранили обрабатываемые числа в десятичном виде, а программы задавались путём установки перемычек на специальной коммутационной панели.)

Нейман первым предложил хранение программы в числовой форме – в виде набора нулей и единиц – причём в той же памяти, что и обрабатываемые ею числа.

В результате реализации идей фон Неймана была создана классическая архитектура ЭВМ.

Принципы фон Неймана

Использование двоичной системы счисления в вычислительных машинах. Преимущество перед десятичной системой счисления заключается в том, что устройства можно делать достаточно простыми, арифметические и логические операции в двоичной системе счисления также выполняются достаточно просто.

Программное управление ЭВМ. Работа ЭВМ контролируется программой, состоящей из набора команд. Команды выполняются последовательно друг за другом. Созданием машины с хранимой в памяти программой было положено начало тому, что мы сегодня называем программированием.

Память компьютера используется не только для хранения данных, но и программ. При этом и команды программы и данные кодируются в двоичной системе счисления, т.е. их способ записи одинаков. Поэтому в определенных ситуациях над командами можно выполнять те же действия, что и над данными.

Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы. В любой момент можно обратиться к любой ячейке памяти по ее адресу. Этот принцип открыл возможность использовать переменные в программировании.

Возможность условного перехода в процессе выполнения программы. Не смотря на то, что команды выполняются последовательно, в программах можно реализовать возможность перехода к любому участку кода.

Таким образом, формально перечень принципов фон Неймана может быть приведен в следующем виде:

- Принцип использования двоичной системы счисления для представления данных и команд.
- Принцип программного управления.
- Программа состоит из набора команд, которые выполняются процессором друг за другом в определенной последовательности.
- Принцип однородности памяти.
- Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления - чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными.
- Принцип адресуемости памяти.
- Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.
- Принцип последовательного программного управления
- Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.
- Принцип условного перехода (сам принцип был сформулирован задолго до фон Неймана Адой Лавлейз и Чарльзом Бэббиджем, однако Нейман добавил его в общую архитектуру).

Компьютеры, построенные на этих принципах, относят к типу фон-неймановских

Архитектура компьютера фон Неймана

Компьютер состоит из нескольких основных устройств (арифметико-логическое устройство, управляющее устройство, память, внешняя память, устройства ввода и вывода).

Рис. 1. Архитектура компьютера фон Неймана

Арифметико-логическое устройство - выполняет логические и арифметические действия, необходимые для переработки информации, хранящейся в памяти.

Управляющее устройство - обеспечивает управление и контроль всех устройств компьютера (управляющие сигналы указаны пунктирными стрелками).

Данные, которые хранятся в запоминающем устройстве, представлены в двоичной форме.

Программа, которая задает работу компьютера, и данные хранятся в одном и том же запоминающем устройстве.

Для ввода и вывода информации используются устройства ввода и вывода.

Контрольные вопросы

1) Основные идеи положенные в основу структуры ЭВМ фон Нейманом.
2) Принципы организации ЭВМ фон Неймана.
3) Структура ЭВМ по фон Нейману.
4) Состав центрального вычислительного устройства.
5) Назначение и функции арифметико-логического устройства.
6) Назначение и функции управляющего устройства процессора.
7) Назначение и организация памяти.
8) Порядок выполнения программы.
9) Порядок ввода и вывода информации.

Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

Тема 2. Архитектура микропроцессора

Теоретические сведения

Центральный микропроцессор (небольшая микросхема, выполняющая все вычисления и обработку информации) – это ядро ПК. В компьютерах типа IBM PC используются микропроцессоры фирмы Intel и совместимые с ними микропроцессоры других фирм.

Компоненты микропроцессора:

- Устройство управления – управляет всеми устройствами ПК.
- АЛУ выполняет логические и арифметические операции.
- Регистры используются для хранения данных и адресов.
- Схема управления шиной и портами осуществляет подготовку устройств к обмену данными между микропроцессором и портом ввода-вывода, а также управляет шиной адреса и управления.

