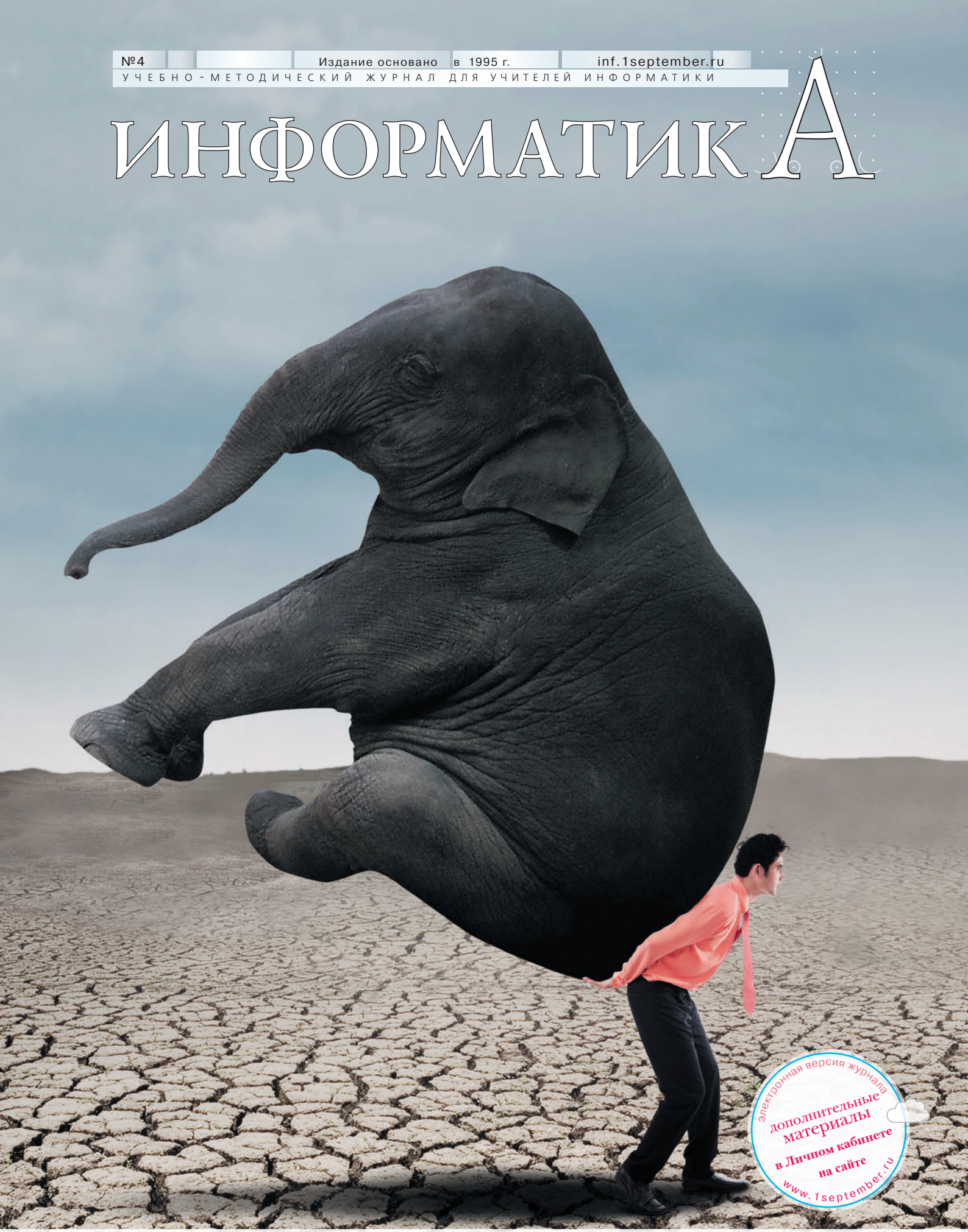
№4 Издание основано в 1995 г. inf.1september.ru
УЧЕБНО-МЕТОДИЧЕСКИЙ ЖУРНАЛ ДЛЯ УЧИТЕЛЕЙ ИНФОРМАТИКИ

ИНФОРМАТИКА



электронная версия журнала
дополнительные материалы в Личном кабинете на сайте www.1september.ru

издательский дом 1september.ru
Первое сентября
апрель 2015
ИНФОРМАТИКА Подписка на сайте www.1september.ru или по каталогу «Почта России»: 79066 — бумажная версия, 12684 — CD-версия

НА ОБЛОЖКЕ

ЗЕМЛЯ ГУЛЛИВЕРА

В этом номере опубликована очень любопытная статья-исследование, посвященная производительности многоядерных процессоров. Когда в редакции готовили эту статью к печати, вспомнился интересный факт не из компьютерного мира, а из мира людей. Речь идет о психологическом эксперименте Рингельмана, который был поставлен еще в начале прошлого века. Допустим, есть два человека, каждый из которых поднимает по 50 кг. Какой вес они поднимут вместе? Оказывается, совсем не 100! Много меньше. И дело тут вовсе не в “неудобно”, причины — чисто психологические. Может быть, и у ядер процессоров так же? (Шутка, конечно.)

В НОМЕРЕ

В ЛИЧНОМ КАБИНЕТЕ

Облачные технологии от Издательского дома “Первое сентября”

Уважаемые подписчики бумажной версии журнала!

Дополнительные материалы к номеру и электронная версия журнала находятся в вашем Личном кабинете на сайте www.1september.ru.

Для доступа к материалам воспользуйтесь, пожалуйста, кодом доступа, вложенным в № 1/2015.

Срок действия кода: с 1 января по 30 июня 2015 года.

Для активации кода:

- зайдите на сайт www.1september.ru;
- откройте Личный кабинет (создайте, если у вас его еще нет);
- введите код доступа и выберите свое издание.

Справки: podpiska@1september.ru или через службу поддержки на портале “Первого сентября”.

ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ

Презентации к статьям номера

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ
по каталогу “Почта России”: 79066 — бумажная версия, 12684 — электронная версия

http://inf.1september.ru

Учебно-методический журнал для учителей информатики
Основан в 1995 г.
Выходит один раз в месяц

РЕДАКЦИЯ:
гл. редактор С.Л. Островский
редакторы
Е.В. Андреева,
Д.М. Златопольский (редактор вкладки “В мир информатики”)
Дизайн макета И.Е. Лукьянов
верстка Н.И. Пронская
корректор Е.Л. Володина
секретарь Н.П. Медведева
Фото: фотобанк Shutterstock
Журнал распространяется по подписке
Цена свободная
Тираж 20 000 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

ИЗДАТЕЛЬСКИЙ ДОМ “ПЕРВОЕ СЕНТЯБРЯ”
Главный редактор: Артем Соловейчик (генеральный директор)
Коммерческая деятельность: Константин Шмарковский (финансовый директор)
Развитие, IT и координация проектов: Сергей Островский (исполнительный директор)
Реклама, конференции и техническое обеспечение Издательского дома: Павел Кузнецов
Производство: Станислав Савельев
Административно-хозяйственное обеспечение: Андрей Ушков
Педагогический университет: Валерия Арсланьян (ректор)

ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА “ПЕРВОЕ СЕНТЯБРЯ”
Английский язык – Е.Богданова
Библиотека в школе – О.Громова
Биология – Н.Иванова
География – и.о. А.Митрофанов
Дошкольное образование – Д.Тюттерин
Здоровье детей – Н.Сёмина
Информатика – С.Островский
Искусство – О.Волкова
История – А.Савельев
Классное руководство и воспитание школьников – М.Битянова
Литература – С.Волков
Математика – Л.Рослова
Начальная школа – М.Соловейчик
Немецкий язык – М.Бузоева
ОБЖ – А.Митрофанов
Русский язык – Л.Гончар
Спорт в школе – О.Леонтьева
Технология – А.Митрофанов
Управление школой – Е.Рачевский
Физика – Н.Козлова
Французский язык – Г.Чесновицкая
Химия – О.Блохина
Школа для родителей – Л.Печатникова
Школьный психолог – М.Чибисова

УЧРЕДИТЕЛЬ:
ООО “ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»”
Зарегистрировано ПИ № ФС77-58447 от 25.06.2014 в Роскомнадзоре
Подписано в печать:
по графику 20.02.2015,
фактически 20.02.2015
Заказ №
Отпечатано в ОАО “Первая Образцовая типография”
Филиал “Чеховский Печатный Двор”
ул. Полиграфистов, д. 1,
Московская область,
г. Чехов, 142300
Сайт: www.chpd.ru
E-mail: sales@chpk.ru
Факс: 8 (495) 988-63-76

АДРЕС ИЗДАТЕЛЯ:
ул. Киевская, д. 24,
Москва, 121165
Тел./факс: (499) 249-31-38

Отдел рекламы:
(499) 249-98-70
http://1september.ru

ИЗДАТЕЛЬСКАЯ ПОДПИСКА:
Телефон: (499) 249-47-58
E-mail: podpiska@1september.ru

“Дополненная” реальность — юридический аспект

Бабушка зовет внучку и говорит:
— Внученька, я завещаю тебе ферму. Там три дома, десять машин, тысяча кур, две тысячи свиней и два магазина. Это все тебе!
— Бабуля, а где эта ферма?
— ВКонтакте, внученька, ВКонтакте!
(*Современный анекдот*)

Сегодня компьютерная виртуальность все больше сливается с реальным миром, — вспомним, например, приснопамятные очки Google Glass, позволяющие “накладывать” компьютерные сообщения и изображения на то, что пользователь видит вокруг себя. Такое совмещение реального и виртуального называют “дополненной реальностью” и надеются, что эта технология может существенно изменить весь стиль жизни современного человека.

Впрочем, кое в чем такое совмещение реальности с виртуальностью нашу жизнь уже меняет. Во всяком случае — в сфере юридической. Так, различные виртуальные объекты (например, аккаунты в сетевых играх или даже отдельные “виртуальные вещи” из них) сегодня становятся объектами реальных судебных дел.

Один из примеров — уголовное дело, возбужденное полицией Караганды за… угон космолета. Обвиняемый — некий хакер, который взломал аккаунт одного из игроков сетевой игры DarkOrbit. “Космическому угонщику” грозит вполне реальный срок по статье 227 Уголовного кодекса Казахстана, в том числе потому, что в современных онлайн-играх в развитие своего персонажа, его транспорта и оружия игроки тратят не только свое время, но и вполне реальные деньги: целый ряд возможностей в таких играх предоставляется только платно.

Еще один аналогичный пример — уголовное дело, возбужденное в Брестской области (Беларусь) за угон виртуального танка. С соответствующим заявлением в милицию обратился 30-летний житель города Пружаны, у которого хакеры украли игровой аккаунт. Как выяснилось, на покупку и модернизацию своего танка ИС-8 игрок потратил не только уйму времени на ведение виртуальных боев, но и более 1 200 000 белорусских рублей. Сотрудники отдела по раскрытию преступлений в сфере высоких технологий УВД Брестского облисполкома связались с разработчиками игры и выяснили, что указанный танк продолжает “участвовать в сражениях”, но уже “под командованием” нового пользователя — из Московской области. В итоге украденный аккаунт был заблокирован, а затем возвращен пострадавшему пользователю.

Наконец, там же, в Беларуси игровой аккаунт был включен в состав имущества, подлежащего разделу при разводе. Причем, что интересно, причиной развода послужила сама эта компьютерная игра, которой муж слишком уж увлекался, обделяя вниманием свою жену. Во время раздела имущества специалисты, да и сами тяжущиеся стороны оценили игровой аккаунт в довольно солидную сумму. А поскольку жена сама иногда заходила в этот аккаунт поиграть, его сочли “совместно нажитым имуществом” и предложили мужу либо установить порядок совместного доступа к аккаунту, либо выплатить жене компенсацию. От совместного доступа мужчина отказался, предпочтя выплатить бывшей жене несколько сот долларов отступных.

По итогам этого судебного разбирательства юристы дали комментарий, что вход в игровой аккаунт — это право использовать компьютерную программу, а воспроизведение компьютерной программы — это имущественное право, которое является объектом гражданского права. К тому же деньги на игру тратились общие, из семейного бюджета.

Д.Ю. Усенков,
ст. н. с. Института информатизации РАО, Москва

Первый? Второй? Первый! И второй!

Д.П. Кириенко,
учитель школы № 179,
Москва

Информацию об использовании языка Python в школе (ссылки на дистрибутивы, литературу, материалы занятий 179-й школы, выступления на конференциях) можно найти на сайте автора [1]

Когда я в 2002 году пришел в школу учить детей информатике, первый вопрос, который я себе задал, — “какой язык программирования использовать для обучения?”. Конечной целью ставилась задача научить школьников программированию на C++, ибо школа была математической, а в научных расчетах без C/C++ не обойтись. Но я понимал, что путь изучения языка C++ тернист и труден, и вряд ли будет легок для школьников, которые никогда не занимались программированием, поэтому над выбором языка программирования действительно пришлось подумать. Против языка Pascal у меня были принципиальные возражения — например, то, что язык Pascal де-факто был представлен только коммерческой реализацией от компании Borland, не являющейся бесплатной и кросс-платформенной. Рассмотрев несколько вариантов, я выбрал язык Python, — хотя раньше я ничего про него не знал, но сразу же понял, что это то, что нужно. Свободный, кросс-платформенный, современный, настоящий промышленный язык программирования — но одновременно простой и элегантный. И с 2002 года использовал в школе язык Python.

Правда, через несколько лет работы с Python я от него на несколько лет отказался. Основные причины были в том, что Python не поддерживался на олимпиадах, а также в том, что я смотрел на Python как на сугубо учебный язык, промежуточный вариант перед изучением “серьезного” программирования на C++. И программы на Python я писал так, как я бы их писал на C++, просто меняя синтаксис языка. У школьников было примерно такое же отношение — узнав, что у нас будет изучаться язык Python, некоторые имеющие представление о программировании говорили: “Хотим Java!” или “Хотим C++!”.

Убрав Python с уроков, я не убрал его из своей души. Со временем я понял, что большинство программ “просто для себя” я пишу на языке Python, в моих глазах Python из “игрушечной” замены C++ стал настоящим, полезным языком программирования. Также постепенно (не без моего участия) Python стал появляться на олимпиадах, вышла третья версия языка Python, да и вообще упоминание этого языка уже не вызывало вопросов: “а что это такое?”. Поэтому лет пять назад я пересмотрел свое отношение к Python и его месту в школе и решил попробовать полностью перевести свой курс программирования на Python, не рассматривая его более как учебный язык на пути к освоению C++. Одновременно с этим я начал публиковать различные свои материалы, переработанные для языка Python, выступать на конференциях с рассказом о Python, заинтересовал многих из своих коллег, в итоге сейчас уже сложно кого-либо удивить самой идеей об обучении программированию в школе на языке Python. В 2014 году 179-ю школу окончили первые школьники, которые изучали программирование только на языке Python с 8-го класса.

Разумеется, рост популярности языка Python привел и к появлению различных дискуссий, причем мне доводилось спорить как о довольно малосерьезных вещах типа правильного написания названия языка “Питон”, “Пайтон” или “Python”, так и отвечать оппонентам, которые категорически считали невозможным использование Python в школе для обучения. При этом, по моим наблюдениям, возражения против Python в основном высказывают те, кто ни разу не пробовал использовать его на уроках. А те коллеги, которые попробовали хотя бы раз учить на языке Python, больше не хотели возвращаться ни к Pascal, ни тем более к языку C. Например, в Летней компьютерной школе (lksh.ru) после пары лет экспериментов решено было перевести на Python учебные программы всех младших параллелей — это решало сразу несколько методических проблем.

Хороший обзор языка Python с точки зрения использования его в школе дал К.Ю. Поляков в сентябрьском номере “Информатики” [2]. В этой статье достаточно взвешенно и объективно изложены плюсы и минусы языка, однако с выводами автора мне трудно согласиться. Посмотрим подробней на выводы К.Ю. Полякова, изложенные в конце статьи, приведу их целиком.

> 1. “Python предоставляет программисту много свободы, перекладывая на него всю ответственность за возможные ошибки. Такой подход часто полезен для опытных программистов, но может приводить к серьезным проблемам, которые проявляются только во время выполнения некорректных операторов и “вылавливаются” с трудом. В некоторых случаях обычные опечатки могут вызвать логические ошибки в программе, потому что не обнаруживаются транслятором. Одной из причин этого является динамическая типизация (см. примеры выше). Поэтому программы на Python требуют более тщательного тестирования”.

Замечания К.Ю. Полякова во многом верны. Несомненно, в языке Python есть свои тонкости и “подводные камни”, но в целом их достаточно мало и язык гораздо более “защищен” от ошибок программиста, чем, например, язык C, в котором можно написать `if (x = 2)` или `(a < b < c)`, и это не будет ошибкой, но приведет к весьма неожиданному поведению программы. А типизация в Python хоть и является динамической, но контролируется интерпретатором достаточно строго (так и называется — строгая динамическая типизация), например, выполнить операцию сложения для числа и строки нельзя. По своему опыту знаю, что при использовании языка C/C++ у школьников на отладку программ, связанную со случайными ошибками в коде, уходит куда больше времени, а обнаружение этих ошибок значительно более затруднительно, чем при использовании Python. И столь типичные ошибки в языке C/C++, как отсутствие инициализации переменных или выход за границу массива, в Python просто невозможны. Зачастую школьники сталкиваются с проблемой, что программа по-разному работает на разных компьютерах, например, у школьника программа выдает один результат, а в тестирующей системе — другой. В C/C++, да и в Pascal, это, как правило, связано с перечисленными выше ошибками (отсутствие инициализации переменных или выход за границы массива), в Python такие ошибки исключены, поэтому ситуация, когда программа по-разному работает на разных компьютерах, практически невозможна. А найти и исправить такие ошибки в C/C++ или Pascal школьнику неимоверно тяжело, так как на его компьютере программа выдает правильный результат! И тогда без помощи преподавателя, который сможет найти ошибку и сказать “вот здесь переменной sum ты забыл присвоить значение 0, но тебе повезло, и она оказалась равна 0, но вообще так делать нельзя”, школьник практически не сможет обойтись.

Относительную простоту в отладке учебных программ отметила и Е.В. Андреева, которая несколько лет назад решила использовать Python для обучения восьмиклассников в школе “Интеллектуал”, при этом не зная языка, — она изучала его одновременно со школьниками. И ни разу у нее не возникло ситуации, когда она не смогла бы разобраться в каких-то непонятных для школьника местах, не могла бы объяснить поведение интерпретатора. В случае с программами на C/C++ такое скорее всего было бы просто невозможно.

> 2. “В настоящее время существуют две версии Python — 2.x (устаревшая) и 3.x (перспективная), несовместимые между собой. Перевод программ, написанных в версии 2, в версию 3 настолько затруднителен и чреват ошибками, что было решено поддерживать обе версии параллельно”.

Это замечание вообще никакого отношения к школе не имеет и важно только для разработ-

чиков программ, у которых накоплен большой объем программного кода, написанного на Python версии 2. А в школе все просто — если вы хотите попробовать Python, нужно использовать версию 3, и никаких трудностей у вас не будет. В версии 3 языка исправлены многие недостатки версии 2, поэтому в школьном курсе именно на версии 3 и нужно остановить свой выбор.

3. “Python — интерпретируемый язык, поэтому для выполнения программ нужен интерпретатор. Скорость выполнения программ на Python может быть в 100 раз ниже, чем скорость выполнения программ на языке C, при этом Python-программы расходуют больше памяти”.

Интерпретатор языка Python есть в любой современной операционной системе Linux или OS X, в системе Windows он легко устанавливается. Замечание про скорость работы верное, но в школе это практически неважно, все учебные задачи типа “найдите наибольшее из данных чисел” легко решаются на Python.

По сути, скорость работы языка Python критична (для школьника) в двух случаях — на олимпиаде его программа может не уложиться в допустимое ограничение по времени работы, или при реализации какого-либо проекта, критичного к быстродействию, например, связанного с обработкой изображений или больших объемов данных.

Оба эти примера (олимпиады и серьезные проекты) на самом деле важны только для небольшого числа школьников. Быстродействие языка Python становится критичным только на олимпиадах уровня заключительного этапа всероссийской олимпиады по информатике. Даже на региональном этапе пусть не все задачи можно выполнить на полный балл, но заведомо можно стать призером регионального этапа и пройти на заключительный, решая задачи только на Python. А некоторые олимпиадные задачи зачастую проще решать именно на Python, поскольку на разработку, написание и отладку кода уходит гораздо меньше времени. Таким образом, эта проблема актуальна только для тех немногих школьников (250 на всю страну), которые выходят на заключительный этап всероссийской олимпиады, но эти школьники уже, как правило, понимают достоинства и недостатки различных языков программирования и при необходимости освоят и язык C++ практически самостоятельно, если им действительно важна победа в заключительном этапе олимпиады.

Наоборот, на олимпиадах начального уровня, где не требуется реализовывать наиболее эффективные алгоритмы решения задачи, программирующие на языке Python имеют преимущества по сравнению с остальными именно за счет удобства программирования, богатых возможностей языка, лаконичности кода и простоты программ. Например, на муниципальном этапе всероссийской олимпиады школьников в Москве в декабре 2014 года из примерно 2000 участников 9–11-х классов Python использовали около 400, то есть 20%, Python при этом уверенно занимает третье место (Pascal — 49%, C++ — 24%). А на региональном этапе в Москве Python уже значительно опережает Pascal по числу участников (C++ — 51%, Python — 37%, Pascal — 21%).

Аналогично и в проектной работе — возможно, для единичных сложных проектов Python не подойдет, но из-за того, что библиотеки Python содержат массу возможностей для web-программирования, разработки GUI, обработки изображений, работы с базами данных и т.д., он может быть использован для большинства проектов школьников.

4. “В Python используется неклассическая объектная модель: все члены класса — общедоступные, нельзя сделать закрытые (private) или защищенные (protected) поля и методы. Тем самым фактически невозможна строгая инкапсуляция”.

Это также верное замечание, но оно тоже относится скорее к вопросу разработки крупных проектов в промышленном программировании, а не к Python с точки зрения учителя. Если в школе и заходит речь об объектно ориентированном программировании, то обычно ограничиваются только знакомством с основными принципами, и здесь простейших возможностей Python по инкапсуляции данных, когда имя поля (атрибута) класса начинается с двух символов подчеркивания, вполне достаточно. С использованием этой особенности языка Python вполне можно объяснить идею инкапсуляции.

5. “До сегодняшнего дня не создано надежных RAD-систем для разработки программ на Python с графическим интерфейсом, которые можно было бы использовать в учебных и прикладных задачах”.

Отчасти причина этого в том, что разработка GUI на Python столь проста (сам К.Ю. Поляков приводит примеры с использованием tkinter, автор же предпочитает для разработки GUI использовать библиотеку PySide — аналог PyQt), что никакие RAD-системы, которые “упрощают” процесс создания GUI, не нужны, графические приложения легко создаются и без RAD-систем.

6. “Python хорош для профессиональных программистов, но его использование в качестве первого языка программирования может быть неудачным решением. Как признаются учителя, преподающие на Python, те, кто учился программировать на Python, не хотят переходить на другие (более низкоуровневые) языки. Научив школьников сортировать массивы вызовом метода sort, сложно потом объяснить, зачем написаны целые тома об алгоритмах сортировки. А это может привести к появлению плеяды “программистов-только-на-Python”, не готовых к преодолению дополнительных ограничений ради повышения эффективности программы.

Фактически учитель попадает в ситуацию, которая хорошо описывается фразой “В Python такие возможности есть, но учить так нельзя!” (Е.В. Андреева). В то же время, было бы полезным изучение Python в качестве второго языка программирования в классах с углубленным уровнем изучения информатики (например, после Паскаля или С)”.

В этом абзаце на самом деле заключены наиболее серьезные возражения против языка Python в школе, которые приходится слышать от разных людей, поэтому он был процитирован целиком. Кроме того, я бы добавил в число основных недостатков языка Python и следующие моменты, высказанные собеседниками в различных дискуссиях:

1. Python — интерпретируемый язык с динамической типизацией, отсутствие необходимости объявлять переменные и описывать их типы не способствует развитию четкого алгоритмического мышления.

2. Используемые в Python высокоуровневые типы данных (например, длинная целочисленная арифметика или списки вместо массивов) и функции не позволяют понять идеи организации памяти компьютера, например, не позволяют понять, что массив из 100 переменных типа int представляет собой выделенные 400 байт в памяти, представляющие собой 100 значений типа int, каждое из которых занимает 4 байта.