Основные характеристики процессора:

- Разрядность – число двоичных разрядов, одновременно обрабатываемых при выполнении одной команды. Большинство современных процессоров – это 32-разрядные процессоры, но выпускаются и 64-разрядные процессоры.
- Тактовая частота – количество циклов работы устройства за единицу времени. Чем выше тактовая частота, тем выше производительность.
- Наличие встроенного математического сопроцессора.
- Наличие и размер кэш- памяти.

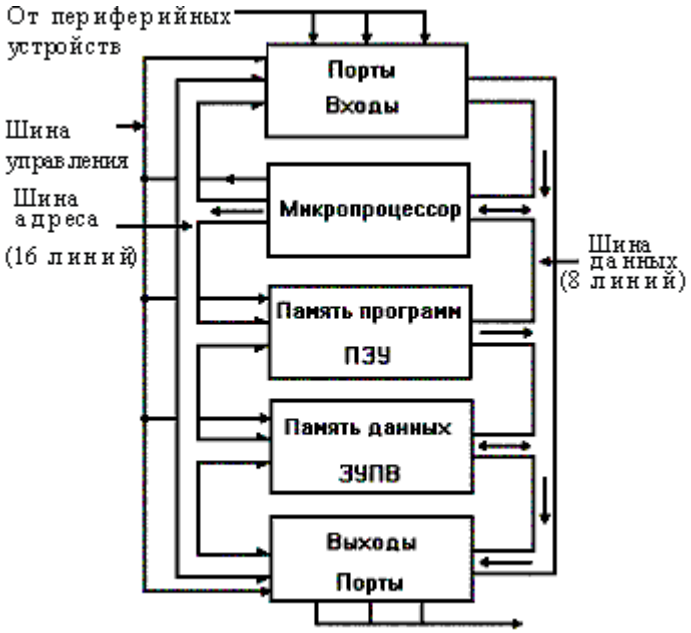

Рис. 1. Архитектура типового микропроцессора

Микропроцессор координирует работу всех устройств цифровой системы с помощью шины управления (ШУ). Помимо ШУ имеется 16-разрядная адресная шина (ША), которая служит для выбора определенной ячейки памяти, пор-

та ввода или порта вывода. По 8-разрядной информационной шине или шине данных (ШД) осуществляется двунаправленная пересылка данных к микропроцессору и от микропроцессора. Важно отметить, что МП может посылать информацию в память микроЭВМ или к одному из портов вывода, а также получать информацию из памяти или от одного из портов ввода.

Постоянное запоминающее устройство (ПЗУ) в микроЭВМ содержит некоторую программу (на практике программу инициализации ЭВМ). Программы могут быть загружены в запоминающее устройство с произвольной выборкой (ЗУПВ) и из внешнего запоминающего устройства (ВЗУ). Это программы пользователя.

Контрольные вопросы

1) Основные компоненты микропроцессора.
2) Структурная схема микропроцессора.
3) Взаимодействие микропроцессора с внешними устройствами.
4) Программирование микропроцессоров.
5) Система прерываний микропроцессора.

Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.

Тема 3. Особенности реализации некоторых вычислительных систем

Теоретические сведения

В семидесятых годах прошлого столетия проектирование и изготовление центральных процессоров было занятием, принципиально доступным каждому. Если какому-нибудь сотруднику, скажем, Стэндфордского университета приходила в голову интересная идея, он мог легко набрать команду, основать фирму, найти инвесторов и уже через год-два выбросить на рынок свои CPU.