3. Из-за легкости программирования и наличия большого числа стандартных функций (например, поиска минимума, сортировки, поиска элемента) у программистов на языке Python не сформируется умение реализовывать эти алгоритмы, а вместо этого они будут считать, что вся сортировка массива заключается в вызове функции sort и понимать, как она работает, не нужно.

Выводы из вышесказанного делаются разные — некоторые считают, что из-за этого вообще не следует изучать Python в школе, другие, как К.Ю. Поляков, считают, что сначала школьники должны изучить язык Pascal или С (то есть строго типизируемый компилируемый язык, позволяющий на достаточно низком уровне работать с памятью), и только после этого можно на профильном уровне изучения информатики осваивать Python как второй язык программирования.

При этом все перечисленные недостатки языка Python проистекают из его архитектуры, и все эти недостатки свойственны всем современным языкам программирования, чьи имена сейчас “на слуху”, — PHP, Ruby, Perl. Все эти языки были созданы в последние 20–25 лет и являются именно такими динамическими интерпретируемыми языками программирования.

А Pascal и С были созданы на 20 лет раньше, в 1970-х годах, это — строго типизируемые компилируемые языки программирования. Язык С++ хоть и моложе С, но сохраняет основные архитектурные особенности языка С, например, наличие прямого доступа к памяти.

Между тем ни одного классического компилируемого языка программирования (с исключительно строгой статической типизацией, создающих настоящий машинный исполняемый код, каковыми являются Pascal или С), который стал бы широко популярен, за последние 20–30 лет создано не было. Такие языки, как Java и C#, также занимающие важное место в современном промышленном программировании, на самом деле являются “гибридными” языками — программа компилируется в промежуточный байт-код, который затем интерпретируется после запуска программы (для Java таким интерпретатором является java virtual machine, для C# — .NET Framework). Запустить программу на языке Java или C# на компьютере, на котором не установлена java virtual machine или .NET, невозможно, поэтому программы, написанные на этих языках, тоже не являются в полной степени переносимыми.

Таким образом, за последние 20–30 лет были созданы исключительно интерпретируемые языки программирования либо языки “гибридного” вида, как Java или C#. При этом у всех этих языков программирования есть общие черты, как, например, автоматическая сборка мусора (которая до сих пор не реализована в языке С++ из-за наличия в языке возможностей прямого доступа к памяти через указатели). То есть тенденция развития современных языков программирования — это именно интерпретируемые, или “гибридные”, языки, содержащие компилятор в промежуточный байт-код и интерпретатор байт-кода. Язык С/С++, по-видимому, еще долго будет оставаться практически безальтернативным вариантом для создания приложений, компилируемых непосредственно в машинный код.

Мир программирования и подходы к программированию меняются, программирование сейчас и 40 лет назад — разное. Сейчас программист может писать сайт на PHP или бухгалтерское приложение в системе 1С и при этом может совсем не уметь программировать на С++. Ну а все современные языки программирования стремятся сделать процесс про-

граммирования удобным. Например, встроенные алгоритмы сортировки есть во всех современных языках программирования (в том числе и в C++), сложно представить современный язык программирования без встроенной функции сортировки. Нет ее в классическом Pascal (а вот в современном Delphi есть), нет в старых бейсиках (а в Visual Basic есть), но это же не означает, что этими новыми языками нельзя пользоваться в школе, ибо в них присутствует встроенная сортировка?

Но, может быть, все-таки остановиться в школе на уже ставших "классическими" языках Pascal или C? Пусть школа учит "фундаменту", а все новомодные языки программирования, в которых сортировка пишется в одну строчку, в школе вредны?

Давайте зададим вопрос, зачем вообще нужно учить программированию в школе? Заглянем в документ — действующий стандарт среднего образования. Но конкретики в этом документе совсем мало, например, в нем написано лишь только "владение стандартными приемами написания на алгоритмическом языке программы для решения стандартной задачи с использованием основных конструкций программирования и отладки таких программ" (базовый уровень изучения информатики по программам среднего (полного) общего образования) или "овладение понятием сложности алгоритма, знание основных алгоритмов обработки числовой и текстовой информации, алгоритмов поиска и сортировки; владение универсальным языком программирования высокого уровня (по выбору), представлениями о базовых типах данных и структурах данных; умение использовать основные управляющие конструкции; владение навыками и опытом разработки программ в выбранной среде программирования, включая тестирование и отладку программ; владение элементарными навыками формализации прикладной задачи и документирования программ" (углубленный уровень). Не дает стандарт ответов на вопрос "зачем?", давайте искать ответы сами.

На базовом уровне изучения информатики необходимо показать школьнику программный принцип управления компьютером (компьютер — это устройство, управляемое программой, программа представляет собой последовательность инструкций, содержит такие алгоритмические конструкции, как ветвления и циклы), а также сформировать основы алгоритмического мышления и познакомить с самим процессом программирования, как сферой деятельности человека. Помимо общеобразовательной цели (современный человек должен понимать принцип работы компьютера, а умение формально составлять алгоритм будет полезно в любой сфере деятельности), здесь имеется еще одна важная цель — найти школьников, которым была бы интересна дальнейшая деятельность в сфере информационных технологий, чтобы вывести их на углубленное изучение информатики на профильном уровне или на программу дополнительного образования, чтобы дальнейшую свою деятельность, начиная с учебы в вузе, они связали с информационными технологиями.

Какой язык программирования нужен для этого? Простой и удобный, но одновременно — мощный и современный. Представьте себе школьника, который никогда ничего не знал о программировании, и ему нужно показать основы этого программирования и познакомить с простейшими алгоритмическими конструкциями. Казалось бы, любой язык для этого подойдет, но не лучше ли взять язык программирования, в котором количество всех "лишних" действий, необходимых для достижения результата, сведено к минимуму? Ведь даже написание слов "begin" и "end" в Pascal занимает время урока, а нужно ли это для достижения задачи — познакомить с идеей программирования? И Python, вероятно, является наилучшим выбором именно для начального обучения программированию — ничего проще из языков программирования общего назначения на сей момент не существует. Он гораздо лаконичней Pascal, и программирование на нем лишено необходимости возни с большим количеством технических вещей, что трудно для совсем начинающих. Новичку гораздо проще написать программу, если она состоит из пяти строчек, а не из пятнадцати. Алгоритмы при этом реализуются те же, а вот времени на написание и отладку кода уходит меньше. Например, рассмотрим процесс создания массива из 1000 элементов, заполненного нулями. На Python для этого нужна одна строчка:

```
a = [0] * 1000
```

А на Pascal — гораздо больше:

```
var a: array[1..1000] of integer;
...
for i := 1 to 1000 do
    a[i] := 0;
```

Результат — одинаковый, получается массив, заполненный нулями, но на Python это существенно короче. Возможно, скептики скажут, что вот нельзя так писать, как на Python, что школьник должен понимать, что массив — это непрерывный фрагмент памяти, что мы должны его объявить, то есть зарезервировать для него место, что мы должны его проинициализировать, заполнив его нулями... Но в стандарте образования нет слов про необходимость понимания принципов хранения массивов в памяти компьютера, есть слова про "владение приемами написания программы с использованием основных конструкций". Так что смысл программы для начинающего программиста не меняется, и даже скорее строка `a = [0] * 1000` лучше отражает то, что школьник хочет получить (хочу список из числа 0, повторенного 1000 раз), проще пишется и требует меньше времени на набор текста. А чем проще будет процесс создания простых программ, тем эффективней будет результат обучения и тем

скорее можно рассчитывать на развитие интереса к программированию, — чтобы появился интерес к программированию, программирование должно быть простым и понятным!

Итак, на мой вкус, для начального обучения программированию Python подходит лучше остальных языков программирования.

Python лучше, чем Basic, потому что Python столь же прост, как Basic, но при этом Python — современный промышленный язык программирования, используемый многими современными IT-компаниями.

Python лучше, чем Pascal, потому что программы на Python гораздо короче и проще, чем на Pascal (как правило, в 2–3 раза). Казалось бы, первый язык программирования должен быть дружественным к пользователю, но можно ли считать дружественным язык, в котором в цикле for индексная переменная может изменяться только на +1 или на –1, а для цикла с другим шагом нужно или составлять математическую формулу, или использовать цикл while (который занимает несколько строк кода)? Это, конечно, один частный случай, но в целом конструкцию языка Pascal с современной точки зрения сложно назвать удобной для программиста, а ведь первый язык должен быть именно удобным.

И я не вижу никаких плюсов в предложении сначала учить детей программированию на Pascal или C, чтобы они сначала прошли трудный путь, намучившись с неуловимыми выходами за границы массива или неинициализируемыми переменными, пусть пишут программы в три раза длиннее, чем это можно было бы сделать на Python, но ни в коем случае не показывать им, что программирование может быть простым и приятным!

Итак, на мой взгляд, все-таки начинающих нужно учить на языке Python. Но вспомним и о профильном уровне изучения информатики, где необходимо показать уже основы программирования как профессии. Я встречался с таким подходом к обучению программированию в довольно известных школах — сначала (примерно классе в 8-м) учим программированию на Basic, затем (классе в 9-м) — на Pascal, а потом (в углубленной группе или на факультативе) особо заинтересовавшихся — на C++. Но в этом случае изучение программирования по сути заменяется изучением языков программирования, интересно ли сначала изучать синтаксис циклов в Basic, затем — синтаксис циклов в Pascal и, наконец, в C++, решая одни и те же стандартные задачи? На профильном уровне имеет смысл углубляться в разные сферы программирования — изучать алгоритмы (с выходом на олимпиады), писать программы с графическим интерфейсом, сетевые приложения или web-сайты, изучать объектно ориентированное программирование, и для всего этого пригодится Python.

Слабые или незаинтересованные в программировании школьники могут остановиться на базовом курсе, получив общее представление о программировании. А вот сильные школьники могут двигаться дальше, и в их распоряжении будет современный универсальный язык программирования, реально используемый для разработки программного обеспечения в ведущих мировых IT-компаниях. На Python можно писать веб-сайты, компьютерные игры, GUI, клиент-серверные приложения, то есть практически все, что угодно. Python может открыть школьнику многогранный мир программирования, в котором каждый найдет себе что-то по своему вкусу; сложно найти столь же универсальный язык программирования, который был бы при этом столь же прост. Поэтому Python отлично подойдет не только как первый, но и как второй язык программирования, и вполне возможно организовать профильный уровень изучения программирования на языке Python на протяжении многих лет. В 179-й школе у меня были классы, на которых Python использовался на уроках в течение четырех лет, и за эти годы рассматривались самые разные темы — алгоритмы, объектно ориентированное программирование, вопросы представления целых и действительных чисел в памяти компьютера, разработка GUI, рисование фракталов, использование регулярных выражений, сетевое взаимодействие.

Вернемся к критике языка Python. Существенное возражение заключалось в том, что в языке много высокоуровневых вещей, и возникает соблазн их использовать, вместо того чтобы изучать, как это устроено. Но, как я уже писал, во всех современных языках программирования есть встроенные сортировки и другие сложные функции. И это проблема не языка программирования (сортировки и удобные большие стандартные библиотеки должны быть в любом современном языке программирования), а методическая — нужно ли учить использовать встроенную сортировку или нужно учить использовать функцию sort? Это и имела в виду Е.В. Андреева, говоря “В Python такие возможности есть, но учить так нельзя!” — нельзя подменять изучение программирования исключительно на изучение богатых возможностей языка программирования, так как это не несет развивающей функции и формирует навыки, малоприменимые при переходе на другие языки программирования.

Общий методический подход, который я использую в 179-й школе, такой — сначала нужно понять, как это работает, потом разрешается использовать соответствующую функцию языка. Например, сначала дается задача написать обмен значений двух переменных через вспомогательную переменную, потом я показываю, как это удобно делается при помощи кортежей: `(a, b) = (b, a)`. Сначала нуж-

но написать поиск максимума из двух-трех чисел, потом можно использовать функцию `max`. Когда написали программу поиска максимума в последовательности (списке), можно использовать функцию `max` для списка. Сначала написали сортировку сами, потом можно пользоваться функцией `sort` и т.д. Только после того, как нужный алгоритм реализован самостоятельно, разрешается использовать соответствующую стандартную функцию языка программирования. Но я настаиваю и на том, чтобы школьники умели использовать и стандартную функцию сортировки — для того они и даны в языке программирования. Если мы изучили сортировку и теперь у нас другие задачи, в которых сортировка — только вспомогательный инструмент, зачем тратить время на реализацию сортировки, любой профессиональный программист будет всегда использовать встроенную функцию сортировки.

Поэтому проблема не в том, есть ли в языке программирования `sort` или нет, а проблема в том, как пользоваться возможностями языка программирования. Каким-то школьникам достаточно объяснить, что такое "сортировка выбором" — она понятна и очень легко пишется на том же Python, а с кем-то можно обсуждать и эффективные алгоритмы сортировки, и наличие встроенной сортировки в языке этому не мешает. Например, в *Massachusetts Institute of Technology* (MIT) — флагмане вузовского IT-образования, все начальные курсы, посвященные программированию и алгоритмам, несколько лет назад перевели на Python.

Что же касается "сахарных" возможностей Python, таких, как встроенная длинная арифметика или корректность логических выражений вида `a < b < c`, то, может быть, нормально, что язык программирования стал освобождать программиста от заботы о выборе числового типа данных или переформулировки на языке программирования обычного математического условия. Это не просто удобно, а избавляет от большого числа ошибок, ибо удобство языка для разработчика было одним из основных принципов проектирования Python. Это естественная эволюция в программировании вообще, аналогичная переходу от программирования в машинных кодах к языкам высокого уровня в 60–70-х годах прошлого века. Ведь сейчас никого не удивишь записью обращения к элементу массива в виде `A[5] = -12`, где `A` — массив 32-битных целых чисел в программе на C. Эта запись удобна, и мало кто при этом задумывается, как это простое действие на самом деле реализовано на низком машинном уровне: нужно взять адрес массива `A` в памяти, прибавить к нему 20 байт (размер одного числа, умноженный на количество элементов) и записать в 4 байта, расположенных по этому адресу, число –12, записанное в дополнительном коде, значение –12 при компиляции должно быть переведено в дополнительный код и записано в таком виде в машинном коде программы.

Теперь перейдем к аргументу "А это может привести к появлению плеяды "программистов-только-на-Python", не готовых к преодолению дополнительных ограничений ради повышения эффективности программы". Я бы сформулировал эту проблему немного по-другому — использование Python в школе приведет к появлению плеяды программистов, плохо понимающих низкоуровневое устройство компьютера, операционной системы, принципы взаимодействия программы с памятью и т.д.

Например, если программисту на языке Python добавить один элемент в конец списка, то он пишет `A.append(0)`. Программисту на C для этого нужно при помощи функции `malloc` выделить новый блок памяти большего размера, нежели старый массив, скопировать содержимое старого массива в новый блок памяти, добавив одно значение в конец, удалить старый блок памяти. Все это — непростая рутинная работа, чреватая ошибками, должен ли школьник уметь это делать самостоятельно? Нужно ли школьников учить этому внутреннему устройству памяти компьютера, работе с указателями, должен ли любой современный программист разбираться во всех этих подробностях низкоуровневой работы с данными? С одной стороны, это фундаментальные основы устройства компьютера. Но с другой стороны, а для чего тогда в языке C++ появились динамические массивы (`vector`), увеличить размер которых можно одной командой, не задумываться при этом о том, почему многократное добавление по одному элементу в конец динамического массива не будет приводить к постоянному копированию всего массива при добавлении каждого нового элемента?

Программирование становится все более и более массовой профессией, программист может писать базы данных или web-сайты, такие программисты нужны в большом количестве. Но зачем программисту, который пишет web-сайт, нужно знать о подробностях работы с памятью и указателями на уровне операционной системы? Наоборот, чем больше он будет "изолирован" от этой работы, тем меньше ошибок будет в его программах, вот почему в современных языках программирования концепция указателя из языка C меняется на концепцию безопасных ссылок, не допускающих прямого доступа к памяти.

Несомненно, нужны и специалисты, которые будут знать и принципы работы с памятью, и досконально разбираться в вопросах сложности алгоритмов сортировки, но все-таки это вопросы, скорее, относящиеся уже к сфере высшего профессионального образования в области IT. А программисты, умеющие написать сайт на PHP или программу для бухучета в 1С, тоже нужны, и их нужно очень много. Таким программистам в целом необязательно разбираться в алгоритмах сортировки, им как раз необходимо уметь пользоваться стандартной функцией `sort` для любых задач и знать, что она умеет сортировать массивы

лучше, чем если бы они это написали самостоятельно (и уж совсем странно требовать от web-программиста знания алгоритмов быстрой сортировки; конечно, ничего плохого не будет в этом знании, но для работы ему это совершенно ненужно). Ведь никого же не возмущает возможность сортировки данных средствами электронной таблицы безо всякого понимания, как именно это устроено.

Система обучения программистов должна быть многоуровневой и включать в себя среднее профессиональное образование (на котором учат прикладным вещам и технологиями) и систему высшего образования (где как раз уместно обсуждать вопросы сложности алгоритмов сортировки), точно так же, как это есть во многих отраслях, например, в медицине: медик со средним специальным образованием выполняет стандартные медицинские процедуры, а медик с высшим образованием умеет назначать лечение, то есть анализировать причины и принимать решение. Так же и программисту со средним специальным образованием лучше уделять внимание прикладным вещам и технологиям (например, web-программист должен изучать такие сугубо прикладные вещи, как HTML и CSS) и иметь лишь общее представление о том, что бывают разные алгоритмы сортировки, а вот уже системный программист с высшим образованием должен, несомненно, разбираться в алгоритмах сортировки разной эффективности.

Вновь оглянемся в прошлое лет на 50 назад, когда стали появляться языки программирования высокого уровня. Наверняка подобная дискуссия шла и в те годы — допустимо ли отказаться от изучения программирования в машинных кодах или на ассемблере и перейти на высокоуровневые языки? В те годы программист должен был хорошо знать устройство компьютера, систему машинных команд. Дональд Кнут в первом томе “Искусства программирования” [3] (первое издание которого вышло в 1968 году) пишет, что перед ним стоял сложный выбор — использовать для записи алгоритмов в своей книге машинно-ориентированный язык или языки высокого уровня (Кнут называет их “алгебраическими”), например, ALGOL или FORTRAN. В итоге Кнут останавливается на машинно-ориентированном языке, приводя, в частности, такие аргументы: “тот, кто серьезно интересуется компьютерами, должен хорошо знать машинный язык, так как он лежит в основе работы компьютера”, “Новые алгебраические языки входят и выходят из моды приблизительно каждые пять лет, в то время как я пытаюсь говорить о «вечных истинах»” или “серьезный программист должен быть готов к тому, что в ходе работы ему придется сталкиваться с различными машинными языками”. Отчасти Кнут оказался прав в том, что наиболее популярные в те годы языки ALGOL и FORTRAN уже давно вышли из моды, и использование их в книге, посвященной алгоритмам, было бы неудачным решением, но и программирование на машинном языке уже давно стало неактуально для большинства программистов (сам Кнут признает, что после 1970 года практически не использовал машинный язык для собственных программ). Большинство современных программистов прекрасно обходится без машинного языка и ассемблера (который уже исчез из учебных планов большинства IT-специальностей в университетах), но от этого количество программистов нисколько не уменьшилось, а, наоборот, существенно возросло. Им совершенно не нужно программировать на ассемблере, и упрекнуть программиста в непрофессионализме только за то, что он не знаком с ассемблером, сейчас нельзя. Но и специалисты, знающие машинные коды и ассемблер, никуда не денутся, поскольку они и сейчас востребованны.

Похожая ситуация происходила и с распространением автомобилей. В начале и середине XX века водитель автомобиля должен был знать устройство автомобиля и сам производить довольно сложный ремонт. Любой водитель был в значительной степени механиком. Сейчас многие водители если и имеют общее представление об устройстве автомобиля, то уж точно никогда не занимаются его ремонтом, предпочитая выполнять все техническое обслуживание в автосервисе. Хорошо это или плохо? Это, на мой взгляд, неизбежно. Широкое распространение автомобилей приведет к тому, что у большинства водителей не будет глубоких познаний об устройстве автомобиля. А автомобили, в свою очередь, становятся удобней и надежней, поэтому использовать их можно и без специальных навыков.

Точно так же и с программированием — программирование становится все более распространенным, а сами языки программирования — все более удобными и простыми. Количество программистов растет и будет расти (хотя все равно в отрасли ощущается огромная нехватка квалифицированных кадров), при этом уровень глубокого понимания вещей будет снижаться и перекладываться на интерпретаторы, компиляторы или даже операционные системы. Это неизбежно, но это не повод отказываться от современных высокоуровневых языков программирования, точно так же, как никому в голову не придет отказываться от современных автомобилей только потому, что они стали надежней и удобней и не требуют специальных познаний для постоянного использования.

Литература

1. Информация про Python на сайте автора *http://www.179.ru/~dk/python.html*.

2. *Поляков К.Ю.* Язык Python глазами учителя. М.: Информатика, 2014, № 9.

3. *Дональд Кнут.* Искусство программирования, том 1. Основные алгоритмы. 3-е изд. М.: “Вильямс”, 2006.

Действительно ли процессор с двумя ядрами сосчитает любую задачу вдвое быстрее?

Мне кажется, у проектировщиков процессоров иссякли идеи, и они хотят переложить ответственность за невозможность повышать производительность компьютеров в соответствии с законом Мура на тех, кто создает программы.
Д.Кнут

Дружные сороки тигра одолеют[1].
Монгольская пословица

Е.А. Еремин,
г. Пермь

Введение

Для начала предлагаю обдумать такую аналогию: если один экскаватор выроет яму за 10 минут, то следует ли из этого, что 10 таких экскаваторов выроют такую же яму за одну минуту? Вы только представьте себе это впечатляющее зрелище: 10 экскаваторов стоят в кружок и дружно, не мешая друг другу, грузят землю на грузовики! (Стоящие, видимо, в этом же круге? Или сзади, чтобы экскаватору требовался разворот на 180 градусов?)

С трудом верится, не правда ли. Но тогда как быть с рекламой, которая упорно убеждает нас приобретать как можно больше процессорных ядер? А действительно ли два ядра обрабатывают данные вдвое быстрее, три ядра — втрое и т.д.? Давайте обсудим данную проблему и даже проведем практическую проверку данного вопроса с помощью

[1] В одном из старых номеров журнала “Наука и жизнь” эта пословица приводилась в “расширенной” редакции: “Дружные сороки могли бы победить тигра. Дело в том, что сороки никогда не бывают дружными”.

программы-имитатора, которую можно найти в приложении к журналу.

Начнем с того, что если у нас есть *N*-ядерный процессор и *N* разных, не связанных друг с другом задач, то выигрыш в производительности, несомненно, получится. В частности, обслуживание каждого из клиентов сервера можно передать собственному ядру. Случай настолько прозрачный и понятный, что и рассматривать его неинтересно.

Мы и не будем. Лучше обратимся к более близкой к повседневной практике ситуации. Представим себе типичный персональный компьютер, за которым сидит рядовой пользователь. Получится ли у него набрать для себя *N* сложных задач, перегружающих современный процессор, да еще и решать их ***одновременно***? Рискну высказать мнение, что большинству обычных пользователей такое, как правило, не требуется, да и способности человека к одновременному выполнению нескольких работ тоже ограниченны. Возможно, геймеры знают какой-нибудь подходящий пример (у них даже, кажется, есть такой термин — "тяжелые игры"), но я, честно, не вижу в каждодневной жизни такого примера.

Тогда остается рассмотреть наиболее правдоподобный вариант применения компьютера — проведение расчетов по единственной задаче. Это самый простой и в то же время однозначный и убедительный путь проверить гипотезу о пользе многоядерности. Поэтому темой данной статьи и выбрано **изучение возможностей ускорения вычислений за счет многоядерного (многопроцессорного) решения задачи**.

Подчеркнем, что мы воспользуемся методом моделирования, а для него нет большой разницы между несколькими ядрами в одном процессоре или несколькими соединенными между собой процессорами.

Часть 1. Модель

Модель однопроцессорного компьютера

Очень давно, когда школьная информатика только начиналась, были такие школьные комплекты учебной вычислительной техники (КУВТ), как БК-0010 и УКНЦ. Как сейчас помню, они прежде всего были приспособлены для изучения языка программирования BASIC, хотя возможности их были все же шире. Но все эти машины объединял процессор с замечательной по простоте и логичности системой команд [1]. Не случайно в самых первых школьных учебниках информатики [2, 3] основоположники курса выбрали этот процессор в качестве базы для изложения материала.

Увы, такой удобный для изучения процессор просуществовал недолго. Ушли в историю БК и УКНЦ, а вместе с ними ушел и он. На смену пришли гораздо более мощные персональные компьютеры IBM PC. Их процессор от фирмы Intel хотя и был построен на схожих базовых принципах, но в деталях оказался много сложнее. Если для первоначального освоения системы команд процессора PDP вполне достаточно было понять небольшой набор главных идей, то для Intel дополнительно требовалось запомнить множество "научно-медицинских" фактов по поводу конкретных инструкций. Пример одного из таких фактов: процессор Intel "всегда умножает регистр, имя которого вы указываете в инструкции, на регистр AX, и сохраняет ответ в паре регистров AX:DX" [4]. Именно так и никак по-другому; и даже не мечтайте одной командой перемножить BX и CX! И таких жестких правил приходится запоминать довольно много.

Методическим ответом на сложность "живого" процессора стало создание **учебных моделей ЭВМ**. В отечественной литературе первой такой моделью была, по-видимому, "Кроха" [5]. Существует целый ряд подобных моделей [6], в том числе и зарубежных. Но все они либо создавались раньше, чем воцарилось четвертое микропроцессорное поколение (и, как следствие, были на него не очень похожи), либо были слишком упрощенными, чтобы продемонстрировать все базовые принципы работы современных компьютеров.

Итак, получается следующая картина:

- существовал очень просто и логично устроенный процессор, но он больше не применяется;
- процессоры, используемые на практике, слишком сложны и неудобны для обучения;
- все процессоры, несмотря на существенную разницу в деталях, имеют очень похожие общие принципы (именно их и надо изучать! [7, 8]);
- существующие учебные модели компьютера обладают недостаточными возможностями для того, чтобы объяснить наиболее важные особенности микропроцессоров.

Вывод очевиден: для удобства изучения главных идей устройства вычислительных машин надо было разработать новую учебную модель, похожую на "замечательный" PDP, причем, как водится при моделировании, оставить только самые существенные свойства моделируемого объекта.

В результате в 1997 году была разработана и реализована в виде учебного программного обеспечения модель под названием "**E97**" [9]. Ее применение в обучении информатике оказалось удачным, поэтому позднее описание "E97" было включено в известный учебник информатики [10].

Справедливости ради стоит заметить, что если бы Дональд Кнут предложил свою образовательную модель RISC-компьютера под названием "MMIX" [11] на десятилетие раньше, то, возможно, "E97" и не появился бы.

Немного о "E97"

Поскольку обсуждаемая в статье многопроцессорная модель будет самым непосредственным об-

разом опираться на модель “E97”, имеет смысл рассмотреть последнюю подробнее.

“E97” — это модель компьютера, содержащая процессор, память (ОЗУ и ПЗУ с полезными стандартными подпрограммами), а также виртуальный дисплей и клавиатуру, совпадающую с реальной клавиатурой вашего компьютера.

Процессор “E97” имеет разрядность 16 бит. Он способен обрабатывать 8-битные символы и 16-битные целые числа. Нетрудно догадаться, что это важно как минимум по двум причинам. Во-первых, модель может работать с двумя наиболее распространенными видами информации — числовой и текстовой (большинство моделей ограничиваются исключительно числами, и потому не совсем понятно, как же компьютер управляется с иными видами данных). Во-вторых, модель дает представление о том, как команды “переключаются” от чтения однобайтовых данных к многобайтовым.

Центральной частью программной модели процессора являются четыре 16-битных регистра с именами R0–R3. Очень важно, что в отличие от процессоров семейства Intel все они абсолютно равноправны: например, вы можете, ни секунды не задумываясь, сложить между собой любую пару регистров.

Упомянем, что, кроме указанных R0–R3, “E97” содержит еще три служебных регистра, поддерживающих выполнение программы: счетчик адреса команд (программный счетчик), регистр состояния и указатель стека.

Процессор умеет копировать информацию между различными регистрами и ячейками ОЗУ, а также обмениваться данными со специализированными ячейками внешних устройств — *портами*. Он может выполнять все арифметические и основные логические операции, а также обладает полным ассортиментом условных переходов (т.е. по всем шести математическим отношениям: равно, не равно, больше и т.д.); предусмотрен механизм вызова подпрограмм. Кроме этого, есть еще несколько менее очевидных команд, о которых на стадии начального знакомства можно не упоминать.

Любая команда кодируется в виде набора из шестнадцатеричных цифр. Очевидно, что 16-битная инструкция должна состоять из четырех hex-цифр. Очень удобно, что каждая из них имеет самостоятельное значение: код операции, два операнда (данные для обработки) и вспомогательная часть под названием “модификатор”. Система кодирования команд довольно проста; как показывает опыт, через пару занятий большинство студентов пишут коды нужных им команд, практически не задумываясь, а многие при этом почти не заглядывают в таблицы. Тем не менее ради простоты изложения в данной статье мы отступим от записи команд в виде hex-кода и воспользуемся еще более простой формой нотации — ассемблерной. Это позволит автору упростить изложение, а читателю — ускорить понимание основных идей.

Приведем примеры типичных команд “E97” (см. табл. 1).

Внимательное рассмотрение таблицы показывает, что входящие в команду операнды (регистры, константы, ячейки) расставляются без особых ограничений. Например, можно к любому выбранному регистру прибавить содержимое другого регистра, константу или содержимое ячейки — как того требует ваш алгоритм.

Константа обозначается символом “#”. Любой адрес ячейки заключается в круглые скобки.

Таблица 1. Примеры команд “E97”

mov R1,R3	R1 ==> R3	Содержимое R1 скопировать в R3
mov #1,R0	1 ==> R0	Константу 1 записать в R0
mov R2,(30)	R2 ==> (30)	Содержимое R2 скопировать в ячейку ОЗУ 30
mov R2,(R3)	R2 ==> (R3)	Содержимое R2 скопировать в ячейку, адрес (номер) которой берется из R3
mov (30),(32)	(30) ==> (32)	Содержимое ячейки 30 скопировать в ячейку 32
add R1,R2	R2 + R1 ==> R2	Сложить содержимое регистров; ответ в R2
add (50),R1	R1 + (50) ==> R1	Сложить содержимое ячейки 50 с R1; ответ в R1
add #1,R3	R3 + 1 ==> R3	Содержимое R3 увеличить на 1
sub R2,R1	R1 – R2 ==>R1	Из R1 вычесть содержимое R2; ответ в R1
sub #2,R3	R3 – 2 ==> R3	Содержимое R3 уменьшить на 2
cmp #0D,R0	Сравнить R0 с 0D	Сравнить содержимое R0 с константой 0D
and #0F,R1	R1 И 0F ==> R1	Логич. И между R1 и константой 0F; ответ в R1
not R0	НЕ R0	Выполнить логическое НЕ с содержимым R0
in p6,R0	Порт 6 ==> R0	Прочитать содержимое порта 6 в R0
out #2,p6	2 ==> порт 6	Вывести 2 в порт 6
bne L1	Переход по неравно	Если результат ≠0, перейти к команде с меткой L1
beq L2	Переход по равно	Если результат =0, перейти к команде с меткой L2
call P	Вызов подпрограммы	Перейти с возвратом к команде с меткой P
stop	Стоп	Закончить выполнение программы
nop	Нет операции	Ничего не делать (просто ждать)

Результат всегда записывается во второй операнд. Все операции над двумя операндами выполняются по единой схеме:

<операнд2> <действие> <операнд1> ==> ==> <операнд2>

Запомнив эти несложные рекомендации, уже можно приступать к написанию программ. Конкретный пример будет рассмотрен позднее в части 2 при анализе тестовой задачи.

Модель многопроцессорного компьютера

Для того чтобы выбрать наиболее простой и удобный подход к изучению параллельных вычислительных систем, необходимо предварительно познакомиться с основными способами, применяемыми на практике для их реализации. Ниже будут рассмотрены наиболее существенные альтернативные варианты организации таких систем.

Многомашинные и многопроцессорные системы

Существует несколько вариантов объединения вычислительных устройств для параллельных вычислений. Во-первых, можно распределить задачу между множеством компьютеров, которые соединены сетью; в результате получится многомашинный комплекс. Во-вторых, современные процессоры (начиная с Pentium и выше [12]) имеют встроенные аппаратные средства для организации многопроцессорных систем. Наконец, в-третьих, многие модели современных многоядерных процессоров фактически представляют собой несколько процессоров, собранных в одном корпусе.

В данной статье мы не будем рассматривать многомашинные комплексы, ограничившись взаимодействием процессоров в рамках одной вычислительной машины. Кроме того, мы не будем делать особой разницы между многопроцессорными системами и многоядерными процессорами, руководствуясь тем, что для моделирования функций межпроцессорного взаимодействия конкретные инженерные детали объединения процессоров особого значения не имеют.

Системы с одинаковыми/неодинаковыми процессорами

Большинство многоядерных процессоров состоят из нескольких одинаковых ядер, ни одно из которых не имеет специальных возможностей для управления другими ядрами. В частности, именно такие процессоры производит “законодатель мод” в мире микропроцессоров — фирма Intel. Часто такие системы называют *симметричными*. Нетрудно догадаться, что для производителей микроэлектроники такой подход очень удобен, поскольку он фактически переносит всю работу по распределению вычислений на “будущее” программное обеспечение (“как только вы напишете для нашего замечательного процессора программное обеспечение, вы немедленно получите мощный и производительный компьютер!”). Предполагается, что распределением работы будет управлять операционная система, которая станет еще сложнее, чем для процессора с единственным ядром.

Очевидно, что изучение такой организации параллельных вычислений весьма сложно, ибо требует не только понимания устройства самого многоядерного процессора, но и знания основ работы многозадачных операционных систем, причем далеко не самых простых. Данный вывод подтверждается изложенной в работе [13] методологией, предназначенной для изучения принципов работы многоядерных процессоров. Предлагаемый учебный программный комплекс, изображенный на *рис.* 1 в цитируемой статье, настолько сложен, что на момент ее написания реализован не полностью. В любом случае, если такой крупный вуз, как Московский государственный институт радиотехники, электроники и автоматики, может себе позволить подобную учебную систему, для остальных нужно что-нибудь более скромное и простое.

Существуют ли альтернативные решения? Да, существуют. Несколько фирм (Sony, IBM и Toshiba) предложили процессор принципиально иной организации — **Cell** (полное официальное название — ***C**ell **B**roadband **E**ngine* — CBE) [14]. Его планировалось внедрить в игровые приставки, но эта часть проекта оказалась неудачной. Тем не менее процессор реально был создан. Более того, IBM, основываясь на архитектуре CBE, выпустила более позднюю модель **PowerXCell 8i**, которая в несколько раз быстрее своей предшественницы обрабатывала вещественные числа двойной точности. Высокая производительность PowerXCell 8i привлекла внимание разработчиков суперкомпьютеров, и многие из них (включая находящийся в МГУ **Ломоносов**) содержат блоки на основе этих процессоров.

Существенным отличием архитектуры Cell является то, что он состоит из главного процессора (PPE — ***P**owerPC **P**rocessor **E**lement*, который, кстати говоря, сам является двухъядерным!) и восьми подчиненных (SPE — ***S**ynergistic **P**rocessor **E**lement*). Для нас сейчас существенным является то, что главный процессор распределяет вычислительную работу, так что вся задача решается непосредственно внутри процессора. Подчеркнем, что программу по распределению работ на выделенном центральном процессоре можно сделать гораздо проще, чем ОС для симметричного многоядерного процессора, особенно если речь идет о несложных учебных задачах. Именно это обстоятельство склоняет нас к выбору такой архитектуры для нашего учебного ПО.

Объективности ради следует заметить, что одна из главных причин неудачи проекта Cell применительно к игровым приставкам состояла в сложности программирования. Мы попытаемся преодолеть этот недостаток путем упрощения системы

команд для применяемых процессоров. Для простоты изучения можно дополнительно сделать системы команд главного и подчиненных процессоров одинаковыми. Более того, в качестве процессоров предлагается применить описанную выше учебную модель процессора “E97”, т.е. ту же самую модель, которая использовалась студентами ранее при изучении классической архитектуры с единственным процессором. Иными словами, изучение параллельной архитектуры оказывается естественным продолжением и углублением знаний, полученных студентами раннее при изучении классической компьютерной архитектуры.

Особо подчеркнем, что такая преемственность моделей очень удобна и полезна. Начнем с того, что при переходе от изучения обычного процессора к многоядерному студентам не приходится осваивать новую систему команд. Еще один аспект преемственности заключается в том, что для взаимодействия процессоров между собой использованы те же самые механизмы, что и при взаимодействии обычного одноядерного процессора с внешними устройствами. Благодаря такому решению изучение нового, многоядерного, процессора позволяет повторить и углубить следующие фундаментальные принципы ввода-вывода:

- обращение к внешним устройствам через порты;
- обмен данными через шину;
- механизм прямого доступа к памяти.

В итоге после анализа достоинств и недостатков одинаковости/неодинаковости процессоров в параллельной компьютерной системе мы приходим к некоторой смешанной модели. Она состоит из процессоров с одинаковой системой команд, но выполняющих разные функции: один главный, а остальные подчиненные.

Системы с общей/распределенной памятью

Известно [15, 16], что системы для параллельных вычислений бывают с *общей* и с *распределенной* памятью. В первом случае все процессоры берут данные из единой, общей для всех, памяти и туда же возвращают результаты. Теоретически это очень удобно, но, как мы увидим далее, на практике реализовать не так-то просто. Системы с раздельной памятью, где каждый процессор имеет собственную память, а обмен данными происходит непосредственно между самими процессорами, реализовать существенно легче. Зато появляются этапы передачи данных и результатов из одного блока памяти в другой, что сильно замедляет процесс обработки.

Поясним, почему могут возникать трудности в архитектуре с общей памятью. Начнем с парадоксального и несколько искусственного примера. Пусть в системе с общей памятью работают всего два процессора и в данный момент времени один из них (P1) “хочет” в соответствии с программой прибавить к ячейке с адресом S число 2, а второй (P2) к этой же самой ячейке — единицу (см. *рис.* 1). Для наглядности можно считать, что в ячейке S хранится счетчик некоторых объектов. Если предположить, что перед началом операции в рассматриваемой ячейке лежало число X, то какой результат получится после параллельного выполнения указанных команд? Логика требует, чтобы ответ был $X + 3$, т.е. P1 прибавил 2, а P2 добавил еще 1. Но если команды действительно выполняются одновременно, то с большой вероятностью сначала обе инструкции получат из памяти число X, а потом первый процессор “насчитает” $X + 2$, а второй — $X + 1$. Дальше все определяется тем, какой из них обратится к памяти для записи позднее, — таков и будет итоговый результат.

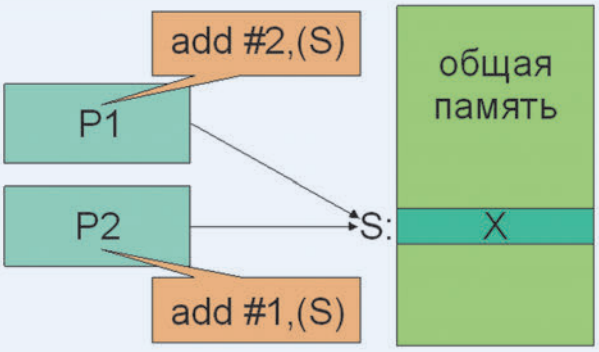


Рис. 1. Сложности работы с общей памятью

Даже если вычисления не так сильно взаимосвязаны и процессоры P1 и P2 будут работать с разными ячейками, все равно при обращении к общей памяти возникнет задержка: одна микросхема физически не в состоянии ответить обоим процессорам одновременно[2]. А значит, в системе появится очередь и ожидание. Таким образом, заманчивая модель с общей памятью на практике оказывается не такой быстрой, как интуитивно кажется.

Итак, каждый способ присоединения к памяти имеет свои достоинства и недостатки. Какой выбрать? В нашей модели без особых трудностей удалось реализовать оба механизма, так что студент может даже провести сравнение быстродействия этих двух архитектур. Несколько забегая вперед, заметим, что в соответствии с общей теорией механизм распределенной памяти действительно оказался медленнее.

Учебный компьютер “E14”

В результате проведенного выше анализа была создана и реализована в виде учебного программного обеспечения модель многопроцессорного компьютера “**E14**”. Она устроена следующим образом.

Процессоры

Компьютер “E14” состоит из пяти взаимосвязанных процессоров — одного **центрального**

[2] Для грубой оценки числа конфликтов в “E14” предусмотрен счетчик случаев, когда к общему ОЗУ одновременно обращаются несколько процессоров (он включается в меню Options).

(CPU) и четырех **периферийных (PPU1 – PPU4)**. Главный процессор распределяет вычислительные работы и руководит всеми периферийными процессорами, попутно обеспечивая необходимую синхронизацию совместных действий. При желании (“в свободное время”) его можно использовать для вычислений, как и остальные процессоры.

Все процессоры, образующие “E14”, полностью программно совместимы с учебной моделью “E97”.

Механизмы взаимодействия процессоров в “E14” поясняются на *рис.* 2.

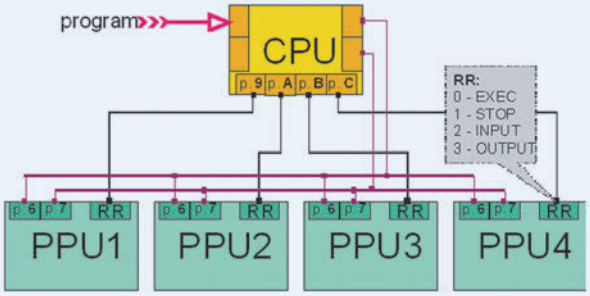


Рис. 2. Взаимодействие процессоров в “E14”

Опишем приведенный рисунок подробнее.

Начнем с периферийных процессоров PPU1 – PPU4. У каждого из них есть регистр режима RR. Целочисленное значение в этом регистре определяет текущее состояние процессора. Всего состояний четыре:

- RR = 0 — выполнение программы (EXEC);
- RR = 1 — ожидание (STOP);
- RR = 2 — работа по программе ввода данных из CPU (INPUT);
- RR = 3 — работа по программе вывода данных в CPU (OUTPUT).

Значение регистра RR задается из CPU. Для этого каждый RR связан с определенным портом в CPU: из рисунка видно, что регистр RR у PPU1 присоединен к порту CPU номер 9, у PPU2 — к порту A_{16} и т.д. Например, чтобы запустить процессор PPU3 на счет, центральному процессору достаточно просто записать нулевое значение в порт *B* командой **out #0,pB**. Кроме того, прочитав значение из соответствующего порта, CPU сможет узнать, в каком состоянии сейчас находится данный периферийный процессор. Это, например, важно, если CPU ожидает, когда PPU завершит вычисления.

Описанный механизм взаимодействия между процессорами нужен не только для запуска (остановки) PPU, но также позволяет организовать передачу данных непосредственно из ОЗУ CPU в ОЗУ PPU или наоборот. Как именно это происходит, мы рассмотрим подробнее, когда будем обсуждать устройство ОЗУ.

Есть и еще один канал обмена данными между процессорами: все процессоры подсоединены к общей шине (**межпроцессорной магистрали**). Мы видим, что общей линией объединены все порты 6 (*шина адреса*) и 7 (*шина данных*). В классическом варианте шины есть еще *шина управления*, но в “E14” она отсутствует: ее роль выполняют порты 9 — C у CPU. Весь обмен через шину происходит под управлением центрального процессора.

Как пользоваться шиной? Поясним на простых примерах.

Допустим, мы хотим передать число 5 из CPU в ячейку 18 памяти PPU1. Тогда CPU достаточно выполнить всего три команды:

```
out #18,p6      выставить адрес на шину
out #5,p7       выставить данные на шину
out #2,p9       перевести PPU1 в режим INPUT
```

Чтение содержимого 18-й ячейки памяти PPU1 в CPU будет чуть-чуть длиннее:

```
out #18,p6      выставить на шину адрес
out #3,p9       перевести PPU1 в режим OUTPUT
nop             ждем, пока PPU1
nop             выведет данные на шину
in p7,R0        читаем данные с шины в R0
```

Необходимость в ожидании возникает потому, что PPU1 потребуется выполнить две команды, чтобы поместить данные на шину: первая читает адрес из порта 6, а вторая копирует в порт 7 содержимое нужной ячейки. Только после этого центральный процессор сможет прочитать требуемое значение.

Заметим, что при включении режима INPUT или OUTPUT выбранный PPU переходит на одну из подпрограмм обмена, которые хранятся в его ПЗУ.

Память

Для удобства разделения памяти между процессорами в “E14” используется классический метод **страничной адресации**. Его механизм проиллюстрирован на *рис.* 3.

Рис. 3. Страничная адресация памяти

Суть страничного метода сводится к следующему. Каждый процессор имеет собственное (локальное) адресное пространство. В “E14” разрядность адреса установлена в 10 бит, что соответствует объему страницы ОЗУ, равному $100\ 0000\ 0000_2 = 400_{16} = 1024_{10}$ байт = 1 Кбайт. К локальному адресу в старшие биты добавляется **номер страницы**, который у каждого PPU хранится в специальном регистре **MPR** (***M**emory **P**age **R**egister*). В итоге мы получаем 10 + 3 = 13-битный физический адрес.

Особо подчеркнем, что благодаря страничному методу локальная адресация у PPU одинакова (адреса меняются от 0 до 3FF), так что программы для всех периферийных процессоров можно сделать идентичными.

В “E14” имеется пять (по числу процессоров) страниц ОЗУ с номерами от 0 до 4.

Заметим, что формально можно считать страницы независимыми (находящимися “в разных виртуальных микросхемах”), что разрешает одновременное обращение к каждой из них “своего” процессора. Это автоматически позволяет избежать конфликтов, связанных с одновременным обращением к памяти[3].

Работа со страницами ОЗУ организована следующим образом (*рис.* 4).

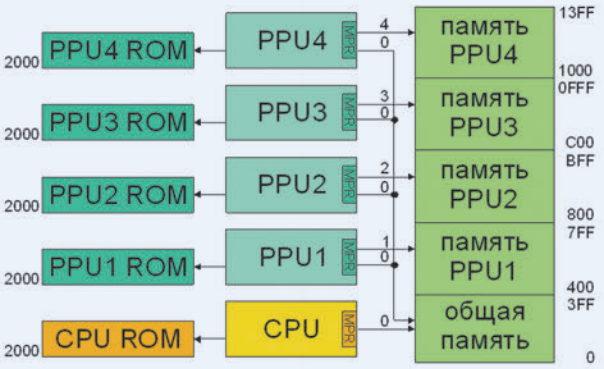


Рис. 4. Распределение памяти между процессорами

В центральной части рисунка вы видите процессоры, каждый из которых имеет регистр MPR. Регистр этот может принимать только два значения: 0 и целое число, равное номеру процессора (0 — CPU, 1 — PPU1, …, 4 — PPU4). Это означает, что каждый PPU может подключить либо свою собственную страницу, либо нулевую (общую). Очевидно, что CPU, номер которого равен 0, всегда подключен к нулевой странице. Предлагаемый механизм гарантирует каждому PPU собственную страницу ОЗУ, а также потенциальную возможность доступа к общей (нулевой) странице. Данная возможность позволяет изучать архитектуру с общей памятью, но она не используется в архитектуре с распределенной памятью.

Кроме ОЗУ, каждый процессор, начиная с локального адреса 2000, имеет ПЗУ. ПЗУ у всех PPU идентичны; в них записаны основные программы обмена данными между CPU и PPU. Благодаря этим программам PPU способен принимать программу и данные сразу после запуска. Еще ПЗУ обеспечивает переключение страниц ОЗУ, ибо невозможно, выполняя программу на странице ОЗУ, отключать эту страницу. Что касается CPU, то его ПЗУ содержит программу загрузки шестнадцатеричных кодов из файла и их рассылки в нужные процессоры.

Возможно, описанная система взаимодействия узлов показалась вам слишком сложной. В свое оправдание могу сказать, что используемые механизмы обмена не содержат по своей сути ничего нового по сравнению с одиночным компьютером: шины и порты там тоже имеются. Но зато у модели “E14” есть важное достоинство: она способна имитировать две параллельных архитектуры — с общей и с распределенной памятью (*рис.* 5).

В архитектуре A1 разрешен доступ *всех* процессоров к нулевой странице ОЗУ. Межпроцессорная магистраль не используется. Таким образом, имеем модель с общим ОЗУ. Архитектура A2 устроена по-другому: нулевая страница доступна только CPU, а обмен данными идет по межпроцессорной шине.