Через тридцать с небольшим лет процессоры усложнились до такой степени, что разработка хоть сколько-нибудь быстрого по современным меркам кристалла требует огромной армии инженеров, гигантских инвестиций и целого моря времени. И дело здесь отнюдь не в тонких кремниевых технологиях и стоящих миллиарды долларов полупроводниковых фабриках - просто уже в восьмидесятых годах разработка принципиально нового CPU требовала двух-трех, а в девяностых - пяти-шести лет напряженной работы. Те же китайцы, даже располагая подробной информацией о тридцатилетней истории проектирования процессоров, владея новейшими фабриками по производству кремниевых кристаллов и не стремясь изобретать что-то новое, потратили на разработку собственного простейшего MIPS32-подобного процессора Godson (примерно эквивалентного по производительности i486) несколько лет. Это не считая еще одного года, когда новый кристалл отлаживали. А на разработку MIPS64-подобной архитектуры с приемлемой производительностью (~Pentium III 500-600 МГц) у китайской Академии наук ушло еще четыре года, - четыре года, потраченных только на то, чтобы повторить успех более чем двенадцатилетней давности.

Организация CISC процессоров

Так уж исторически сложилось, что поначалу совершенствование процессоров было направлено на то, чтобы сконструировать по возможности более функциональный компьютер, который позволил бы выполнять как можно больше разных инструкций. Во-первых, так было удобнее для программистов (компиляторы языков высокого уровня еще только начинали развиваться, и все по-настоящему важные программы писались на ассемблере), а во-вторых, использование сложных инструкций зачастую позволяло сильно сократить размеры написанной на ассемблере программы. А где меньше инструкций - меньше и затраченное на исполнение программы время.

Надо признать, что достигнутые на этом пути успехи действительно впечатляли - в последних версиях ЭВМ выразительность ассемблерного листинга зачастую не уступала выразительности программы, написанной на языке высокого уровня. Одной-единственной машинной инструкцией можно было сказать практически все, что угодно. К примеру, такие машины, как DEC VAX, аппаратно поддерживали инструкции "добавить элемент в очередь", "удалить элемент из очереди" и даже "провести интерполяцию полиномом" (!); а знаменитое

семейство процессоров Motorola 68k почти для всех инструкций поддерживало до двенадцати (!) режимов адресации памяти, вплоть до взятия в качестве аргумента инструкции "данных, записанных по адресу, записанному вон в том регистре, со смещением, записанным вот в этом регистре". Отсюда и общее название соответствующих архитектур: CISC - Complex Instruction Set Computers ("компьютеры с набором инструкций на все случаи жизни").

На практике это привело к тому, что подобные инструкции оказалось сложно не только выполнять, но и просто декодировать (выделять из машинного кода новую инструкцию и отправлять ее на исполнительные устройства). Чтобы машинный код CISC-компьютеров из-за сложных инструкций не разрастался до огромного размера, машинные инструкции в большинстве этих архитектур имели неоднородную структуру (разное расположение и размеры кода операции и ее операндов) и сильно отличающуюся длину (в x86, например, длина инструкций варьируется от 1 до 15 байт). Еще одной проблемой стало то, что при сохранении приемлемой сложности процессора многие инструкции оказалось принципиально невозможно выполнить "чисто аппаратно", и поздние CISC-процессоры были вынуждены обзавестись специальными блоками, которые "на лету" заменяли некоторые сложные команды на последовательности более простых. В результате все CISC-процессоры оказались весьма трудоемкими в проектировании и изготовлении. Но что самое печальное, к моменту расцвета CISC-архитектур стало ясно, что все эти конструкции изобретались в общем-то зря - исследования программного обеспечения того времени, проведенные IBM, наглядно показали, что даже программисты, пишущие на ассемблере, все эти "сверхвозможности" почти никогда не использовали, а компиляторы языков высокого уровня - и не пытались использовать.