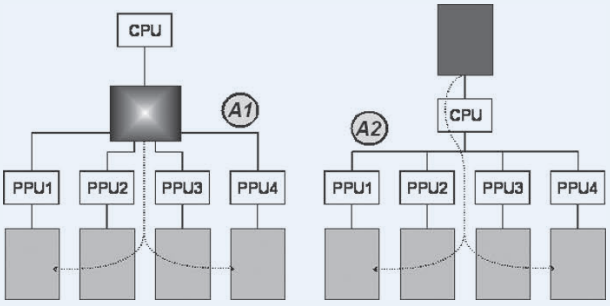


Рис. 5. Архитектуры с общей (A1) и распределенной памятью (A2)

Подчеркнем, что, согласно принятым в “E14” принципам, все действия должны производиться только по инициативе и под руководством центрального процессора. Выглядит это примерно так. По программе из своего ПЗУ центральный процессор читает файл с программой в свою память (в нулевую страницу). Программа разбита на блоки, которые заранее распределены между процессорами, так что CPU остается только разослать их PPU-“адресатам”. Одна из программ остается в нулевой странице — это программа для самого CPU. Она стартует, и процессор, работая по ней, в нужное время запускает (переводит в режим EXEC) нужные PPU на выполнение уже загруженных в них программ. И еще CPU всегда “собирает” в свою память результаты вычислений всех PPU.

Более подробно этот процесс будет описан в следующей части на примере конкретной задачи.

Часть 2. Задача

В качестве тестовой задачи была выбрана **задача суммирования одномерного массива**, или, выражаясь более техническим языком, задача суммирования расположенных в последовательных ячейках памяти целых чисел.

Данная задача по своей сути хорошо распараллеливается: массив чисел можно разделить на части и “раздать” на суммирование имеющимся процессорам, а затем “собрать” и сложить полученные суммы. При небольшом количестве процессоров такой алгоритм является наиболее естественным. Интуитивно он кажется самым быстрым. Возможно, здесь моя интуиция меня подводит. Дело в том, что алгоритм суммирования является предметом подробного математического изучения (см., например, [17]) и теория рассматривает эффективность разных алгоритмов. Но теорию главным образом интересует область $Np \to \infty$, т.е. когда процессоров становится очень много, так что к нашему случаю

[3] Это требование важнее, чем кажется на первый взгляд: не забывайте, что *каждая команда* начинается со считывания ее из ОЗУ, так что в начале каждой операции все пять процессоров “дружно” обращаются к памяти за очередной инструкцией.

она прямого отношения не имеет. В любом случае предложенный нами алгоритм деления массива слагаемых на части имеет право на существование и может служить объектом тестирования.

Удобно принять общее количество суммируемых чисел $N0$ равным 60, поскольку это наименьшее общее кратное для чисел 2, 3, 4 и 5, иначе говоря, 60 чисел всегда делятся поровну между 2-, 3-, 4- и 5-м процессорами. В большинстве наших экспериментов $N0 = 60$, а при проверке бо́льших значений $N0$ всегда выбиралось кратным 60 (120, 180 и 240).

Будем обозначать через N количество чисел, которое просуммирует каждый из процессоров при параллельном алгоритме. Предполагая, что все процессоры проделывают одинаковую работу, получим простейшую формулу

$$N = N0 / Np \qquad (1)$$

Центральный вопрос, который был поставлен в статье, следующий: **если у нас имеется *Np* процессоров, то во сколько раз уменьшится время решения задачи при ее распараллеливании?**

Договоримся еще об одном, возможно несколько спорном, допущении. Эффективность программ будем оценивать **по числу машинных команд, выполненных для получения суммы**. Конечно, команды бывают разные по длительности (например, обращение к ОЗУ выполняется дольше, чем к регистру). Однако трудно сделать параллельную модель, которая бы это аккуратно передавала. Например, попробуйте себе представить, как вы организуете параллельное исполнение двух следующих фрагментов программ (константа τ — это некоторое характерное время[4]):

команда, исполняемая за 2τ	команда, мая за 3τ	исполняе-
команда, исполняемая за 2τ команда, исполняемая за τ	команда, мая за 3τ	исполняе-

Приведем еще два аргумента в пользу выбранного показателя эффективности.

Во-первых, в теоретической информатике число команд действительно может рассматриваться как мера сложности алгоритма. Например, в известной энциклопедии [18] в статье “Теория алгоритмов” прямо сказано следующее: “Физическое время выполнения алгоритма — это величина $\tau\, t$, где t — число действий (элементарных шагов, команд), а τ — среднее время выполнения одного действия. Число шагов t определяется описанием алгоритма в данной алгоритмической модели и не зависит от физической реализации модели; τ — величина физическая и зависит от скорости обработки сигналов в элементах и узлах ЭВМ. Поэтому объективной математической характеристикой трудоемкости алгоритма в данной модели является число действий t”.

Во-вторых, на практике довольно часто в качестве грубой оценки используют непосредственное значение тактовой частоты [15]. Часто для простоты предполагается, что каждая команда выполняется за один такт, и тогда значение тактовой частоты есть просто количество команд, выполняемых за одну секунду. Кстати говоря, экспериментальная проверка этого положения в [8] показала, что оно неплохо согласуется с действительностью, по крайней мере для не слишком “навороченных” процессоров.

Однопроцессорная программа

Программа для вычисления суммы N чисел, расположенных в последовательных ячейках памяти, имеет следующий вид (листинг 1).

Перед нами — классический вариант решения задачи. Смею надеяться, что если вы хоть немного знакомы с ассемблером, то имеющейся в таблице информации вам уже достаточно, чтобы его по-

Листинг 1

Ассемблер		Действия	“Аналог” на Паскале	Комментарии
Метки	Команды			
	mov #N,R0	***N*** ==> R0	R0 := N;	Количество слагаемых
	mov #ARRAY,R1	1A ==> R1	R1 := @ARRAY;	Адрес начала массива
	mov #0,R2	0 ==> R2	R2 := 0;	Обнуление суммы
CYCLE:	add (R1),R2	R2 + (R1) ==> R2	**repeat** R2 := R2 + array[R1^];	Добавление очередного слагаемого
	add #2,R1	R1 + 2 ==> R1	R1 := R1 + 2;	Следующий адрес
	sub #1,R0	R0 – 1 ==> R0	R0 := R0 – 1;	Уменьшение счетчика
	bne CYCLE	переход при ≠ 0	**until** R0 = 0;	Если не 0 — к повторению цикла
	mov R2,SUM	R2 ==> (18)	SUM := R2;	Сохранение суммы в память
	stop	Стоп		Конец — в режим STOP
SUM:				Ячейка для суммы
ARRAY:				Начиная с этой ячейки —
				массив чисел

[4] По смыслу τ — это время выполнения одного машинного такта. Потактовое параллельное моделирование слишком трудоемко, поэтому в “E14” принято покомандное моделирование: выполняется очередная команда CPU, затем PPU1, PPU2, …, PPU4, после чего указанный цикл повторяется.

нять. Многим, возможно, поможет некоторый условный "аналог" программы на Паскале, также представленный в таблице.

Автор понимает, что читатель не обязан быть знаком с ассемблером. Поэтому добавим к таблице некоторые объяснения.

Память распланирована так. Сначала располагается программа (в "E97" и, следовательно, в "E14" тоже программа всегда начинается с нулевого адреса). Затем следует ячейка под сумму (после кодирования она получит адрес 18) и, наконец, суммируемый массив (адрес начала — 1A). То, что массив расположен последним, очень удобно, поскольку его размер никак не повлияет на распределение адресов. Кроме того, поскольку программа "окончательная" и никаким изменениям не подлежит, адреса для SUM и ARRAY тоже не будут изменяться.

Перейдем к программе.

Сначала в регистр R0, который является переменной для обратного отсчета до нуля, заносится значение N — количество чисел. Затем в R1 помещается адрес (указатель) на начало массива (иными словами, на первое число). В дальнейшем при помощи R1 будет извлекаться очередной элемент массива. Подготовку к циклу завершает обнуление регистра R2, где будет накапливаться сумма.

Далее начинается сам цикл. Используя текущее значение адреса в R1, извлекается очередное число и прибавляется к сумме в R2. Путем добавления двойки (каждое целое число занимает два байта!) в R1 формируется адрес следующего числа. Из счетчика оставшихся циклов R0 вычитается единица и, если результат все еще ненулевой, цикл повторяется.

Когда счетчик R0 обращается в ноль, цикл заканчивается. Результат суммирования из R2 копируется в зарезервированную ячейку с именем SUM. Команда stop завершает выполнение программы.

Разобравшись, как работает программа, мы легко можем вычислить количество команд, которое она выполнит для суммирования N чисел. Поскольку каждый цикл состоит из четырех команд, циклу предшествует три, а после него выполняется еще две, то результат описывается несложной формулой

$$K = 4N + 5 \qquad (2)$$

Выражение для K состоит из двух частей: первая соответствует циклу суммирования (ее величина зависит от N, а значит, от числа процессоров); вторая же является постоянной, так что количество складываемых чисел на нее никак не влияет. Если N велико, то добавкой в пять команд можно пренебрегать.

В частности, для нашего базового случая, когда количество чисел $N = 60$, получается $K = 245$. При подстановке $N = 240$ имеем $K = 965$. Запомним эти значения: именно с ними мы будем сравнивать результаты работы параллельных алгоритмов при оценке их эффективности.

Идеально параллельная программа

Теперь попробуем подключить к работе все остальные процессоры. Полагая, что CPU тоже будет участвовать "на равных" в вычислении суммы, получим число считающих процессоров $Np = 5$. Тогда каждый из них согласно формуле (1) при $N0 = 60$ получит для суммирования по $N = 12$ чисел.

Подчеркнем, что программа суммирования, приведенная в листинге 1, годится и для работы в PPU. Это возможно потому, что локальная адресация во всех PPU одинакова.

Для начала рассмотрим максимально благоприятную для параллелизма ситуацию: предположим, что программа и данные для суммирования уже каким-то образом загружены в память каждого из PPU. Будем пока пренебрегать необходимостью собрать и сложить все пять сумм для получения единого ответа. В итоге получаем следующую благостную картину. Каждый PPU согласно формуле (2) при $N = 12$ выполнит по 53 команды. Поскольку CPU, кроме этих 53 команд суммирования, должен будет еще запустить на счет каждый из четырех PPU, то для него количество команд будет равно 57. Очевидно, что при выбранном алгоритме CPU начинает свои вычисления и, следовательно, заканчивает их последним, так что его время совпадает с общим временем работы системы и $K = 57$. По сравнению с однопроцессорной программой при $N0 = 60$ получаем выигрыш $V = 245/57 \approx 4{,}3$, т.е. заметно меньше пяти. И это при данном $N0$ максимально возможный результат!

Вспомним теперь, что в погоне за высокими результатами мы так увлеклись, что не обратили внимания на необходимость собрать воедино результаты суммирования для каждого процессора! А это тоже потребует выполнения некоторой последовательности команд. Очевидно, что эта поправка не будет малой, поскольку обмен данными между процессорами — это очень затратный процесс. Ясно также и то, что результат будет зависеть от выбранного алгоритма и от архитектуры системы. Все эти вопросы будут подробно рассматриваться позднее в части 3. Пока лишь приведем, несколько забегая вперед, результат наилучшего из рассмотренных случаев. Оказывается, если к анализируемому нами идеальному алгоритму добавить пересылку в память CPU результатов суммирования из каждого PPU, а также команды их последующего сложения, то наилучший результат, который удалось получить на "E14", это $K = 70$ (см. табл. 3). Таким образом, реальный выигрыш от параллельности расчетов уменьшается до значения $V = 245/70 = 3{,}5$! Да, наш оптимизм относительно возможностей многоядерности начинает уменьшаться. А мы еще даже не пробовали учесть процесс передачи элементов массива в память к PPU, что необходимо для их независимой параллельной обработки!

Но есть и хорошие новости. Запишем формулу для V в рассматриваемом случае $Np = 5$:

$$V = \frac{4N0+5}{4N0/5+9} = \frac{4N0+5}{4N0+45} \cdot 5 = Cn \cdot 5.$$

Подчеркнем, что первое выражение практически совпадает с одной из форм записи известного закона Амдала [19, 15], с той разницей, что постоянные слагаемые, описывающие "неускоряемый" компонент задачи, в числителе и знаменателе различны[5]. Легко видеть, что при значительном количестве суммируемых чисел $N0$ этими постоянными слагаемыми рядом с $4N0$ можно пренебречь, и тогда $Cn = 1$, а $V = 5$. Таким образом, чем большее значение $N0$ мы возьмем, тем ближе окажется ускорение вычислений для нашего идеального случая к числу процессоров Np. Зависимость $Cn(N0)$ приведена на *рис.* 6.

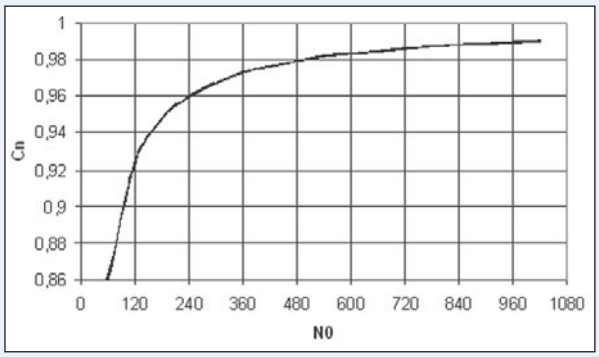


Рис. 6. При большом числе слагаемых $Cn \to 1$ (а $V \to Np$)

Часть 3. Результаты

Опробованные алгоритмы

Как уже рассказывалось в части 1, "E14" позволяет моделировать как архитектуру с общей (*A1*), так и с распределенной (*A2*) памятью. Кроме того, для каждой из архитектур опробовано по несколько параллельных алгоритмов. Введем для них обозначение *S* (схема вычислений) с соответствующим номером. Тогда полное обозначение алгоритма будет иметь, например, следующий вид: *A2S1* — архитектура с распределенной памятью, схема 1. В каждом алгоритме может быть задействовано от одного до четырех PPU; договоримся обозначать этот факт сокращением вроде *P3* — в алгоритме участвуют три PPU. Кроме того, CPU, который всегда "организует" вычисления, тоже может принимать участие в суммировании чисел — в этом случае в конце записи будем ставить знак "+". В частности, запись *P4+* означает, что в вычислениях использовано четыре PPU совместно с CPU. В последнем случае задействованы все имеющиеся в "E14" процессоры.

[5] Напомним, что разница в четыре команды возникла из-за запуска четырех PPU на счет.

Всего было опробовано восемь вычислительных схем. Опишем их подробнее.

Стандартная схема вычислений выглядит следующим очевидным образом. Как вы, вероятно, помните, все загружаемые данные попадают в память CPU. Поэтому первое, что надо сделать, это скопировать каждому PPU его часть данных (числа, которые тот должен просуммировать). Процесс копирования инициируется CPU. Используются стандартные механизмы копирования: для A1 это прямой доступ к памяти (DMA), а для A2 — обмен данными по межпроцессорной магистрали.

Количество чисел для каждого процессора определяется соотношением (1). Так, например, для алгоритмов *P4* (четыре PPU) и *P3* + (три PPU + CPU, всего четыре процессора) каждому процессору "достанется" по $N0/4$ чисел.

Далее CPU запускает каждый из PPU на счет, и те по загруженным в них программам производят сложение чисел. Общая сумма помещается в определенную ячейку. По командам от CPU содержимое всех частичных сумм копируется в ОЗУ CPU и в завершение программы суммируется.

Стандартный алгоритм, как это обычно бывает, оказался самым медленным. Поэтому при обработке результатов и распределении обозначений ему был присвоен самый большой номер. В итоге для каждой из изучаемых архитектур имеем схемы ***A1S2*** и ***A2S2***.

Следующим шагом была сделана попытка увеличить скорость вычислений путем совершенствования алгоритмов. Для обеих архитектур циклы копирования массива были присоединены к общему циклу нахождения суммы. При этом для A1 ради максимального ускорения даже было разрешено читать слагаемые из памяти CPU "по собственной инициативе", не дожидаясь "приказа" от CPU. Для A2 алгоритм также стал нестандартным: центральный процессор по очереди выставлял числа для каждого из PPU, а тот принимал число и немедленно прибавлял к сумме. В обоих случаях удалось добиться некоторого ускорения решения задачи. Полученные вычислительные схемы ***A1S1*** и ***A2S1*** имеют наилучшие из всех опробованных вариантов результаты.

В ходе дальнейшего анализа алгоритма *A1S1* (см. разбор примера ниже) выяснилось, что его можно заметно улучшить. Как оказалось, программа CPU суммирует числа вдвое быстрее, чем это делает каждый из PPU. В результате родился алгоритм *A1S1m*, который распределяет слагаемые между процессорами не поровну по формуле (1), а оставляет на обработку в CPU вдвое больше чисел (например, 20 чисел CPU и по 10 каждому из четырех PPU для алгоритма *P4+*).

Наконец, последняя серия моделирующих экспериментов была проведена следующим (не совсем честным) образом. Команды, копирующие элементы массива, были удалены из алгоритмов, а распре-

деление чисел по PPU выполнялось загрузчиком программ. В итоге после завершения загрузки в ОЗУ PPU попадала не только программа, но и данные для суммирования. Без копирования массива число команд в программе суммирования сокращалось, и результаты улучшались. Тем не менее эти результаты нельзя признать наилучшими, поскольку из задачи механически изъяли одну из ее частей.

Как оказалось, от четырех рассмотренных выше полных схем удалось создать всего три сокращенных: схема ***A2S1*** не допускала "изъятия" команд копирования. Поэтому возникли схемы ***A1S0.1*** (получена из ***A1S1***), ***A1S0.2*** (из ***A1S2***) и ***A2S0*** (из ***A2S2***).

Чтобы не запутаться в многочисленных схемах, все они сведены в табл. 2.

Таблица 2. Обозначения алгоритмов

Алгоритм	Память	Описание
A1S0.1	Общая	Элементы массива копируются в память PPU ***при загрузке***, остальное, как в A1S1
A1S0.2		Элементы массива копируются в память PPU ***при загрузке***, остальное, как в A1S2
A1S1		PPU ***по своей программе*** копирует числа в свою память и при этом суммирует их; потом PPU ***по своей программе*** записывает результат в общую память
A1S1m		Все как в A1S1, но CPU суммирует вдвое больше чисел, чем каждый из PPU. Имеет смысл только для алгоритмов P1+ — P4+
A1S2		Стандартная схема (копирование элементов массива, суммирование, передача результата в CPU); все действия — по командам CPU
A2S0	Раздельная	Элементы массива копируются в память PPU ***при загрузке***, остальное, как в A2S2
A2S1		CPU в цикле передает числа, которые PPU ***немедленно*** суммирует; сумма возвращается в CPU; все действия — по командам CPU
A2S2		Стандартная схема (копирование элементов массива, суммирование, передача результата в CPU); все действия — по командам CPU

Разбор примера

В качестве примера разберем алгоритм *A1S1 P4+* как показавший хорошие результаты и, в то же самое время, самый простой для объяснения.

Программа для PPU1 приведена в листинге 2. Заметим, что для других PPU будет отличаться только адресами в двух командах, строки с которыми выделены цветом.

Листинг 2

Ассемблер		Действия	Комментарии
Метки	Команды		
	mov #3F0,SP	3F0 ==> SP	(При использовании подпрограмм необходимо установить указатель стека SP!)
	mov #N,R3	***N*** ==> R3	Количество слагаемых
	mov #118,R1	118 ==> R1	Адрес начала части массива в ОЗУ CPU (для других PPU адрес другой!)
	mov #0,R2	0 ==> R2	Обнуление суммы
CYCLE:	***call READ***	Вызов подпрограммы ПЗУ	Чтение числа из общей страницы (в R0)
	add R0,R2	R2 + R0 ==> R2	Добавление очередного слагаемого
	add #2,R1	R1 + 2 ==> R1	Следующий адрес
	sub #1,R3	R3 – 1 ==> R3	Уменьшение счетчика
	bne CYCLE	Переход при ≠ 0	Если не 0 — к повторению цикла
	mov R2,R0	R2 ==> R0	Значение суммы
	mov #0FC,R1	00FC ==> R1	Адрес памяти в CPU (для других PPU адрес другой!)
	call WRITE	Вызов подпрограммы ПЗУ	Запись суммы из R0 в общую страницу
	stop	Стоп	Конец — в режим STOP

Разберем, как устроена программа[6].

Программа начинается с занесения значения в указатель стека SP. Это совершенно необходимо, поскольку при вызове подпрограмм READ и WRITE будет пользоваться стековая память, так что важно, чтобы она оказалась в свободном участке ОЗУ.

Далее следует подготовка к циклу суммирования: в регистры заносятся количество чисел, начальный адрес суммируемой части массива и обнуляется R2, в котором будет накапливаться сумма.

Надо обязательно прокомментировать, как получается начальный адрес нужной части массива. Массив для данного алгоритма располагается в общей памяти — на странице CPU (см. *рис.* 4). Его начальный адрес — 100 (здесь и далее все адреса шестнадцатеричные). В рассматриваемом алгоритме *P4+* участву-

[6] Очень полезно сравнивать ее с листингом 1 для однопроцессорной машины.

ют четыре PPU и CPU, т.е. $Np = 5$. Это означает, что массив из 60 чисел делится на пять равных частей по $12_{10} = C_{16}$ чисел. Каждое число занимает по два байта, т.е. 12 чисел требуют $24_{10} = 18_{16}$ байтов ОЗУ. Отсюда немедленно следует, что первая часть массива находится в адресах 100–117 (ее суммирует CPU!), а вторая начинается со 118. Именно этот адрес и ставится в программу PPU1. Продолжив процедуру суммирования (при этом важно не забывать о системе счисления!), легко получить адреса элементов массива и для остальных PPU.

Перейдем, наконец, к рассмотрению цикла. Его первая команда — это вызов подпрограммы ПЗУ, которая, используя R1 в качестве адреса, читает из общей страницы ОЗУ число в R0. Это число сразу же приплюсовывается к общей сумме. Далее в R1 подготавливается адрес следующего числа (он больше на 2) и вычитается 1 из счетчика обратного отсчета. Если значение счетчика ненулевое, то цикл повторяется. Попутно заметим, что всего рассмотренный цикл содержит пять команд, но к ним надо добавить еще четыре команды, образующие подпрограмму READ. Таким образом, всего в цикле оказывается девять команд; позднее мы обсудим, какие выводы о быстродействии алгоритма можно сделать из этого факта.

В завершение результирующая сумма из R2 копируется в R0, а в R1 заносится адрес ячейки в нулевой странице ОЗУ. Стандартная подпрограмма ПЗУ переносит значение из R0 в общую страницу, используя адрес из R1. Заметим, что PPU1 копирует свою сумму в ячейку с адресом FC, PPU2 — в FA и т.д. Общую сумму считает уже CPU.

Программы для PPU2–PPU4 устроены аналогично. Остается рассмотреть программу для CPU, приведенную с некоторыми сокращениями в листинге 3.

Дадим краткие пояснения к программе. Сначала CPU по очереди запускает на счет все PPU, путем записи константы 0 (режим EXEC) в их регистры режима RR. Пока PPU считают, CPU, чтобы не терять времени, тоже суммирует 12 первых чисел массива и результат помещает в ячейку FE. Казалось бы, все процессоры суммируют по 12 чисел, так что время счета будет одинаковым. Но не тут-то было! Дело в том, что CPU берет числа из “своей” памяти, а PPU — из “чужой” (им приходится дополнительно переключать страницы ОЗУ). Поэтому CPU закончит свою работу раньше, и ему придется подождать, пока PPU закончат вычисления. Поскольку все периферийные процессоры одинаковы, то они закончат вычисления практически одновременно (учтите только, что PPU4 запустится на три команды позднее, чем PPU1, следовательно, и закончит он на столько же позднее). Поэтому достаточно дождаться окончания вычислений PPU1.

Ожидание программируется так. CPU считывает через порт 9 содержимое регистра режима PPU1 и сравнивает его с 1 (RR = 1, как вы, наверное, помните, соответствует режиму STOP). Если считано другое значение, то происходит переход на метку WAIT и процесс повторяется.

Как только PPU1 остановится, CPU пройдет дальше по программе и сложит свой результат из ячейки FE с результатами остальных PPU, помещенными в ячейки F6 — FC. Таким несложным способом осуществляется синхронизация вычислений: окончательная сумма считается только тогда, когда периферийные процессоры завершили вычисления, а соответствующие ячейки уже получили свои значения из PPU.

Завершая рассмотрение алгоритма *A1S1 P4+*, оценим его предельное быстродействие по сравнению с однопроцессорным алгоритмом. Восполь-

Листинг 3

Ассемблер		Действия	Комментарии
Метки	**Команды**		
	out #0,p9	0 ==> порт 9	Запуск PPU1 (режим EXEC)
	out #0,pA	0 ==> порт A	Запуск PPU2 (режим EXEC)
	out #0,pB	0 ==> порт B	Запуск PPU3 (режим EXEC)
	out #0,pC	0 ==> порт C	Запуск PPU4 (режим EXEC)
Здесь стояла программа суммирования, отличающаяся от листинга 1 только начальным адресом (100). По ней CPU суммировал “свою” часть массива.			
	mov R2,(FE)	R2 ==> (FE)	Занести сумму в ячейку FE
WAIT:	in p9,R0	Порт 9 ==> R0	Чтение регистра режима
	cmp #1,R0	Сравнить R0 с 1	Когда RR = 1 — режим STOP
	bne WAIT	Переход при ≠ 0	Если не совпало — к повторению
	add (FC),(FE)	(FE) + (FC) ==> (FE)	Прибавить результат, полученный PPU1
	add (FA),(FE)	(FE) + (FA) ==> (FE)	Прибавить результат, полученный PPU2
	add (F8),(FE)	(FE) + (F8) ==> (FE)	Прибавить результат, полученный PPU3
	add (F6),(FE)	(FE) + (F6) ==> (FE)	Прибавить результат, полученный PPU4
	stop	Стоп	Конец — в режим STOP

зуемся той же самой методикой, что была описана в конце части 2. Формула для выигрыша времени V будет такой же самой, только "накладные расходы" параллельного алгоритма нам точно неизвестны (мы не сможем заранее, без запуска имитатора "E14", написать в знаменателе значение константы вместо 9). Тем не менее, как мы видели в части 2, для рассмотрения предела $N0 \to \infty$ знание этого значения оказывается несущественным — оно все равно мало и им можно пренебречь! Так что оказывается, что для большого числа слагаемых $Cn \to 4/9 \approx 0{,}44$ (вспомните, что четыре команды было в однопроцессорном цикле и девять мы имеем сейчас). Таким образом, максимально возможный выигрыш для параллельных вычислений оказывается равным

$$V = 5Cn \approx 5 \cdot 0{,}44 = 2{,}2.$$

Итак, в самом лучшем случае наш пятипроцессорный алгоритм даст выигрыш в быстродействии примерно в два раза! А это, как показывают результаты моделирования на "E14", наилучший результат (при равномерном распределении чисел между процессорами).

Соотношение команд в циклах подсказывает нам, что за одно и то же время CPU успеет просуммировать вдвое больше команд, чем один PPU. А что если попытаться сделать загрузку процессоров более равномерной, распределяя числа не поровну, а дать CPU в два раза больше чисел, чем каждому из PPU? В частности, в модифицированном алгоритме *A1S1m* для рассматриваемого случая $N0 = 60$ и $Np = 5$ числа надо распределить так: двадцать для обработки в CPU и по десять для каждого из четырех PPU. Проверка подтверждает правильность сделанного предположения о росте эффективности вычислений в этом случае. Более того, именно при указанном распределении чисел количество выполненных команд K оказывается минимальным!

Последний результат кажется мне весьма поучительным. Внимательный анализ того, как реально происходят вычисления, позволил довольно легко предсказать возможный путь его оптимизации. Согласитесь, что, не вникая в устройство машинной программы, догадаться о нем было бы трудно.

Результаты

Наконец мы добрались до наиболее интересной части — анализа результатов. Напомню, что цель состояла в том, чтобы оценить ускорение вычислений за счет многоядерного (многопроцессорного) решения задачи. Результаты параллельного моделирования на "E14" для $N0 = 60$ представлены в двух таблицах — 3 и 4. В табл. 3 приведено количество команд для всех опробованных алгоритмов с общей памятью, а в табл. 4 — с распределенной. Выписаны результаты для всех возможных вариантов использования процессоров: от *P1* — единственный периферийный процессор, до *P4+* — четыре PPU плюс CPU. Наилучший (имеющий наименьшее количество команд для каждого метода) вариант в таблицах выделен жирным шрифтом.

Таблица 3. Количество команд K для алгоритмов с общей памятью

A1S0.1				*A1S0.2*				*A1S1*				*A1S1m*		*A1S2*			
P1	258			P1	274			P1	558					P1	770		
P2	140	P1+	139	P2	151	P1+	155	P2	290	P1+	289	P1+	197	P2	406	P1+	411
P3	100	P2+	98	P3	115	P2+	115	P3	199	P2+	200	P2+	153	P3	291	P2+	295
P4	81	P3+	80	P4	100	P3+	98	P4	156	P3+	155	P3+	128	P4	236	P3+	241
		P4+	***70***			P4+	***91***			P4+	***130***	P4+	***111***			P4+	***215***

Таблица 4. Количество команд K для алгоритмов с распределенной памятью

A2S0				*A2S1*				*A2S2*			
P1	254			P1	312			P1	677		
P2	140	P1+	132	P2	258	P1+	286	P2	443	P1+	345
P3	107	P2+	98	P3	244	P2+	262	P3	370	P2+	301
P4	92	P3+	84	P4	***240***	P3+	253	P4	335	P3+	282
		P4+	***78***			P4+	250			P4+	***273***

Еще одна таблица, дополняющая две предыдущих, показывает значения некоторой величины $E = K_{PPU}/K$ для алгоритмов *P4+* (K_{PPU} — это количество команд, которое выполнил каждый из процессоров PPU в данном алгоритме). Очевидно, что значения E не могут превышать единицы, и чем ближе они к единице, тем больше роль эффекта параллельности при вычислениях.

Таблица 5. Количество команд PPU по сравнению с их общим числом K

	A1S0.1	*A1S0.2*	*A1S1*	*A1S1m*	*A1S2*	*A2S0*	*A2S1*	*A2S2*
K	70	91	130	111	215	78	250	273
E	0,86	0,77	0,92	0,92	0,81	0,73	0,22	0,38

Итак, что можно увидеть из приведенных таблиц?

1. Как и ожидалось из общей теории, результаты для архитектуры A1 с общей памятью при прочих равных условиях лучше, чем для A2 — распределенная память.

2. Если не принимать во внимание чисто теоретическую схему S0[7], для архитектуры A2 имитатор не показывает *практически никакой выгоды* от параллельности расчетов: однопроцессорный алгоритм справляется с суммированием за $K = 245$ команд, тогда как лучший результат для *A2S1* — 240. То есть, используя четыре процессора для вычислений и пятый для их организации, нам удалось выиграть всего пять команд! Все четырехкратное ускорение, полученное от суммирования массива на нескольких процессорах, теряется при организации этих вычислений[8]. Напомним, что речь идет о случае $N = 60$, для большего количества слагаемых результат может быть немного лучше.

3. Для реальных алгоритмов *A1S1* и *A1S2* некоторый выигрыш все же возможен: лучший из полученных результатов даст увеличение скорости вычислений в 245/130 ≈ 1,9 раза.

4. Именно анализ кода программ позволил понять, что загрузка при суммировании по алгоритму *A1S1* у CPU и у PPU должна быть не одинаковой, а соотноситься как 2:1. При этом удалось получить выигрыш в 245/111 ≈ 2,2 раза вместо указанного в предыдущем пункте 1,9.

5. При создании многоядерных процессоров, несомненно, предполагается автоматическое распределение работы между ядрами. Оно, как правило, базируется на стандартных методах. В нашем случае стандартные методы — это схема S2, показавшая наихудшие результаты. По-моему, здесь очень уместно привести слова Дональда Кнута, которые он сказал по поводу многоядерных процессоров в одном из своих интервью (цитируется по [20]).

“Мне кажется, у проектировщиков процессоров иссякли идеи, и они хотят переложить ответственность за невозможность повышать производительность компьютеров в соответствии с законом Мура на тех, кто создает программы. Они предлагают процессоры, отлично работающие на отдельных тестах, но я не удивлюсь, что вся эта эпопея многоядерности закончится не меньшим провалом, чем Itanium, где все выглядело прекрасным, пока не выяснилось, что компиляторы с соответствующими ему возможностями предвидения невозможно написать.

Сколько вы знаете программистов, относящихся с энтузиазмом к будущим многоядерным процессорам? У всех, кого я знаю, они лишь вызывают огорчение, хотя разработчики процессоров говорят, что я не прав. Я знаю приложения, адаптируемые к параллельному исполнению; это графический рендеринг, сканирование изображений, моделирование биологических и физических процессов, но все они требуют подходов, которые чрезвычайно специализированны. Моих знаний хватило, чтобы написать о них в “Искусстве программирования”, но я считаю время, затраченное на них, потерянным, в этой области все быстро меняется, и совсем скоро написанное мной никому не будет нужным. Мультиядерность в том виде, как ее представляют, сейчас не прибавляет мне счастья”.

Чтобы не заканчивать на столь минорной ноте, приведем еще один, несколько “утешающий”, график. Он потребовал от меня достаточно кропотливой работы по аккуратному исправлению всех нужных констант, но усилия не пропали даром. На *рис.* 7 вы видите график количества команд *K*, потребовавшихся для нахождения суммы массива из *N*0 элементов с помощью алгоритма *A1S2 P4*, который является стандартным алгоритмом с общей памятью. Отлажены программы и проведены расчеты для значений *N*0, равных 60, 120, 180 и 240. Пунктирная линия на графике — это данные для однопроцессорного алгоритма, сплошная — для четырехпроцессорного.

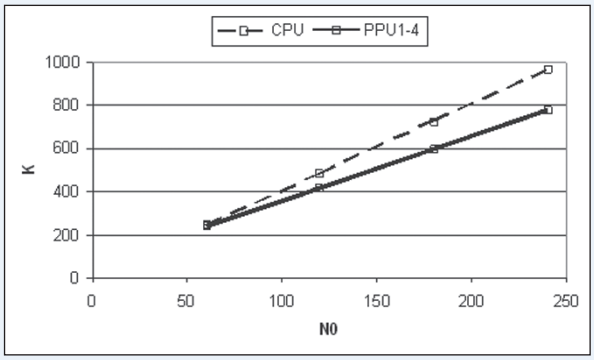


Рис. 7. С ростом числа слагаемых *N*0 эффект от параллельности увеличивается

Из рисунка видно, что разница между графиками при увеличении числа слагаемых в сумме растет. Если при $N0 = 60$ точки практически совпадают (245 и 236), то при $N0 = 240$ разница заметно лучше (965 и 776). Такое повышение эффективности параллельных вычислений с ростом *N*0 хорошо согласуется с обсуждавшейся ранее зависимостью, приведенной на *рис.* 6.

Интересно рассмотреть еще одну, вспомогательную, таблицу результатов (см. табл. 6), где приводится распределение команд между этапами решения задачи для двух стандартных алгоритмов *A1S2* и *A2S2*. Каждый из них разделен на три этапа: *Kc* — число команд, потребовавшееся для копирования массива, *Ks* — число команд, обеспечивших суммирование, и *Kr* — число команд для копирования результатов в CPU и получения там итоговой суммы. Числа в таблице получены путем вставки команды “стоп” сначала после этапа копирования, а затем — после суммиро-

[7] Схема S0 действительно малопригодна для практики: едва ли программист захочет “вручную” разделять большой массив на пять равных частей, как это сделано для S0.

[8] Так и хочется сделать вывод, что плохое администрирование способно свести на нет работу любых высококлассных специалистов!

вания. Зная также общее количество команд, нетрудно высчитать количество команд на каждом этапе.

Таблица 6. Распределение работы между этапами параллельного алгоритма

		Kc	Ks	Kr		Kc	Ks	Kr
A1S2	**P1**	492	249	29				
	P2	257	130	19	**P1+**	252	130	29
	P3	182	90	19	**P2+**	177	90	28
	P4	147	69	20	**P3+**	142	69	30
					P4+	123	58	34
A2S2	**P1**	424	244	9				
	P2	304	125	14	**P1+**	214	125	6
	P3	264	86	20	**P2+**	204	86	11
	P4	244	67	24	**P3+**	199	67	16
					P4+	196	56	21

Из сравнения *Kc* и *Ks* отчетливо видно, что копирование требует заметно больше времени, чем последующее суммирование уже перенесенных в память PPU чисел[9]. Иными словами, высказанный ранее вывод о том, что эффект от параллельного суммирования в значительной степени “съедается” предварительным копированием, убедительно подтверждается количественно.

Число команд получения общей суммы, приведенное в столбце *Kr*, по величине меньше, чем *Ks*, но для большого числа процессоров становится соизмеримым. Так что и этот этап вносит свой вклад в общее замедление алгоритма.

Завершая наше неформальное обсуждение многопроцессорных вычислений, отметим еще один аспект, связанный с рассмотренной тематикой. Как подчеркивал в своем выступлении на недавней конференции один из пионеров школьной информатики А.Г. Гейн [21], в качестве двух новых разделов в школьный курс должны войти основы программирования робототехники и параллельное программирование. Что касается азов робототехники, то мы сегодня видим активную деятельность по распространению Лего-наборов, позволяющих собирать всевозможные управляемые микроконтроллерами объекты. Теперь неплохо бы подумать и о методической основе для изучения параллельных вычислений.

Приложение

В приложении читатель найдет собственно программу-имитатор “E14”, все программы суммирования массивов для нее и некоторые файлы с документацией. Замечу, что в папке listings можно посмотреть разбор некоторых дополнительных машинных программ, не включенных в текст статьи.

Наличие готовых программ позволит проверить любую из приведенных в статье цифр, а также попутно посмотреть, как работает “E14”. Чтобы написать свою собственную параллельную программу, придется дополнительно почитать о принципах кодирования команд процессора “E97” в книгах [8–10] или в материалах сайта [6].

[9] Скорее всего именно поэтому выбранная задача дает столь пессимистичные результаты!

Литература

1. *Лин В.* PDP-11 и VAX-11. Архитектура ЭВМ и программирование на языке ассемблера. М.: Радио и связь, 1989.
2. Основы информатики и вычислительной техники: Пробное учебное пособие для средних учебных заведений. В 2 ч.: Ч. 2 / *А.П. Ершов, В.М. Монахов, А.А. Кузнецов и др.* Под ред. А.П. Ершова и В.М. Монахова. М.: Просвещение, 1986.
3. *Кушниренко А.Г. и др.* Основы информатики и вычислительной техники: Пробное учебное пособие для средних учебных заведений / *А.Г. Кушниренко, Г.В. Лебедев, Р.А. Сворень.* М.: Просвещение, 1990.
4. *Нортон П., Соухэ Д.* Язык ассемблер для IBM PC. М.: Издательство “Компьютер”; Финансы и статистика, 1992.
5. Основы информатики и вычислительной техники. *А.Г. Гейн, В.Г. Житомирский, Е.В. Линецкий, М.В. Сапир, В.Ф. Шолохович.* Свердловск: Изд-во Урал. ун-та, 1989.
6. Учебные модели компьютера. *http://educomp.runnet.ru*.
7. *Столлингс В.* Структурная организация и архитектура компьютерных систем. М.: Издательский дом “Вильямс”, 2002.
8. *Еремин Е.А.* Популярные лекции об устройстве компьютера. СПб.: BHV-Петербург, 2003.
9. *Еремин Е.А.* Как работает современный компьютер. Пермь: изд-во ПРИПИТ, 1997.
10. *Могилев А.В., Пак Н.И., Хеннер Е.К.* Информатика: учебное пособие для студентов педагогических вузов. М.: Академия, 2012.
11. *Кнут Д.Э.* Искусство программирования, том 1, выпуск 1. MMIX – RISC-компьютер нового тысячелетия. М.: ООО “И.Д. Вильямс”, 2007.
12. *Шагурин И.И., Бердышев Е.М.* Процессоры семейства Intel P6. Архитектура, программирование, интерфейс. М.: Горячая линия – Телеком, 2000.
13. *Семенов А.А.* Многоядерная архитектура универсальной 32-разрядной учебной машины. *http://www.rae.ru/forum2011/153/1794*.
14. Cell Broadband Engine Programming Handbook Including the PowerXCell 8i Processor. Version 1.12. IBM, 2009.
15. *Немнюгин С.А., Стесик О.Л.* Параллельное программирование для многопроцессорных вычислительных систем. СПб.: БХВ-Петербург, 2002.
16. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: BHV-Петербург, 2002.
17. *Гергель В.П.* Теория и практика параллельных вычислений. М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2007.
18. Информатика. Энциклопедический словарь для начинающих. / Сост. *Д.А. Поспелов.* М.: Педагогика-Пресс, 1994.
19. *Amdahl G.M.* Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. Proceedings of AFIPS Spring Joint Computer Conference, 1967.
20. *Черняк Л.* Архитектура фон Неймана, реконфигурируемые компьютерные системы и антимашина. Открытые системы, №6, 2008. *http://www.osp.ru/os/2008/06/5340894*.
21. *Гейн А.Г.* Ожидания информатики. / Информатика в школе: прошлое, настоящее и будущее. Всероссийская научно-методическая конференция по вопросам применения ИКТ в образовании. Пермь: Перм. гос. нац. исслед. ун-т, 2014.

журнал

Информатика – Первое сентября

2-е полугодие 2015 года

ПОДПИСКА

на сайте www.1september.ru и в почтовых отделениях РФ

Индекс	Название издания	Периодичность в полугодие	1 месяц		6 месяцев	
			Каталожная цена (руб.)	Подписная цена (руб.)	Каталожная цена (руб.)	Подписная цена (руб.)
Название блока в разделе «Журналы»	**ПЕРВОЕ СЕНТЯБРЯ. ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА** (499)249-31-38					
79066	**Информатика – Первое сентября. Бумажная версия** С дополнительными материалами и презентациями на сайте www.1september.ru *В июле не выходит.* *Подписка на июль не принимается* *(-) 160 г 64 стр.*	5	440.00		2200.00	
12684	**Информатика – Первое сентября. Электронная версия на CD (полная копия бумажной версии)** С дополнительными материалами и презентациями *В июле не выходит.* *Подписка на июль не принимается* *(-) 75 г*	5	160.00		800.00	
сайт 1september.ru	**Информатика – Первое сентября. Электронная версия**	5	–		–	500.00

Подписку принимают во всех отделениях связи Российской Федерации, а также на сайте www.1september.ru

При оформлении подписки на сайте оплата производится по квитанции в отделении банка или электронными платежами on-line

VISA MasterCard QIWI WebMoney

Решение задачи о “счастливых билетах” с помощью параллельных вычислений

П.В. Карабаза

Из курса информатики известно, что процессор, как бы сложно он не был устроен, выполняет алгоритм решения задачи ***по шагам***. Сколько бы окон у вас не было открыто, в данный момент любой процессор выполняет ***одну*** конкретную команду одной конкретной задачи. Кстати, делает он это за вполне определенное время.

А если взять несколько процессоров и заставить их работать одновременно? Когда несколько команд выполняются параллельно, общее время решения задачи должно уменьшиться. Процессоры при этом занимаются *параллельным вычислением* и являются *вычислительной системой*.

Насколько же быстрее такая система может решать задачи? Теоретически N исполнителей позволяют ускорить решение в N раз. Но так ли это на практике?

Отмечу также, что параллельность вычислений имеет смысл только для ***хорошо “распараллеливаемых”*** задач. То есть задач, в которых есть не зависящие друг от друга блоки. Например, сложение большого количества чисел и т.п. Одну из таких задач мы и рассмотрим.

Постановка задачи

Рассмотрим известную задачу подсчета количества “счастливых билетов”. Конечно, с использованием современных программных средств решить ее можно моментально, написав всего 13 строк. Но эта задача хорошо подходит для рассмотрения принципов и возможных преимуществ параллельного вычисления:

• нет входных параметров, что позволяет немедленно запустить все параллельные программы на счет;

• есть возможность разделения на независимые блоки (распараллеливание), так как количество “счастливых билетов” в каждой тысяче не зависит друг от друга;

• результаты работы каждого процессора останется просто сложить, что является простым и быстрым взаимодействием;

• проверка большого количества “билетов” позволяет увидеть значительный эффект параллельного решения;

• правильность полученных результатов можно легко проверить с помощью современных программных средств.

Алгоритм

Что такое “счастливый билет”? По сути, это набор из шести цифр, в котором сумма первых трех равна сумме трех последних. В общественном транспорте я часто замечаю, как тот или иной пассажир проверяет свой билет на “счастливость”. Наверное, интуитивно мы предполагаем, что вероятность получить такой билет совпадает с вероятностью быть удачливым в этот день.

Интересно, какова эта вероятность? Естественно, ее можно подсчитать, разделив число “счастливых” билетов на их общее количество. И если первое является результатом решения нашей задачи, то второе влияет и на алгоритм, и на скорость ее решения! Хорошо, что это число уже известно. На всем знакомой билетной ленте нумерация начинается с 000–000-го и заканчивается 999–999-м номером. Следовательно, общее количество билетов — 1 000 000 штук. И все их нужно проверить.

Самым простым способом проверки является перебор всех чисел, нахождение для каждого соответствующих сумм цифр и их сравнение. Его алгоритм в виде блок-схемы (*рис.* 1) и на языке Pascal представлен ниже:

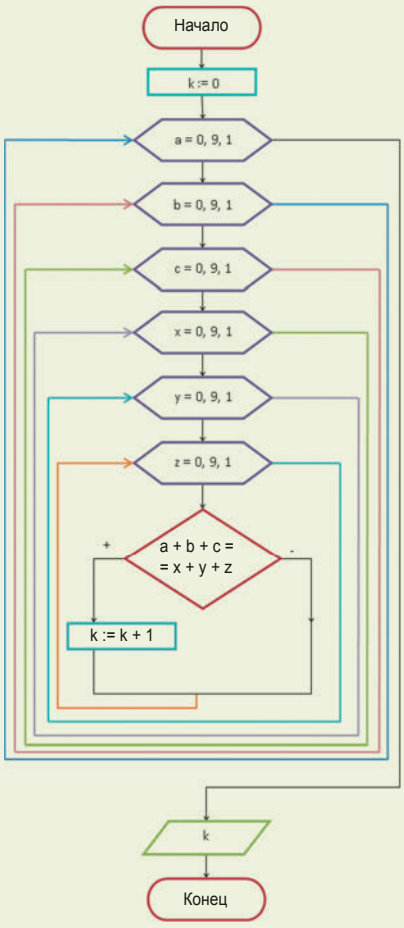


Рис. 1

```
program BILET;
var a,b,c,x,y,z,k: integer;
BEGIN
 k := 0;
 for a := 0 to 9 do
  for b := 0 to 9 do
   for c := 0 to 9 do
    for x := 0 to 9 do
     for y := 0 to 9 do
      for z := 0 to 9 do
       if a + b + c = x + y + z then
          k := k + 1;
 writeln(k);
END.
```

Этот простой алгоритм требует очень много вычислительных ресурсов. Излишне, к примеру, каждый раз считать сумму цифр, так как она изменяется по не совсем тривиальному, но все же цикличному закону:

• при увеличении младшей из трех цифр на единицу их сумма также увеличивается на единицу (пример: 688 + 1 = 689, сумма цифр: 22 → 23);

• при увеличении следующей цифры на единицу, когда разряд единиц обнуляется, сумма цифр уменьшается на 8 (пример: 689 + 1 = 690, сумма цифр: 23 → 15);

• наконец, при увеличении старшего разряда на единицу все младшие обнуляются, а сумма уменьшается на 17 (пример: 699 + 1 = 700, сумма цифр: 24 → 7).

Далее рассматривать не обязательно, так как нам нужно вычислять сумму только трех цифр.

Казалось бы, не самая простая зависимость, но обратите внимание на следующее:

• –8 = –9 + 1;

• –17 = –9 – 9 + 1.

Благодаря вышесказанному алгоритм можно оптимизировать, храня суммы для сравнения в дополнительных переменных. Вот пример в виде блок-схемы (*рис.* 2 на с. 36) и на языке Pascal:

```
program BILET_I;
var a,b,c,x,y,z,k,m,n: integer;
//в m хранится сумма первых трех цифр,
BEGIN  //в n – последних
 k := 0; m := 0; n := 0;
 for a := 0 to 9 do begin
  for b := 0 to 9 do begin
   for c := 0 to 9 do begin
    for x := 0 to 9 do begin
     for y := 0 to 9 do begin
      for z := 0 to 9 do begin
       if m = n then k := k + 1;
       n := n + 1;       //(*)
      end;
      n := n – 9;
  //вместе с (*) получается n := n - 8 (**)
    end;
```

```
      n := n – 9;
//вместе с (**) получается n := n - 17
    end;
    n := 0; m := m + 1;
//последние три цифры становятся нулями,
   end;     //а m изменяется аналогично
   m := m – 9;
  end;
  m := m – 9;
 end;
 writeln(k);
END.
```

Программа получилась в два раза длиннее, но она работает в 1,5 раза быстрее благодаря тому, что не суммирует четыре раза для каждого числа. Вместо этих четырех операций для каждой комбинации цифр обычно выполняется всего одна операция, на каждом десятом — две, на каждом сотом — три и т.д. Для одного миллиона чисел это дает заметное ускорение.