Особенности архитектуры RISC процессоров

Точно так же, как когда-то CISC-процессоры проектировались под нужды asm-программистов, RISC проектировался в расчете на типовой код, генерируемый компиляторами. Для начала разработчики свели к минимуму набор инструкций и к абсолютному минимуму - количество режимов адресации памяти; упаковав все, что осталось, в простой и удобный для декодирования регулярный машинный код. В частности, в классическом варианте RISC из инструкций, обращающихся к оперативной памяти, оставлены только две (Load - загрузить данные в регистр и Store - сохранить данные из регистра; так называемая Load/Store-архитектура), и нет ни одной инструкции вроде вычисления синуса, косинуса или квадратного корня (их можно реализовать "вручную"), не говоря уже о более сложных[Канонический пример - инструкция INDEX, выполнявшаяся на VAX медленнее, чем вручную написанный цикл, выполняющий ровно тот же объем работы]. Да что там синус с косинусом - в некоторых RISC-процессорах пытались отказаться даже от трудно реализуемого аппаратного умножения и деления! Правда, до таких крайностей ни один коммерческий RISC, к счастью, не дошел, но как минимум две попытки (ранние варианты MIPS и SPARC) предприняты были.

Второе важное усовершенствование RISC-процессоров, целиком вытекающее из Load/Store-архитектуры, - увеличение числа GPR (регистров общего назначения). Варианты, у которых меньше шестнадцати GPR, - большая редкость, причем почти все эти регистры полностью равноправны, что позволяет компилятору свободно распоряжаться ими, сохраняя большую часть промежуточных данных именно там, а не в стеке или оперативной памяти. В некоторых архитектурах, типа SPARC, "регистровость" возведена в абсолют, в некоторых - оставлена на разумном уровне; однако почти любой RISC-процессор обладает куда большим набором регистров, чем даже самый продвинутый CISC. Для сравнения: в классическом x86 IA-32 всего восемь регистров общего назначения, причем каждому из них приписано то или иное "специальное назначение" (скажем, в ESP хранится указатель на стек) затрудняющее или делающее невозможным его использование.

Среди прочих усовершенствований, внесенных в RISC, - такие нетривиальные идеи, как условные инструкции ARM или режимы работы команд. Например, некий модификатор в архитектуре PowerPC и некоторых других показывает, должна ли инструкция выставлять по результатам своего выполнения определенные флаги, которые потом может использовать инструкция условного перехода, или не должна. Это позволяет разнести в пространстве инструкцию, выполняющую вычисление условия, и инструкцию собственно условного перехода - что в конвейерных архитектурах зачастую позволяет процессору не "гадать", будет совершен переход или нет, а сразу достоверно это знать. В классическом CISC они почти не встречаются, поскольку на момент разработки соответствующих наборов инструкций ценность этих решений была сомнительной (они выйдут на сцену только в конвейеризированных процессорах).

Использование конвейера

Идея конвейера, давным-давно предложенная Генри Фордом, состоит в том, что производительность цепочки последовательных действий определяется не сложностью этой цепочки, а лишь длительностью самой сложной операции. Иными словами, совершенно неважно, сколько человек занимаются производством автомобиля и как долго длится его изготовление в целом, - важно то, что если каждый человек в цепочке тратит, скажем, на свою операцию одну минуту, то с конвейера будет сходить один автомобиль в минуту, ни больше и ни меньше; независимо от того, сколько операций нужно совершить с отдельным автомобилем и сколько заняла бы его сборка одним человеком. Применительно к процессорам принцип конвейера означает, что если мы сумеем разбить выполнение машинной инструкции на несколько этапов, то тактовая частота (а вернее, скорость, с которой процессор забирает данные на исполнение и выдает результаты) будет обратно пропорциональна времени выполнения самого медленного этапа. Если это время удастся сделать достаточно малым (а чем больше этапов на конвейере, тем они короче), то мы сумеем резко повысить тактовую частоту, а значит, и производительность процессора.

Недостатки конвейера неочевидны, но, как обычно и бывает, из-за нескольких "мелочей" реализовать грамотно организованный конвейер совсем не просто.