Но стоит ли этим заниматься, когда у нас есть несколько процессоров, которые и так в разы ускорят решение? Во-первых: никогда нелишне оптимизировать программу. А во-вторых: теперь у нас есть два алгоритма, пусть и для одной задачи. Оба алгоритма хорошо распараллеливаются, а значит, они оба послужат для подтверждения или опровержения гипотезы об эффективности *параллельного вычисления* (*N* исполнителей ускорят решение в *N* раз).

Распараллеливание

Итак, у нас есть алгоритм нахождения количества “счастливых билетов” для одного процессора. Теперь нужно переписать его для нескольких. Естественно, заставлять два процессора считать одно и то же не имеет смысла. Каждый из них должен выполнять свою часть задачи. Несколько процессоров, например, могут находить количество “счастливых билетов” на разных непересекающихся промежутках. Эти промежутки могут следовать друг за другом, или можно проверять билеты через один, два и т.д. Первое, конечно, проще организовать. Общее количество билетов разделяется на равные части, например, по одной тысяче на каждый процессор, а полученные результаты суммируются. Чем больше процессоров, тем меньше количество билетов, которое проверит каждый из них. А поскольку они работают одновременно, время, затраченное на решение задачи, уменьшается. Напомню, что в нашей задаче количество “счастливых билетов” в одном промежутке не влияет на количество таковых в другом. Это и делает ее хорошо распараллеливаемой.

Пусть, к примеру, у нас есть 10 равнозначных процессоров, собранных в вычислительную систему. Тогда нуждающийся в проверке 1 млн. билетов можно разделить на 10 частей, по 100 000 билетов в каждой. В таком случае одному процессору из десяти нет необходимости перебирать все значения первой цифры. Она остается неизменной в пределах одной сотни тысяч и может быть принята как константа, необходимая для вычисления суммы первых трех цифр. Именно в этой константе, в первой цифре и будет отличие алгоритмов для разных процессоров одной вычислительной системы. Процессоры подсчитают количество “счастливых билетов” каждый в своей сотне тысяч и сложат полученные результаты. Теоретически задача будет решена в 10 раз быстрее.

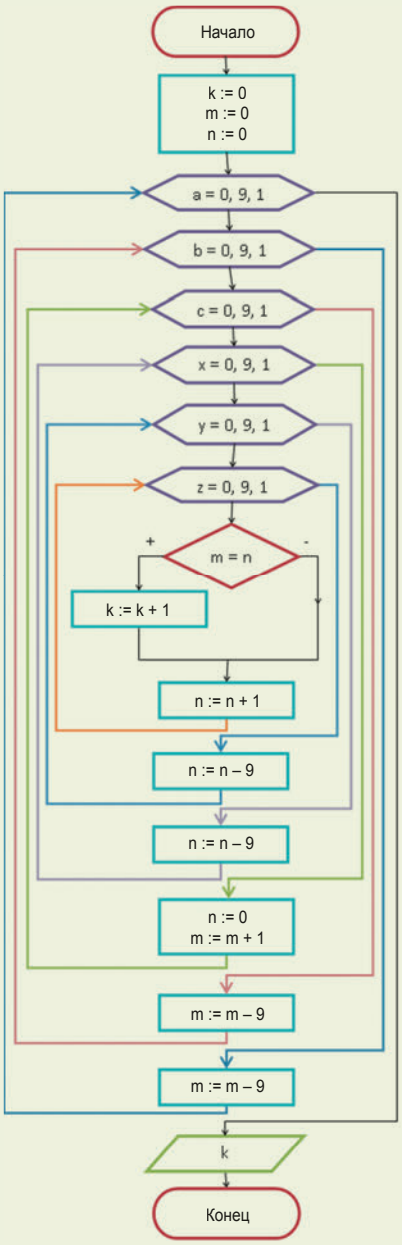


Рис. 2

Инструмент

Алгоритм решения есть. Теперь его нужно реализовать. Для этого необходима вычислительная система, состоящая из более чем одного вычислителя. В современных персональных компьютерах фактически несколько ядер, но организовать их параллельную работу самостоятельно и заставить выполнять только нашу задачу довольно сложно. Суперкомпьютер, содержащий множество процессоров, мне также никто не предоставит.

Инструментом реализации алгоритма может стать учебная модель многопроцессорного компьютера “E14” (программа имеется в электронном приложении к журналу). Она доступна и наглядна. В этой модели пять процессоров, и есть возможность самостоятельно распределить работу между ними, написав соответствующую программу. Удобно, что “E14” считает количество выполненных команд для каждого процессора, что можно принять за количественный критерий “быстроты” решения задачи. Так как команды выполняются за примерно одинаковое время,

общее время решения задачи напрямую зависит от их количества. Для оценки эффекта от параллельности вычислений можно найти количество “счастливых билетов”, используя сначала один процессор, а потом все процессоры. Затем сравнить количество команд, выполненных только одним процессором, с количеством команд, выполненных одним из пяти, так как они работают одновременно. Результат сравнения и будет показателем эффективности параллельного вычисления. Из вышесказанного следует, что “E14” подходит для реализации распараллеленного алгоритма решения задачи.

Что необходимо знать о “E14” для решения нашей задачи?

• В “E14” пять процессоров (CPU, PPU1, PPU2, PPU3, PPU4), каждый из которых может выполнять вычисления.

• CPU должен “запустить” остальные процессоры, а значит, его алгоритм будет несколько длиннее остальных.

• У каждого процессора четыре равноправных регистра (R0, R1, R2, R3).

• Оперативная память каждого процессора составляет $3FF_{16}$ однобайтовых ячеек, что позволяет сохранить до $200_{16} = 512_{10}$ целых 16-разрядных чисел.

• Процессоры PPU имеют механизм передачи результатов своей работы в память CPU, который и просуммирует их, что еще немного удлинит его программу.

К сожалению, разрядности регистров процессоров недостаточно, чтобы вместить искомое количество “счастливых билетов”. Разрядность “E14” 16 бит, первый из которых отвечает за знак числа. Следовательно, максимальное число, которое может храниться в регистре, не превышает 32 767. Количество “счастливых билетов” несколько больше. Выходом может стать нахождение его не для всей билетной ленты, а для ее половины. Гистограмма, представленная ниже (*рис.* 3), создана по результатам приведенного ранее алгоритма, выполненного в PascalABC. Она показывает количество билетов для каждой первой цифры (пример: среди билетов, у которых первая цифра 2, “счастливых” 5631).

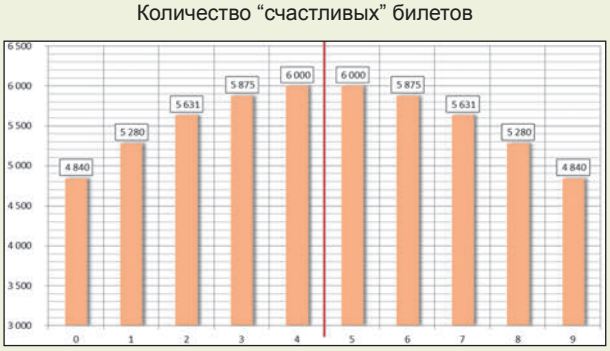


Рис. 3

Как видно на гистограмме, количество “счастливых билетов” распределено симметрично относительно середины, а значит, в первой половине их столько же, сколько во второй. И это 27 626, что укладывается в рамки, обусловленные разрядностью регистров “E14” (27 626 < 32 767).

Итак, используя учебную модель многопроцессорного компьютера “E14”, подсчитаем количество “счастливых билетов” среди первых пятисот тысяч билетов (от 000–000-го до 499–999-го), распределив их между пятью процессорами, по 100 000 на каждый. Также отдельно будет выполнен подсчет только на одном центральном процессоре (CPU). Затем сравним количество выполненных команд в первом случае с аналогичным количеством во втором случае, получив тем самым представление об эффективности параллельного вычисления.

Реализация

Приступим к реализации составленного алгоритма с помощью выбранного инструмента. “E14” работает по заранее написанной программе, сохраненной в текстовый файл. Чтобы не вдаваться в подробности

Простой алгоритм	Оптимизированный алгоритм

`mov 0, R1`
`mov 0, [90]`
`m1: mov 0, [92]`
`m2: mov 0, [94]`
`m3: mov 0, [96]`
`m4: mov 0, [98]`
`m5: mov 0, [9A]`
`m6: mov [90], R2`
`add [92], R2`
`add [94], R2`
`mov [96], R3`
`add [98], R3`
`add [9A], R3`
`cmp R3, R2`
`jne k0`
`add 1, R1`
`k0: add 1, [9A]`
`cmp 0A, [9A]`
`jne m6`
`add 1, [98]`
`cmp 0A, [98]`
`jne m5`
`add 1, [96]`
`cmp 0A, [96]`
`jne m4`
`add 1, [94]`
`cmp 0A, [94]`
`jne m3`
`add 1, [92]`
`cmp 0A, [92]`
`jne m2`
`add 1, [90]`
`cmp 0A, [90]`
`jne m1`
`stop`
	`mov 0, R1`
	`mov 0, R2`
	`mov 0, R3`
	`mov 0, [90]`
	`m1: mov 0, [92]`
	`m2: mov 0, [94]`
	`m3: mov 0, [96]`
	`m4: mov 0, [98]`
	`m5: mov 0, [9A]`
	`m6: cmp R3, R2`
	`jne k0`
	`add 1, R1`
	`k0: add 1, R3`
	`add 1, [9A]`
	`cmp 0A, [9A]`
	`jne m6`
	`sub 9, R3`
	`add 1, [98]`
	`cmp 0A, [98]`
	`jne m5`
	`sub 9, R3`
	`add 1, [96]`
	`cmp 0A, [96]`
	`jne m4`
	`mov 0, R3`
	`add 1, R2`
	`add 1, [94]`
	`cmp 0A, [94]`
	`jne m3`
	`sub 9, R2`
	`add 1, [92]`
	`cmp 0A, [92]`
	`jne m2`
	`sub 9, R2`
	`add 1, [90]`
	`cmp 0A, [90]`
	`jne m1`
	`stop`

языка команд “E14” при обсуждении, на с. 37 приведен алгоритм решения задачи нахождения количества “счастливых билетов” на языке ассемблера:

- все числа написаны в шестнадцатеричной системе счисления;
- в R1 хранится искомое количество;
- в ячейках 90 – 9A — цифры проверяемого билета;
- в R2 — сумма первых трех цифр;
- в R3 — сумма последних трех.

Итак, файл с программой написан по всем правилам и готов к применению. Запускаем модель многопроцессорного компьютера “E14”. Ее интерфейс представлен на *рис.* 4.

В центре находится панель центрального процессора (CPU). За ней четыре панели PPU, по одной на каждый. На каждой панели четыре регистра (R0, R1, R2, R3), а также регистр состояния, счетчики команд и стека. Напомню, искомое количество “счастливых билетов” хранится в регистре R1 для любого из процессоров.

Сверху по центру находится пульт “E14” (*рис.* 5), на котором цветовыми индикаторами отображается состояние каждого процессора и всей системы в целом:

- выполнение программы (EXEC);
- ожидание (STOP);
- работа по программе ввода данных из CPU (INPUT);
- работа по программе вывода данных в CPU (OUTPUT).

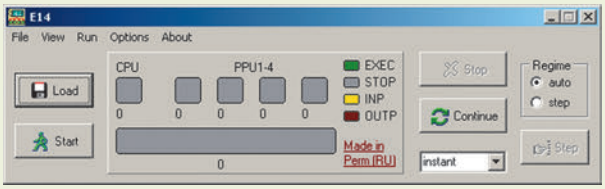


Рис. 5

Также под соответствующим индикатором в реальном времени выводится количество команд, выполненных каждым процессором. Максимальное из них, которое и является показателем времени решения задачи при параллельном вычислении, выводится под общим (нижним) индикатором системы.

С помощью пульта осуществляется управление всей моделью многопроцессорного компьютера. Можно увеличить или уменьшить скорость выполнения программы, что никак не повлияет на количество выполненных команд.

Есть возможность остановить выполнение программы (*Stop*) и выполнять ее по одной команде (*Step*). Для запуска программы используется кнопка *Start*, но перед этим ее необходимо загрузить в процессоры, нажав *Load* и выбрав

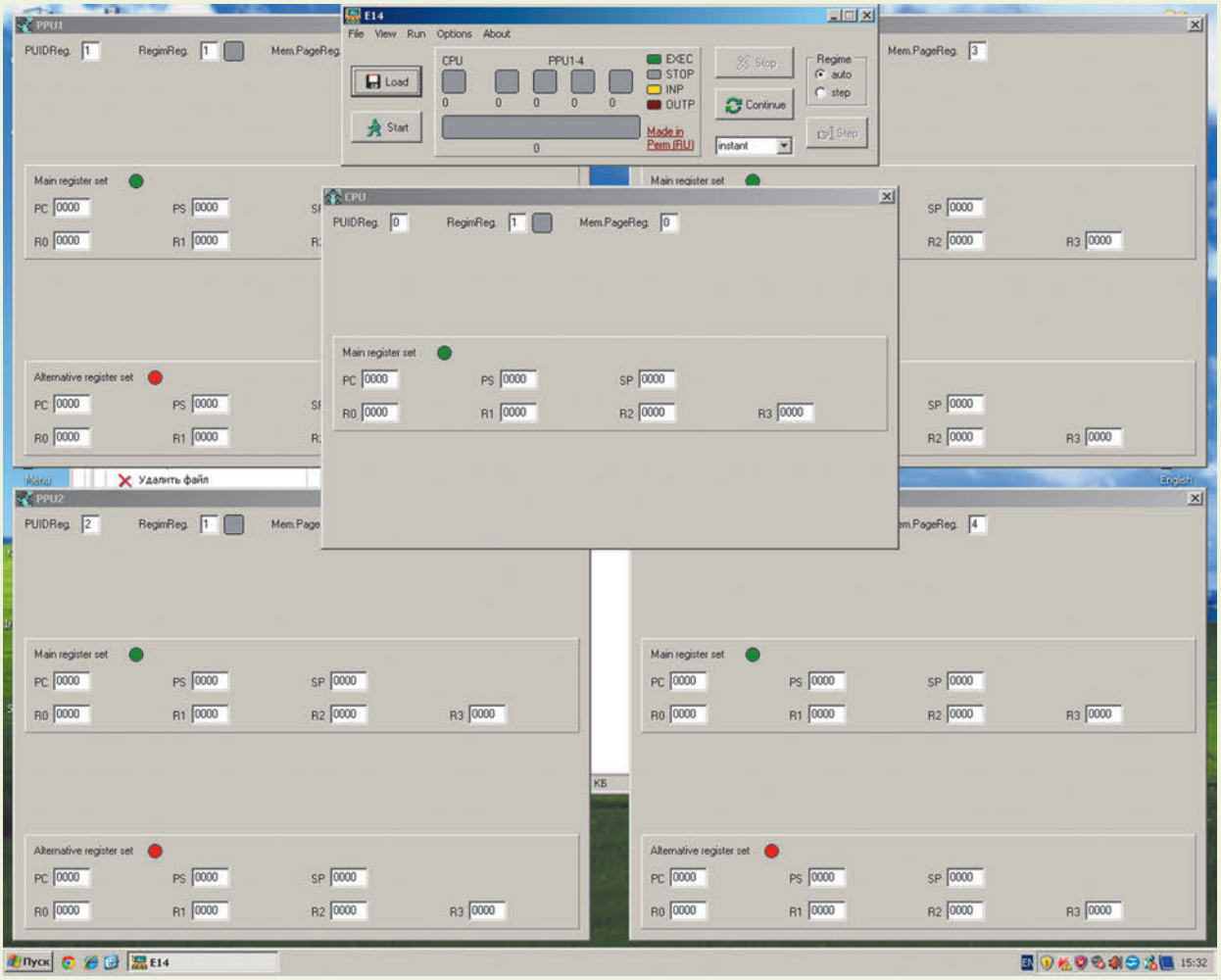


Рис. 4

соответствующий, ранее подготовленный файл. Отображение процесса загрузки на пульте представлено на *рис.* 6.

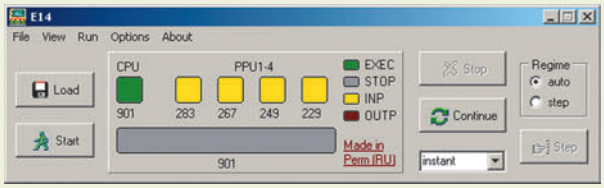


Рис. 6

Можно посмотреть состояние памяти, выбрав *Memory* на вкладке *View* (*рис.* 7).

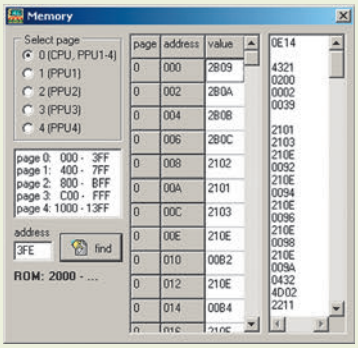


Рис. 7

Далее, после пуска программы, можно наблюдать, как CPU по очереди запускает остальные процессоры (*рис.* 8). В данном примере выполнение начали CPU, PPU1 и PPU2, а PPU3 и PPU4 еще ожидают команды центрального процессора. Нижний индикатор показывает, что система уже начала работу.

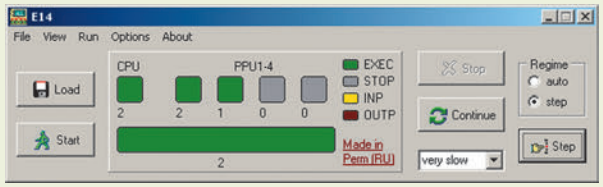


Рис. 8

Когда все процессоры запущены и решают задачу, суммарное число выполненных команд в разы больше даже максимального из слагаемых. Например, на *рис.* 9 пять процессоров совместными усилиями выполнили 39 959 команд за время, которое один тратит на 7993 команды. Эффективность параллельного вычисления налицо.

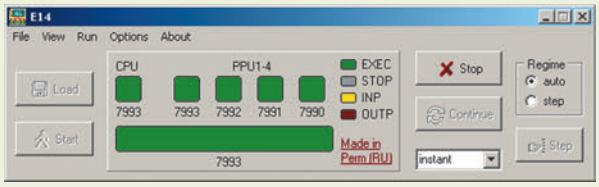


Рис. 9

Осталось дождаться, когда “E14” закончит работу, и записать, сколько команд он при этом выполнил. Это нужно проделать для четырех программ: двух вариантов алгоритма (простого и оптимизированного) и двух вариантов распределения работы (один процессор и пять), а затем сравнить полученные результаты. Конечно, искомое количество “счастливых билетов” будет одинаковым, а вот время, затраченное на его вычисление, различным. И также различным будет показатель этого времени — число выполненных команд.

Результаты

Итак, “E14” честно выполнил четыре программы, ошибок в которых не было. “Счастливых билетов” на половине билетной ленты 27 626 штук, и на всей, соответственно, 55 252. Результаты вычислений совпали, а распределение количеств осуществленных при этом команд представлено в таблице на *рис.* 10.

Количество выполненных команд

Алгоритм	Количество процессоров		Эффективность параллельности
	Один	Пять	
Стандартный	5 749 849	1 150 464	4,9979
Оптимизированный	3 305 906	661 675	4,9963

Рис. 10

Как видно, эффект параллельного вычисления при равномерном распределении задачи чуть-чуть меньше ожидаемого. Пять одновременно работающих процессоров не ускорили решение задачи ровно в пять раз. Причина кроется в необходимости, во-первых, организовать работу и, во-вторых, сложить результаты в одном месте. Всем этим занимается CPU, так как он является центральным, управляющим. Поскольку только один процессор может осуществлять организацию работы, то и распараллелить решение организационных вопросов невозможно.

А распределять работу и собирать результаты ее выполнения необходимо в любом случае. Как бы хорошо ни распараллеливалась задача, ее распараллеливание еще нужно правильно организовать. То есть задача организации параллельного вычисления является плохо распараллеливаемой. И при этом она всегда будет сопутствовать решению любой вычислительной задачи.

Вывод

К сожалению, гипотеза о том, что *N* исполнителей ускорят решение в *N* раз, не подтвердилась в точности. Эффективность, хоть и незначительно, но все же меньше теоретической даже на специально подобранной задаче. А в ней нет вводимых данных, и обработка результатов простейшая. То есть затраты на организацию параллельности минимальны. А в большинстве случаев организация параллельных вычислений может быть сложнее. Тогда эффективность снизится еще больше.

Тем не менее решение задачи на “E14” наглядно показывает, что параллельные вычисления могут давать существенный эффект. При этом, правда, очень велика роль анализа алгоритма при подготовке его к распараллеливанию. Так что надеяться на то, что компьютер сам сумеет оптимально распределить работу между своими многочисленными процессорами или ядрами, пока еще рано, а программисты, умеющие это делать квалифицированно, определенно будут востребованны.

CAM
Мы можем делать вещи

И.А. Калинин,
зам. нач. управления информационных систем и технологий в образовании ФГБОУ Гос. ИРЯ им. А.С. Пушкина

Подавляющее большинство людей, если спросить: “Какое занятие возникает перед глазами при слове «компьютер»?” — расскажет вам что-нибудь о вычислениях, написании текста, картинках на том или ином экране, общении в сети, чем и ограничится. Кто-то вспомнит про написание программ. Почти всегда речь пойдет о чем-то по сути не материальном. Неудивительным образом, компьютер противопоставляют реальному миру.

Но любой, кто сталкивался с современной технологией производства, знает, насколько это не так. В этой статье (и, возможно, нескольких следующих) нам хотелось бы поговорить именно о компьютере, который делает вещи. Самые настоящие.

Общие сведения

Производственные системы (и просто отдельные станки) под общим компьютерным управлением традиционно называют CAM — ***C**omputer-**A**ided **M**anufacturing*, переводится это как “управляемое компьютером изготовление”. Системы эти применяют (а сейчас и практически всегда объединяют) с системами компьютерного проектирования — CAD.

Сердцем такой системы чаще всего является станок с числовым программным управлением (CNC — ***C**omputer **N**umeric **C**ontrol*).

Несмотря на то что станки такие очень разнообразны, управление ими строится на нескольких общих понятиях:

1. Рабочий орган, инструмент — активная часть станка: резец, фреза, лазер и т.д. Орган, воздействующий на материал.

2. Ось (измерение) — координата, задающая одно из положений инструмента. Это может быть положение по длине, по высоте, угол поворота заго-

товки или самого инструмента. Чем больше осей, тем больше возможностей у станка — и тем сложнее обеспечить его работу.

3. Программа для станка — последовательность перемещений по всем осям (траектория) и изменения режимов работы инструмента.

Программа для станка фактически представляет собой траекторию передвижения инструмента (с какой-то скоростью), на которой инструмент либо работает, либо нет. Существует несколько языков подготовки таких программ; самый известный стандартный язык, поддерживаемый подавляющим большинством управляющих комплексов, — G-code.

Все, что делает станок, будет зависеть от его возможностей и программы.

Станок с ЧПУ состоит из нескольких блоков:

1. Собственно станок — механическая конструкция для крепления материала и инструмента, обеспечивающая их точное перемещение. Основными параметрами станка будут параметры жесткости (насколько станок точно и твердо обеспечивает работу — не прогибается ли, например, под весом, не смещается ли от вибрации и т.д.) и точности (можно ли сдвинуть инструмент на 0,1 мм или только на 1 мм? Не проскальзывает ли, например, во время перемещения стол с заготовкой?).

2. Рабочий орган, инструмент. Может быть, это будет неподвижный резец, может быть — фреза, может быть — лазер или еще что-то.

3. Шаговые двигатели, или сервомоторы, — двигатели, обеспечивающие точное перемещение инструмента.

4. Роутер — собственно устройство управления, подающее команды на двигатели и рабочий орган. Роутер обеспечивает перевод команд от управляющего компьютера в команды инструменту или перемещение по осям. Чаще всего это делается отработкой команд вида “Шаг-Направление” (“Step-Dir”).

Роутер на самом деле только управляет перемещениями, а остальное будет зависеть от компьютера — встроенного или отдельно стоящего (со специальной программой, например — Mach3).

Довольно долго такие станки (и вообще слово “станок”) связывались с каким-то большим, дорогим и специальным производственным оборудованием, большим цехом и т.д. Но современные станки стали гораздо меньше и часто используются и в условиях небольшой мастерской, офиса и даже дома. Такие станки стали появляться и в школах и колледжах.

Одна из программ, в рамках которой школьники могут получить доступ к такой технике, — центры технологической поддержки образования, создаваемые правительством Москвы на базе высших учебных заведений (подробнее: *http://ctpo-portal.stankin.ru/about/*).

Поговорим о нескольких таких станках.

Лазерная резка

Суть процесса предельно проста: специальная газовая трубка генерирует лазерное излучение в инфракрасном (тепловом) диапазоне. Это излучение фокусируется на материале, нагревает его и он разрушается (плавится) в точке нагрева (физика процесса намного сложнее — но сейчас для нас это не важно). Поскольку точка мала (как правило — около 0,01 мм), а материал разрушается быстро, то, перемещая такой луч, мы будем фактически разрезать лучом нашу заготовку на части.

Что может резать такой станок? В зависимости от мощности трубки и управляющих программ:

- пластики (акрил, ПВХ, полистирол…);
- дерево (доска, шпон, фанера);
- ткань;
- бумагу;
- стекло;
- металл (вплоть до тугоплавких сталей).

Кроме резки, такой станок может не прорезать материал, а оставлять на нем след — выполнять гравировку. Насколько глубоко — зависит от режима работы.

Задавая параметры работы станка, мы должны учитывать, что по мере использования трубка постепенно слабеет, причем нельзя в обычных условиях определить, насколько. При этом материал, с которым мы работаем, часто имеет изменяющиеся параметры (например, фанеру могут проклеить разным клеем). Поэтому все параметры приводятся приблизительно и часто требуют пробной резки или гравировки перед тем, как выполнять весь заказ.

Приведем несколько примеров, заодно покажем, как работает программа управления таким станком.

Вот пример такого станка, на котором мы проверяли все написанное в этой статье, — см. фото.

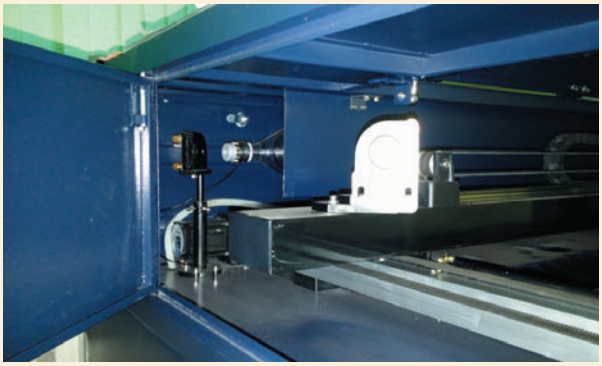
Это станок StepDir 10060 — станок лазерной резки и гравировки, с мощностью трубки 100 Вт, размером рабочего стола 100 × 60 см. Он управляется контроллером RDC6332G — фактически встроенным компьютером.

Конечно, для резки стекла или металла, в зависимости от типа лазера, мощность луча должна быть не менее 200–300 Вт (для CO2 лазера — более 1000 Вт) и иметь систему автоматической фокусировки луча. Такие станки дороги. В условиях обычных помещений чаще используют станки с мощностью трубки до 100 (иногда — 180) Вт.

Вот как выглядит трубка (основной элемент) нашего станка:

Обратите внимание — во время работы трубка должна быть закрыта, знак опасности на ней проставлен не просто так. К трубке подведена дистиллированная вода — для охлаждения. Вода подается обычной помпой все время работы — иначе трубка перегреется и выйдет из строя (а это довольно дорогая часть станка).

Трубка генерирует инфракрасный лазерный луч, который через систему зеркал передается на фокусирующую линзу и попадает на материал.



А вот это и есть рабочая часть. Заодно тут видно, как мы разместили материал для раскроя:

Шаговые двигатели перемещают этот рабочий орган по двум осям. Место реза обязательно охлаждается — иначе очень часто материал начнет гореть или испортится место обрезки. На самом деле от способа обдува многое зависит, — например, если нужно препятствовать горению, то обдувают не воздухом, а углекислым газом или азотом. Очень рекомендуется к таким станкам подключать вытяжку — иначе все очень быстро оказывается в дыму.

Подготовка рисунка

Рисунок, который будет представлять собой траекторию обработки, очевидно, должен быть векторным. Как мы увидим ниже, растровый рисунок тоже можно использовать — но для других целей.

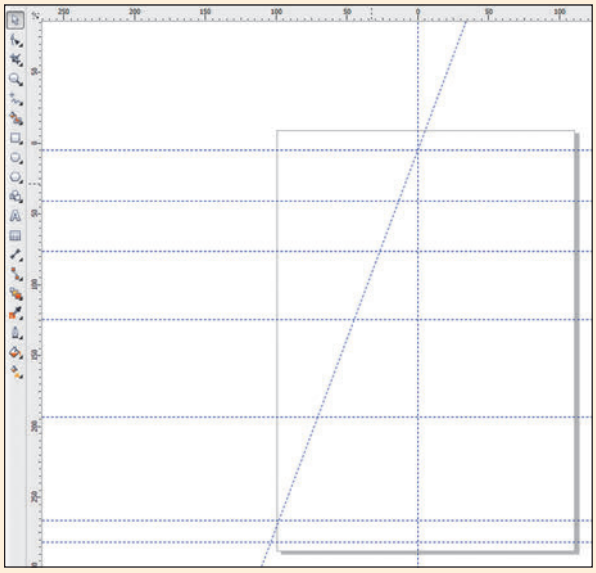
Наш проект-рисунок для примера — сувенирная новогодняя елка.

Рисунок можно готовить в самых разных программах: общие принципы будут практически одинаковыми. Елка симметрична, так что начнем с ее половины.

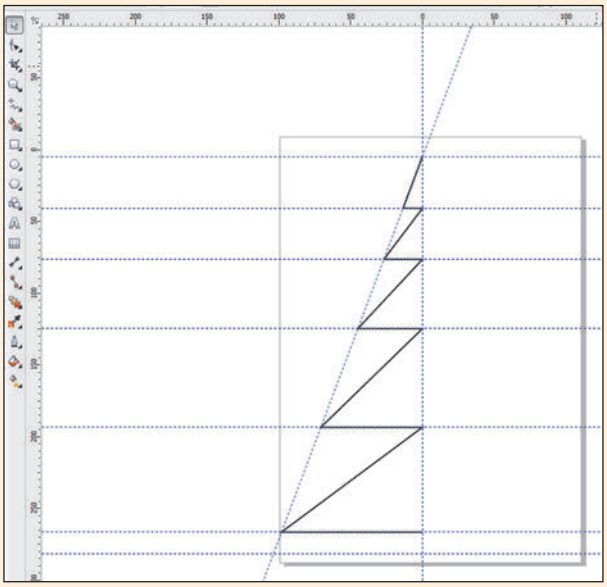
Мы не можем в рамках этой статьи рассматривать использование программ векторной графики, так что рассмотрим только основные этапы и приемы построения.

Как и во всех программах векторной графики, основа этого рисунка — кривая, состоящая из нескольких частей.

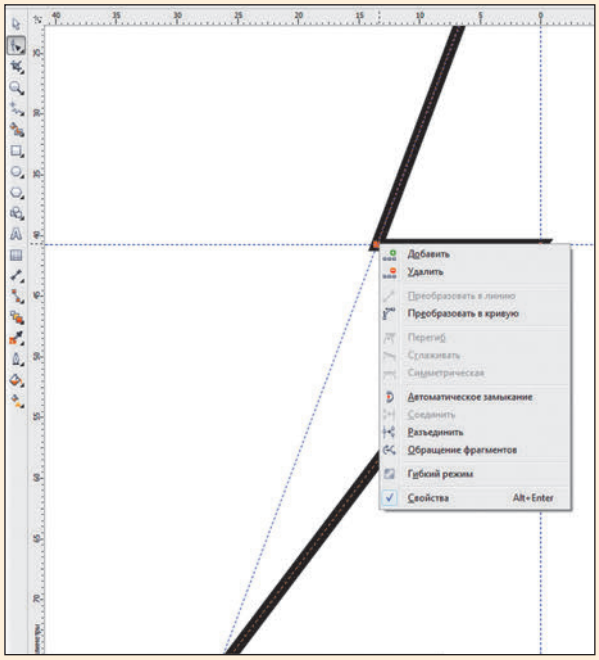
Чтобы елка была ровной, начнем с построения вспомогательных линий:



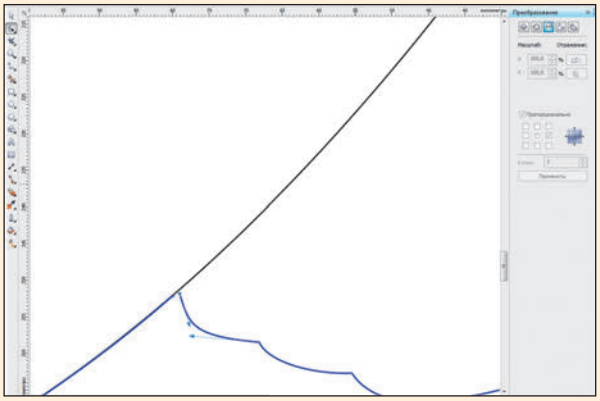
Теперь построим ломаную из отрезков:



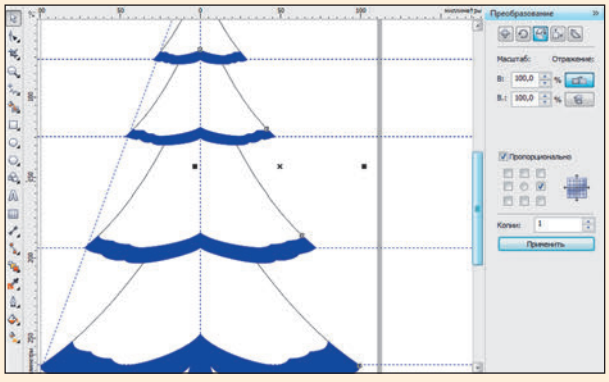
После этого сменим инструмент на управление узлами и преобразуем сегменты ломаной в кривые Безье:



После этого, манипулируя направляющими векторами (на рисунке — линии со стрелками), добьемся нужной формы ветвей:



Построив половину елки, воспользуемся зеркальным отображением относительно ее правого края и получим полную елку:



Обратите внимание: мы указали, что отображение создаст копию половины елки.

Последняя операция, которую мы выполним, — предусмотрим выемки для сборки. Толщина выемки зависит от толщины материала. В нашем случае — 4 мм. Необходимо предусмотреть, что лазерный луч проплавит примерно 0,2 мм, так что размер выемки нужно сделать чуть меньше.

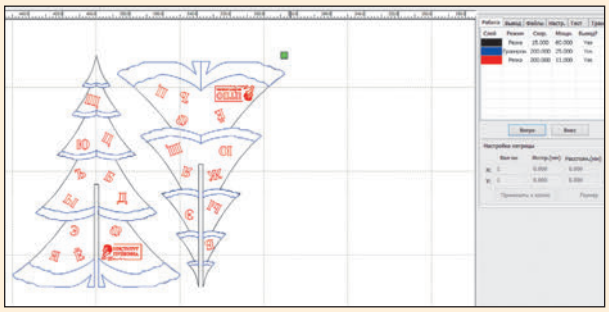
На основной елке можем добавить элементы оформления, в нашем случае — буквы русского алфавита.

Подготовка программы обработки

Программу для обработки мы формируем и отправляем контроллеру станка с помощью программы LaserWorks RDCAM 6.0.24.

В качестве первого примера мы готовили сувенирную елку. Режимов во время ее вырезания мы будем использовать два: гравировка (для снега) и резка (чтобы выпилить ее из полосы материала). Резку мы используем и для того, чтобы написать на елке буквы.

Для работы мы будем использовать программу, пришедшую в комплекте со станком: LaserWorks (RDCAM).



Программа имеет полный набор инструментов для создания кривых — такой же, какой мы использовали ранее, — но управление ими не так удобно.

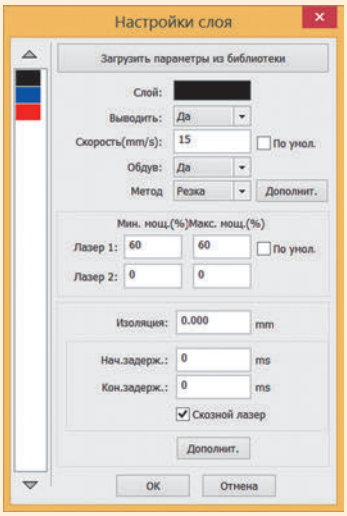
Обратите внимание на зеленую точку в правом верхнем углу. Именно с этой точки начинается отсчет координат, и именно она считается начальным положением лазера.

Зададим режимы работы лазера. Для этого мы должны нарисовать линии разным цветом — именно разные цвета и будут использоваться для определения режима.

Наш материал — акрил (оргстекло), как мы уже говорили — толщиной 4 мм. Обычно для гравировки хватает 30–40 ватт, и гравировать можно быстро (200 миллиметров в минуту). Для резки скорость будет ниже, а мощность — выше (иначе срез сразу заплавляется снова или вообще не прорезается до конца). Очень может быть, что если задаться целью, можно подобрать более оптимальный режим.

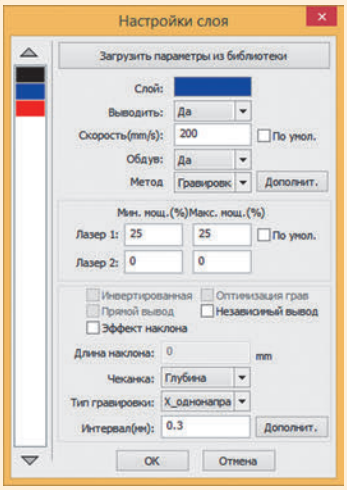
Если мощность будет мала — лист не будет прорезан, если велика — материал может загореться или слишком сильно оплавиться (особенно там, где много мелких деталей).

Контур елки отнесем к черному слою. Вот параметры этого слоя:

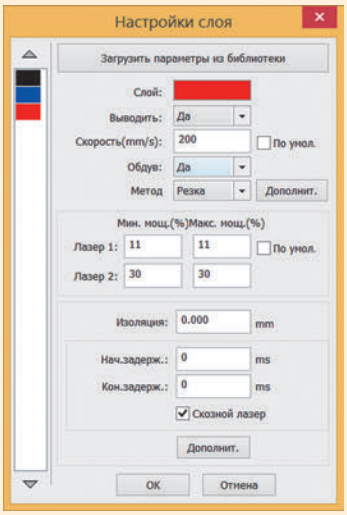


Минимальное значение используется там, где скорость перемещения снижается, — во время смены направления, на поворотах и т.д. От нас не требуется соблюдения жестких требований к качеству среза, поэтому мы не указываем минимальное значение, и вообще значения подбираем из опыта.

“Снег” мы сделаем в режиме гравировки (синий слой):



Буквы будут нарисованы в режиме резки (красный слой). Тут нам как раз нужно, чтобы буквы не прорезались, а именно состояли из линий. Поэтому и скорость больше, а мощность ставим невысокую.



Напомним, что начальное положение обработки задано зеленой точкой. Но это точка на рисунке, а как она соотносится с листом материала? Никак, — мы должны выставить начальное положение лазера — причем по всем трем осям.

Поэтому сначала подготовимся:

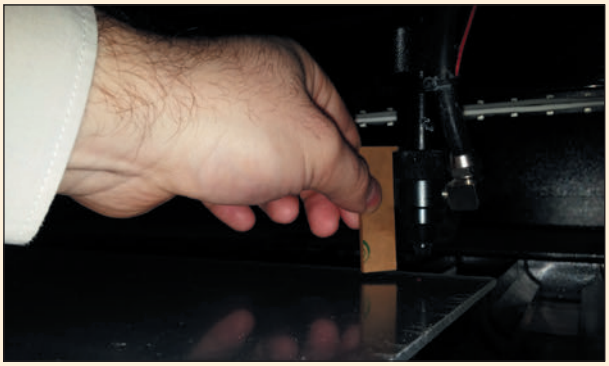
1. Кладем лист на рабочее поле.

2. Подгоняем головку на начальную точку — командами с пульта управления станком.

Вот фотография пульта:

На этом пульте мы можем перемещать головку командами X+ и X– по длинной стороне и Y+, Y– — по короткой стороне. Можно задать и скорость перемещения — в миллиметрах в минуту (можно быстро — но не точно, можно точно — но медленно). Команды можно подавать и из программы LaserWorks.

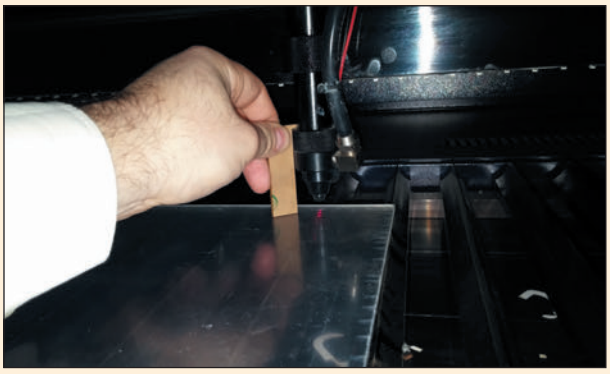
3. Выставляем верное расстояние до материала — от этого зависит фокусировка луча.



Фокусное расстояние нашей линзы — 54 миллиметра. Чтобы его выставить, мы прикладываем заранее вырезанный шаблон. Видно, что головка стоит слишком высоко.

Менять положение головки по вертикали мы не можем, зато можем опустить или поднять сам рабочий стол. Подгоняем расстояние кнопками Z+ и Z–.

Во время работы это расстояние меняться не должно и в программу не вносится — иначе луч потеряет фокусировку. В промышленных устройствах (существенно более дорогих и мощных) используют системы автофокусировки. Это позволяет, например, резать по сложным поверхностям.



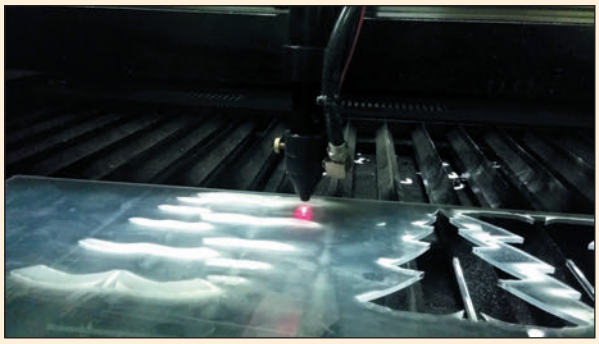
Вот теперь все верно.

4. Теперь прогоняем головку по периметру рамки вокруг всех вырезаемых деталей — чтобы убедиться, что все поместится на листе материала. Луч вполне может выходить за рамки детали.

Сам лазер, как мы уже говорили, инфракрасный — и его не видно. Поэтому для позиционирования у нас есть простейший маркер — лазерная указка, по которому мы и контролируем его положение.

5. Закрываем крышку (ПРИ ОТКРЫТОЙ КРЫШКЕ ЛАЗЕР В РАБОЧИЙ РЕЖИМ НЕ ПЕРЕЙДЕТ).

6. Проводим резку:

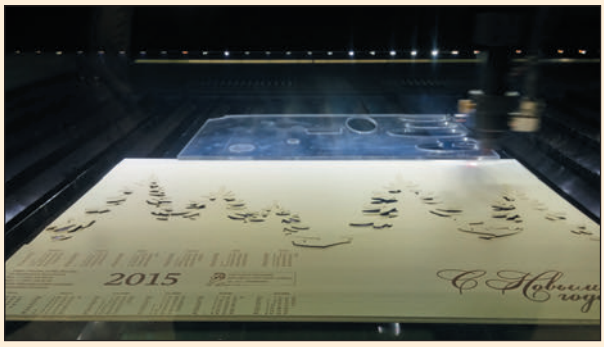


Как видно, это не первая елка на этом листе оргстекла.

Вот что у нас получается:

Полагаем, что техника сборки елки вполне очевидна из рисунка.

Еще один пример, приведем без подробностей. Одна из популярных задач таких станков при подготовке рекламных и представительских материалов — резка бумаги и картона. Точное позиционирование позволяет вырезать в плотной бумаге все, что только можно нарисовать:



На этой фотографии вырезается сложный контур новогодней открытки. Плотность бумаги — 280 г/см^3, скорость лазера — до 200 мм/мин., мощность — 22 ватта.

Еще один пример работы станка — гравировка. В режиме гравировки лазер не проходит по контуру, а “штрихует” его. В нашей программе возможны четыре варианта: штриховка по оси *X* в одном направлении, штриховка по оси *X* в двух направлениях, и такие же два варианта штриховки по оси *Y*.

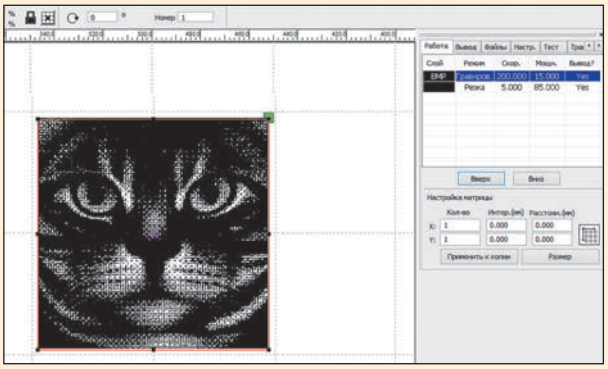
С помощью гравировки мы уже делали “снег” на ветках сувенирной елки, но в таком режиме можно гравировать и растровое изображение.

При этом у нас нет возможности рисовать разными цветами и изображение должно быть адаптировано к оттенкам серого цвета. Имеющийся у нас станок не отрабатывает изменений мощности “на лету”, поэтому изображение мы адаптируем к оттенкам серого, используя “матрицы точек” (точно так же формируется полутоновое изображение на монохромном лазерном принтере). При этом мы понизим разрешение, но размер выжигаемой точки все равно не может быть слишком мелким — материал выжигается все равно в каком-то круге.

Возьмем изображение кота (найдено в Google, никаких особенностей не имеет).

В программе обработки растровой графики переведем его в оттенки серого цвета, указав, что изображение будет монохромным. После адаптации получится вот что:

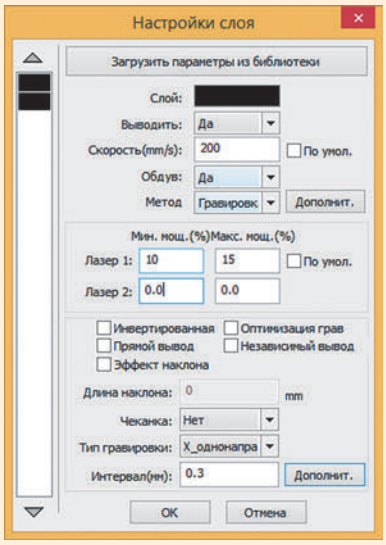
И результат импортируем в LaserWorks RDCAM:



При импорте LaserWork автоматически переводит изображения из цветного в оттенки серого цвета, но качество адаптации у него получается хуже. Именно поэтому мы обработали изображение заранее.

Вокруг изображения нарисуем прямоугольник — чтобы его вырезать из листа. У слоя, обозначенного “BMP”, есть только один вариант обработки: гравировка. При этом есть несколько параметров, для обычной линейной гравировки недоступных.

Наш материал — 8-миллиметровая хвойная фанера (не очень удачный вариант — но другой не было). Исходя из этого, зададим параметры гравировки:



Важно выставить правильный интервал между линиями. Несмотря на то что по умолчанию для растровой гравировки выставляется 0,1 мм, этого для нашей ситуации мало — мы объединяли точки, а поэтому, чтобы сохранить нужный оттенок, выбираем 0,3 мм.

Скорость и мощность лазера подобраны экспериментально.

Для обрезки мы установили скорость 5 мм, а мощность — 80 %.

Вот фотография результата:

Мы рассказываем далеко не обо всех возможностях таких станков. Например, мы ничего не рассказываем о гравировке стекла, об использовании поворотной оси и многих других возможностях.

Даже показанные в этой небольшой статье способы позволяют сделать многое. С помощью такого станка можно кроить ткань и кожу, гравировать сложные рисунки на дереве, стекле, пластике и металле, делать печати и сложные мозаичные рисунки.

Возможность быстрого и точного изготовления деталей предоставляет массу возможностей для организации и проведения самых разных проектов — от конкурса елочных игрушек до создания сложных моделей самолетов и упражнений в робототехнике.

Автор благодарит Центр технологической поддержки образования Государственного института русского языка им. А.С. Пушкина за предоставленную возможность воспользоваться станком.