Идея конвейера в процессоре очень красива на словах и в теории, однако реализовать ее даже в простом варианте чрезвычайно трудно. Но выгода от конвейеризации столь велика и несомненна, что приходится с этими трудностями мириться, ведь ничего лучшего до сих пор не придумано.

Организация суперскалярных процессоров

У полноценной конвейеризации, более или менее эффективно обходящей перечисленные выше проблемы, есть одно несомненное достоинство: она настолько сложна, что, единожды реализованная, позволяет легко построить на ее основе целый ряд интересных новшеств. Для начала заметим, что коль уж у нас есть очереди готовых к исполнению инструкций и мы знаем взаимозависимости между ними по данным, есть техника переименования регистров, позволяющая разным инструкциям одновременно задействовать одни и те же регистры для разных целей, и, наконец, есть надежно работающая система сброса конвейера, то мы можем запускать на исполняющие устройства сразу несколько инструкций (если они не зависят друг от друга и могут быть безболезненно выполнены одновременно) или переупорядочивать независящие друг от друга инструкции так, как сочтем нужным.

Процессоры, использующие первую технику, называются суперскалярными. К примеру, сугубо теоретически, по числу исполнительных устройств, Pentium 4 может выполнять семь инструкций за такт, а Athlon 64 - девять. Реальные цифры, конечно, гораздо скромнее и определяются трудностью полноценной загрузки всех исполнительных устройств, однако Pentium 4 все же способен исполнять в устоявшемся режиме две (при некоторых условиях - четыре), а Athlon 64 - три инструкции за такт, одновременно производя две (A64 - три) операции по адресации и выборке данных из оперативной памяти. Может показаться, что реализация суперскалярного процессора очень проста (достаточно со стадии schedule просто распределять инструкции по разным исполнительным устройствам), однако такой лобовой подход обычно упирается в то, что Front-end процессора перестает успевать загружать исполнительные блоки работой. Поэтому на практике хорошо сделанные суперскалярные архитектуры, подобные AMD K7/K8, приходится специально "затачивать" под суперскалярность.

Процессоры, использующие вторую технику, называются процессорами с внеочередным исполнением инструкций (Out-of-Order processors, OoO). Техника переупорядочивания инструкций замечательна тем, что резко ослабляет негативные эффекты от медленной оперативной памяти и от наличия зависимых цепочек инструкций. Если, например, инструкция A обратилась к оперативной памяти, а нужных данных в кэше не оказалось или если A занимается ожиданием результатов выполнения какой-то другой инструкции, то OoO-процессор сможет пропустить вперед другие инструкции, не зависящие от результатов

выполнения инструкции A. Кроме того, продвинутый планировщик OoO-процессора иногда может использоваться для реализации специфических деталей той или иной архитектуры - например, для спекулятивного исполнения по данным в случае Pentium 4 или одновременного исполнения нескольких веток программного кода в IA-64. Реализация OoO-процессоров не требует специальной оптимизации всего конвейера - это всего лишь усложнение схемы планировщиков, запускающих готовые к исполнению инструкции на исполнительные устройства в другом порядке, нежели они на планировщики поступили, плюс усложнение схем сброса конвейера и сохранения полученных результатов: результат выполнения прошедших вне очереди инструкций все равно должен сохраняться в последовательности, строго соответствующей расположению инструкций в изначальном коде[Это связано с тем, что если случится какая-то ошибка, то результаты выполнения запущенных вперед очереди инструкции придется аннулировать].

Архитектура процессоров. MIPS, Sparc, ARM и PowerPC

"Pentium 4" восьмидесятых годовMIPS (Microprocessor without Interlocked Pipeline Stages), "процессор без блокировок в конвейере". Основная идея, которой руководствовался Джон Хеннеси, со своей командой проектировавший в 1981 году первый MIPS-процессор, такова. Сильно упростив внутреннее устройство CPU и используя очень длинный (по тем временам) конвейер, можно получить процессор, не умеющий выполнять сравнительно сложные инструкции, зато работающий на очень высоких тактовых частотах, позволяющих скомпенсировать потери производительности на эмуляцию этих сложных инструкций. Изначально предполагалось, что MIPS-процессоры не будут аппаратно поддерживать даже операции умножения и деления - благодаря чему можно было обойтись без сложных в реализации блокировок конвейера[Процедура приостановки конвейера, инициируемая, когда процессору встречается "медленно выполняющаяся" операция, которую невозможно выполнить на какой-то из стадий за один такт. В процессорах тех времен такими операциями являлись умножение и деление; в современных процессорах блокировку может вызвать неудачное обращение в оперативную память, не находящуюся в кэше CPU] (отсюда и название архитектуры). Тем не менее даже в самых первых MIPS'ах блокировки в конвейере, равно как и аппаратные инструкции умножения и деления все-таки присутствовали - "в чистом виде" идея оказалась малопригодной для создания коммерческих процессоров.

В 1984 году Хеннеси с командой покинул Стэндфордский университет и основал компанию MIPS Computer Systems. В 1985 она выпустила первый 32-разрядный MIPS-процессор R2000; в 1988 году - гораздо более быстрый, работающий с виртуальной памятью и поддерживающий многопроцессорность R3000. R3000 стал первым по-настоящему коммерчески успешным MIPS-процессором и использовался в рабочих станциях Silicon Graphics. Кстати, вариант MIPS R3000A хорошо известен в народе как центральный процессор приставки Sony PlayStation

В 1991 году вышел первый 64-разрядный MIPS R4000, легший в основу целого ряда различных процессоров, выпускавшихся по лицензиям другими фирмами. R4000 оказался настолько важен для SGI, что она не колеблясь приобрела испытывавшую тогда финансовые затруднения MIPS Computer Systems и превратила эту компанию в собственное подразделение MIPS Technologies. Тогда же SGI начала продавать лицензии на производство MIPS-процессоров сторонним фирмам, которые взялись разрабатывать свои, улучшенные варианты R4000. Помимо всего прочего, начиная с R4600 и R4700 (разработка Quantum Effects Devices) MIPS-процессоры стали основой для знаменитых маршрутизаторов Cisco, являющихся сегодня неотъемлемой частью большинства крупных сетей, включая интернет. Использовались 64-разрядные MIPS-процессоры и в приставках: R4300 - в Nintendo 64, R5900 - в PlayStation 2.

Контрольные вопросы

1) Организация CISC процессоров.
2) Архитектурные особенности CISC процессоров.
3) Особенности архитектуры RISC процессоров.
4) Примеры реальных RISC-процессоров.
5) Организация конвейерных процессоров.
6) Организация суперскалярных процессоров.
7) Особенности архитектур процессоров MIPS, Sparc, ARM и PowerPC.

Список библиографических источников

1. Макарова, Н.В. Информатика : учебник для вузов / Н.В.Макарова [и др.]; под ред.Н.В.Макаровой.– 3-е изд., перераб.– М.: Финансы и статистика, 2007.– 768с.
2. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель; пер.с англ. под ред.С.М.Молявко.– 5-е изд.– Киев: Энтроп:Корона-Век, 2007.– 736с.
3. Абель, П. Ассемблер. Язык и программирование для IBM PC / П.Абель;пер.с англ.под ред.С.М.Молявко.- 5-е изд. - Киев : Энтроп:Корона-Век, 2007 .- 736с.
4. Юров, В.И. Assembler : практикум:учебное пособие для вузов / В.И.Юров .- 2-е изд. - М.[и др.] : Питер, 2006 .- 400с.
5. Жмакин, А.П. Архитектура ЭВМ : учеб.пособие для втузов / А.П.Жмакин .- СПб. : БХВ-Петербург, 2006 .- 320с.
6. Магда, Ю.С. Ассемблер для процессоров Intel Pentium / Ю.С. Магда.— М.[др.]: Питер, 2006.– 410с.