1 апреля открывается прием заявок на 2015/16 учебный год

Фестиваль педагогических идей «Открытый урок» festival.1september.ru

Свидетельство о регистрации СМИ Эл. № ФС77-53231

В течение 12 лет – самый массовый, представительный и посещаемый педагогический форум Рунета. Самая большая коллекция авторских разработок учителей.

Разместить публикацию может каждый педагог. Всем авторам предоставляются документы о публикации. По итогам каждого учебного года выпускаются электронные и бумажные сборники.

В рамках фестиваля для желающих проводится конкурс презентаций. Всем участникам конкурса высылаются специальные дипломы.

Удобный Личный кабинет участника фестиваля, возможность автоматического создания личного профессионального портфолио. В помощь участникам – квалифицированные сотрудники оргкомитета. Единственный в России образовательный сайт, имеющий службу поддержки в режиме on-line 7 дней в неделю.

Участвуйте в фестивале, размещайте свои работы, получайте документы о публикации!

Фестиваль творческих и исследовательских работ учащихся «Портфолио ученика» project.1september.ru

Свидетельство о регистрации СМИ Эл. № ФС77-53211

Площадка для публикации работ учащихся, выполненных под руководством педагогов.

Всем ученикам и педагогам предоставляются документы о публикации. По итогам каждого учебного года выпускаются электронные и бумажные сборники.

В рамках фестиваля для желающих проводится **конкурс проектных работ**.

Все участники конкурса награждаются специальными дипломами.

Участвуйте вместе с учениками!

Опасен ли умный робот?

Обыватель боится роботов и искусственного интеллекта (ИИ). Можно долго перечислять произведения научной фантастики, в которых механические слуги были созданы и прочно заняли важную нишу в жизни, но по какой-то причине вышли из-под контроля и обернулись против своих создателей.

В последнее время наводить первобытный страх начали крупные и влиятельные фигуры в мире науки и информационных технологий. Сразу два очень известных человека высказались за ограничение работ в области искусственного интеллекта, так как они считают эти работы очень опасными. Этими людьми оказались известный астрофизик Стивен Хокинг и не менее известный предприниматель, организатор компании Space X, Илон Маск. Илон Маск заявил, что ИИ “потенциально опасней ядерного оружия”, а Стивен Хокинг назвал недооценку угрозы со стороны ИИ “величайшей ошибкой человечества”. По мнению Хокинга, роботы просто перехитрят человека, станут умнее своего создателя.

Основанием для опасений этих двух очень уважаемых людей является чрезвычайно высокая скорость работ в области компьютерной техники. Еще в 1965 году американский ученый и предприниматель Мур высказал предположение, что число транзисторов на кристалле будет удваиваться каждые два года, а Давид Хаус из фирмы Intel спрогнозировал, что производительность процессоров должна удваиваться каждые 18 месяцев из-за сочетания роста количества транзисторов и быстродействия каждого из них. За прошедшие годы быстродействие компьютеров выросло в тысячи раз, и закон Мура продолжает работать.

Современные роботы по сравнению с людьми, конечно, весьма примитивны и по функциональным возможностям тела, и по возможностям управляющего компьютера, который размещается в “теле”. Однако за ближайшие 20 лет должны смениться не менее четырех поколений конструкций роботов, а быстродействие компьютеров при тех же габаритах должно вырасти не менее чем на три порядка. И робот с более совершенным телом и с более быстрым “мозгом” уже будет казаться гораздо менее примитивным. Специалисты говорят, что в будущем может случиться так, что машины с нечеловеческим интеллектом будут самосовершенствоваться, и ничто не сможет остановить данный процесс. А это, в свою очередь, запустит процесс так называемой “технологической сингулярности”, под которой подразумевается чрезвычайно быстрое технологическое развитие. В статье Хокинга и других ученых [1] отмечается, что такая технология превзойдет человека и начнет управлять финансовыми рынками, научными исследованиями,

людьми и разработкой оружия, недоступного нашему пониманию. Если краткосрочный эффект искусственного интеллекта зависит от того, кто им управляет, то долгосрочный эффект — от того, можно ли будет им управлять вообще.

Важным является ответ на вопрос — может ли на базе компьютера быть создан интеллект, способный выходить за рамки заложенного в него алгоритма? Или искусственный интеллект принципиально алгоритмичен и работает строго по заданным правилам? А если внеалгоритмический ИИ может быть создан, то какой ИИ более опасен — алгоритмический или внеалгоритмический?

Опасения Хокинга и Маски разделяют не все специалисты. На перспективы ИИ имеется и "оптимистическая" точка зрения. Интересная позиция высказана в [2]. Опишем ее.

Принципы работы искусственного интеллекта и человеческого мышления фундаментально отличаются. Работа ИИ полностью определяется заложенными в него программами (алгоритмами). Человеческий же разум способен выходить за рамки стандартного поведения и может решать так называемые "невычислимые задачи". Так называют задачи, которые не сводятся к алгоритмам[1].

На сегодня единственным существом, способным решать невычислимые задачи, то есть обладающим интуицией и способностью к творчеству, является человек. Перечисленными способностями мы обладаем благодаря нашей нейронной сети (сети нервных клеток) и методам обучения детей в сообществе взрослых. Это экспериментальный факт, и пока общепринятой теории сознания и мышления нет, в процессе моделирования можно опираться только на факты, которые известны о факторах, влияющих на возникновение сознания у людей.

С точки зрения авторов книги [2], искусственный интеллект, способный решать невычислимые задачи, может быть создан. Он будет работать (уместно ли здесь слово "жить" вместо "работать"?) на основе искусственной нейронной сети — аппаратного и программного воплощения принципов организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.

На сегодня общепризнано, что в нейронной сети может зародиться сознание, если выполнены некоторые условия.

1. Нейронная сеть должна быть достаточно сложной.

2. Нейронная сеть должна быть соединена с телом, через которое обеспечивается взаимодействие с окружающим миром.

3. Нейронная сеть должна находиться в обучающей среде. На сегодня единственной известной обучающей средой, в которой может вырасти разумное вещество, является человеческое сообщество.

4. Необходимо наличие генетически заданных мотиваций. Нейронная сеть может обеспечить ум, но для действий необходимы воля и цель.

5. Мышление происходит в открытом режиме, то есть в процессе мышления в мозг продолжает поступать огромное количество информации от органов чувств, что создает постоянный информационный шумовой фон, поддерживающий необходимый уровень возбуждения в нейронной сети.

Какие направления исследований могут привести к возникновению мыслящего робота с собственным сознанием?

Независимо от исследований в области роботостроения нейробиологи и программисты работают над созданием виртуального человеческого мозга. Современные суперкомпьютеры позволяют смоделировать нейронную сеть, соответствующую по размеру мозгу крысы. Через двадцать лет должны быть созданы компьютеры, позволяющие смоделировать мозг человека.

Очевидно, что по мере продвижения работ по упомянутым выше двум направлениям (совершенствование тела роботов и разработка более быстрого "мозга" для них) возникнет третья группа исследователей, которая объединит современное тело робота и почти человеческую виртуальную нейронную сеть, в результате может возникнуть не просто робот, а нечто совершенно новое. Назовем его условно — "умный робот". На сегодня возможность возникновения такого робота пока представляется фантастикой, однако некоторые этапы его моделирования уже сегодня очевидны.

Любое моделирование всегда предполагает определенную степень упрощения. В качестве такого упрощения в книге [2] рассматривается схема строения мозга МакЛина.

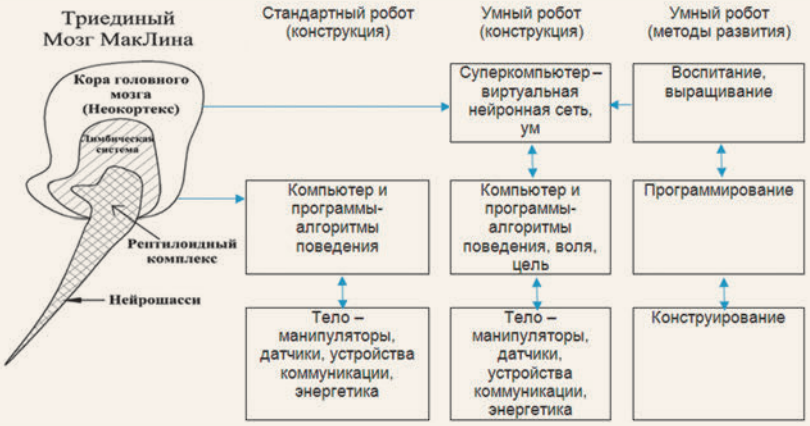
В середине двадцатого века американский физиолог МакЛин, изучая человеческий мозг, пришел к выводу, что его условно можно разделить на три отдела. Рептилоидный комплекс мы унаследовали от древних рептилий. Это тот отдел, где расположены центры управления нашими инстинктами или центры генетической памяти. Лимбическую систему мы унаследовали от их древних млекопитающих (центр эмоций). И, наконец, развитие неокортекса (кора головного мозга) привело к возникновению людей.

Схема МакЛина удобна еще и тем, что в ней прослеживается связь структуры мозга и основных составных частей структуры нашего разума, а именно: воля (цель, мотив); эмоция; интеллект (вычислитель). Здесь эмоция, очевидно, выступает как передаточный механизм между целью и интеллектом, побуждая его к выбору оптимального варианта достижения цели.

С высокой степенью вероятности можно утверждать, что генетически предопределенные формы поведения людей связаны с генетически предопределенной частью нервной системы. И, с другой стороны, формы поведения, обусловленные обучением, связаны с развивающейся в процессе обучения корой головного мозга.

Таким образом, за генетическую память, инстинкты, мотивацию поступков отвечают древние отделы мозга, формирующиеся в соответствии с заданной генетической программой. Высшие формы поведения форми-

[1] Следует различать невычислимые задачи и так называемые "алгоритмически неразрешимые задачи", то есть задачи, которые невозможно решить. Примеры последних приведены, например, в [4].



руются в процессе воспитания и выращивания нейронной сети мозга в его высшем отделе — неокортексе.

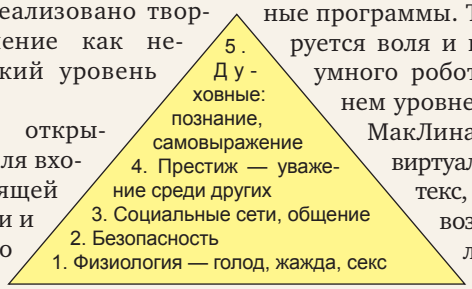
Рассмотрение схемы МакЛина показывает, что мозг развивается путем наслоения новых структур на старые. По-видимому, при моделировании может быть использован такой же подход.

Поэтому реалистическим путем создания умного робота представляется попытка вслед за природой воспроизвести конструктивное устройство мозга МакЛина, а затем воспроизвести перечисленные выше условия зарождения в таком мозге сознания. Основные идеи, которые должны быть реализованы при моделировании умного робота, следующие.

Зарождение мышления в какой-либо системе логических элементов можно только при превышении этой системой определенного уровня сложности. Алгоритм действий, соответствующих определенному уровню так называемой “пирамиды Маслоу” (см. ниже), усложняется по мере повышения уровня мотиваций и претерпевает разрыв для творческого мышления. Это может быть реализовано только для системы, открытой для входящего и исходящего информационных потоков. Только в таком режиме может быть реализовано творческое мышление как неалгоритмический уровень разума.

Обеспечить открытость разума для входящей и исходящей информации можно только при наличии очень сложного тела. Человеческое тело имеет 300 степеней свободы. Тело современных роботов, как правило, не более 50. Чтобы обеспечить интенсивные входной и выходной информационные потоки, тело должно иметь разнообразные датчики, манипуляторы и коммуникационные устройства. Такое тело способно обеспечить не только прием информации от внешней среды, но и взаимодействие со средой. Опыт показывает, что для развития мозга необходимо именно взаимодействие со средой.

Итак, при создании умного робота целесообразно следовать логике построения триединого мозга МакЛина. Рептилоидный уровень мозга необходимо моделировать на базе обычного компьютера без моделирования нейронной сети. На этом уровне с помощью прямого программирования легко задаются мотивационные программы. На верхних уровнях мозга МакЛина моделируется нейронная сеть. Фактически современное моделирование роботов закладывает основы моделирования тела умного робота и рептилоидного уровня его мозга. Моделирование нейронной сети готовит модель неокортекса. При таком подходе с помощью прямого программирования задаются мотивационные программы. То есть формируется воля и цель будущего умного робота. А на верхнем уровне модели мозга МакЛина формируется виртуальный неокортекс, и появляется возможность сделать робота действительно умным.



Пирамида Маслоу

Как может быть создан робот, способный на внеалгоритмические действия? Очевидно, что с помощью прямого программирования мыслителя, способного решать невычислимые задачи, создать невозможно. Эту же мысль можно сформулировать немного по-другому: мыслителя, способного решать невычислимые задачи, невозможно создать без использования неалгоритмических методов. Так как это положение легко доказать от противного, его можно считать теоремой.

При любых конструктивных решениях для верхних уровней мозга МакЛина получить творчески мыслящее существо в соответствии с этой теоремой можно только через выращивание, то есть через *длительный воспитательный процесс*, как это происходит у людей. То есть вырастить умного робота могут только воспитывающие его умные люди. Алгоритм способен родить только алгоритм.

И в заключение вопрос: “Какой ИИ более опасен — алгоритмический или неалгоритмический?”. На взгляд авторов книги [2], вопрос может быть поставлен шире и распространен на людей. Какой человек более опасен — лишенный воли и управляемый кем-то или способный выбирать свой путь самостоятельно? Авторы считают, что история развития человечества давно нашла ответ на этот вопрос…

Литература

1. *http://www.independent.co.uk/news/science/stephen-hawking-transcendence-looks-at-the-implications-of-artificial-intelligence-but-are-we-taking-ai-seriously-enough-9313474.html*.

2. *Шерозия Г.А., Шерозия М.Г.* Человеческий разум, рожденный в сетях искусственных логических элементов, — введение в проект создания нового человека (*www.humanbrain.info*).

3. Умный робот? / Вокруг света, 2015, № 2.

4. *Андреева Е.В., Босова Л.Л., Фалина И.Н.* Математические основы информатики. М.: Бином. Лаборатория знаний, 2005.

Подготовил
Д.М. Златопольский

Играем в двоичную систему счисления

Ю.В. Пашковская,
учитель информатики
гимназии № 1,
г. Жуковский, Московская обл.

Известно, что в обучении игра способна сделать сложное — простым, скучное — азартным, непонятное — очевидным. А коллективная игра тем более.

Игру, о которой пойдет речь в данной статье, можно сделать как элементом урока, так и основой командного соревнования, проводимого, к примеру, в рамках "Недели информатики" или во время другого внеклассного мероприятия.

Суть ее заключается в том, что из восьми учеников формируется команда "Байт". Каждый ее участник моделирует один разряд двоичного числа: если он стоит лицом к зрителям, то его значение равно 1, если спиной — 0*. Требуется закодировать заданное учителем (или ведущим конкурса) число. Например, число 35 схематично должно в результате выглядеть так — см. *рис.* 1.

Заданное число сообщается только одному участнику команды — ее капитану. А тот, в свою очередь, может передать информацию только соседу — и так далее до конца. В этом случае игра начинает напоминать "испорченный телефон", где результат зависит от слаженности действий всех игроков команды. Число считается закодированным, когда последний участник команды поднимет руку вверх. Во время игры пользоваться бумагой и калькулятором запрещается.

В зависимости от того, какой игрок будет назначен на роль капитана, варьируются и правила построения искомой кодировки. Возможны два варианта выбора капитана.

Вариант 1

Каждый игрок в зависимости от занимаемой в команде позиции (разряда) должен определить и запомнить свой "вес": 128, 64, 32, 16, 8, 4, 2 или 1 (разряды считаются слева направо). Капитаном команды считается игрок с наибольшим "весом".

В этом случае каждый игрок по очереди, начиная с капитана, должен действовать согласно алгоритму, иллюстрируемому схемой на *рис.* 2 на с. 46.

Например, пусть требуется изобразить число 35. В этом случае действия членов команды следующие:

1) капитан поворачивается к зрителям спиной (поскольку 35 < 128) и сообщает рядом стоящему игроку число 35 без изменения;

2) второй игрок (с "весом" 64), сравнив свой "вес" с сообщенным ему числом, также поворачивается спиной (35 < 64) и сообщает следующему игроку все то же число 35;

3) третий участник игры (его "вес" — 32) становится к зрителям лицом (так как 35 > 32) и сообщает соседу слева (в данном случае) число 3 (35 – 32);

4) четвертый игрок (с "весом" 16) поворачивается спиной (3 < 16) и сообщает следующему участнику число 3;

5) пятый игрок (с "весом", равным 8) поворачивается спиной (3 < 8) и сообщает соседу справа (в данном случае) число 3;

6) шестой участник (его "вес" — 4) поворачивается спиной (3 < 4) и сообщает соседу число 3;

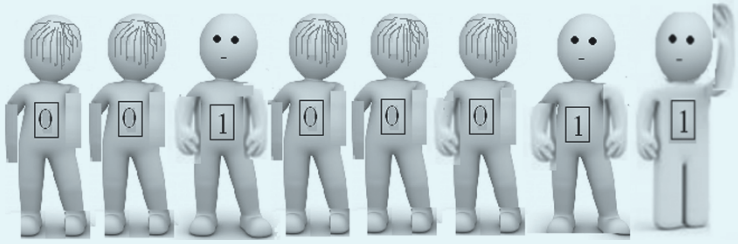


Рис. 1

* Соответствующие значения можно закрепить на одежде участников игры. — **Прим. ред.**

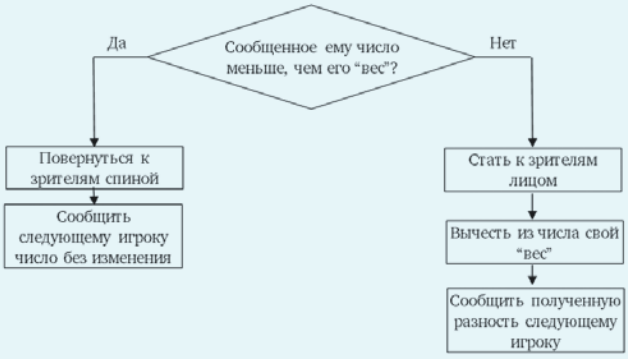


Рис. 2

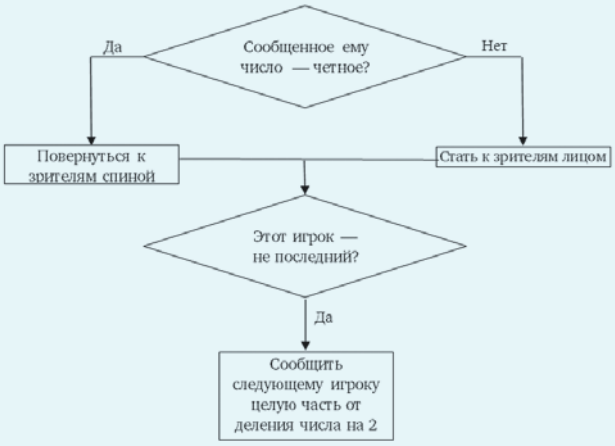


Рис. 3

Рис. 4

7) стоящий предпоследним игрок ("вес" — 2) становится к зрителям лицом (так как 3 > 2) и сообщает последнему игроку число 1;

8) последний, восьмой, игрок также становится лицом к зрителям (его "вес" равен числу) и поднимает руку вверх — заданное число закодировано (см. *рис.* 1).

Вариант 1

В этом случае капитаном считается игрок, изображающий младший разряд (крайний справа, если смотреть со стороны зрителей). Ему же и сообщается исходное число.

Начиная с капитана, каждый игрок по очереди действует согласно следующему алгоритму — см. *рис.* 3.

Например, в случае, когда исходное число равно 35, действия игроков, которых будем нумеровать, справа налево, если смотреть со стороны зрителей, будут выглядеть так:

1) так как 35 — нечетное число, капитан становится к зрителям лицом и сообщает соседу число 17;

2) второй игрок (моделирующий второй справа разряд) поворачивается лицом, так как число 17 — нечетное, и сообщает соседу справа (в данном случае) значение 8;

3) третий участник игры поворачивается спиной (8 — четное число) и сообщает следующему игроку число 4;

4) следующий игрок поворачивается спиной и сообщает соседу число 2;

5) пятый игрок также поворачивается к зрителям спиной и сообщает соседу число 1;

6) так как 1 — нечетное число, шестой участник становится к зрителям лицом и сообщает соседу число 0 (результат целочисленного деления 1 на 2);

7) игрок, моделирующий второй слева бит, поворачивается спиной и сообщает соседу слева (в данном случае) результат целочисленного деления 0 на 2, то есть 0;

8) последний участник, которому было сообщено четное число 0, поворачивается к зрителям спиной и поднимает руку, информируя жюри о том, что число сформировано.

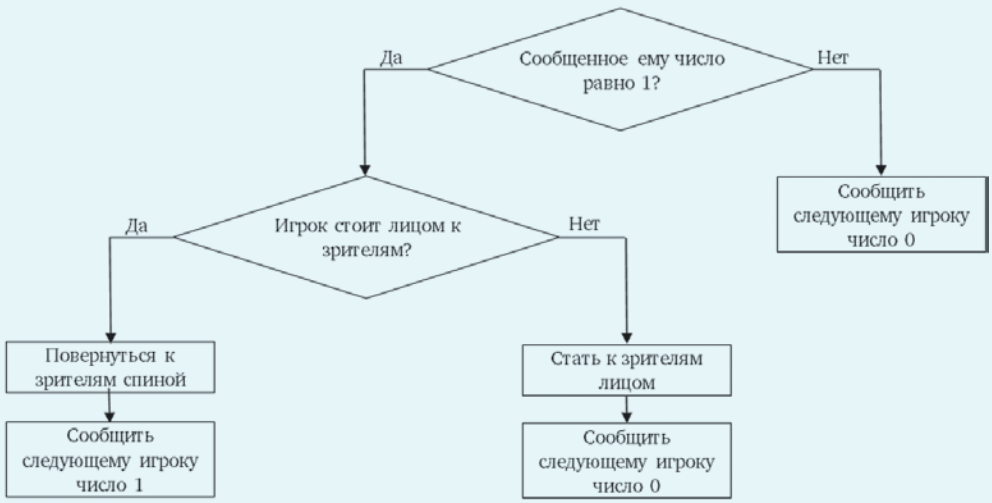


Рис. 5

В игре могут участвовать одновременно несколько команд (“Байт 1”, “Байт 2” и т.д.). В таких случаях оценивается не только правильность полученного результата, но и время выполнения задания.

Дополнительные задания и задания повышенной сложности

1. Умножение на число, представляющее собой степень двойки

После того как команда/команды выстроилась/выстроились правильно, дается дополнительное задание: “Удвоить значение закодированного числа”.

Секрет быстрого решения заключается в том, что умножение на 2 в двоичном коде выражается в приписывании справа цифры 0, а при использовании 8-разрядной записи числа бит старшего разряда пропадает. Это означает, что игроки, начиная со старшего разряда, должны продублировать значение своего “младшего” соседа, а последний игрок (моделирующий младший разряд) поворачивается спиной и поднимает руку.

Внимание! Данное задание может быть предложено, если в первом задании кодировалось число, меньшее 128.

При выполнении заданий на умножение полученного числа на 4, 8 и т.д. действия участников аналогичны с учетом величины сдвига.

2. Кодировка числа, заданного в недесятичной системе счисления

Здесь имеются в виду системы счисления с основанием, равным степени двойки, — четверичная, восьмеричная и шестнадцатеричная. Например, игрокам предлагается представить двоичную кодировку числа 2011_4, или 134_8, или AE_{16}.

Успех команды зависит от быстрого разделения игроков на пары, тройки и четверки (в зависимости от основания исходной системы счисления) и представления каждой цифры исходного числа соответствующим количеством бит. Так, число 2011_4 будет представлено парами 10, 00, 01 и 01, число 134_8 — группами 01, 011 и 100, число AE_{16} — четверками 1010 и 1110.

3. Кодировка отрицательного числа

Для выполнения задания учащиеся должны знать, что отрицательное число представляется в дополнительном коде. Капитан, которому сообщается отрицательное число, сначала должен (один или с другими членами команды) определить его дополнительный код для 8-разрядной сетки, после чего задача сводится к построению полученного кода (см. *рис*. 2 и 3).

4. Увеличение полученного числа на единицу

Для того чтобы выполнить это задание, нужно помнить, что в двоичной системе счисления 1 + 1 = 10, и учитывать перенос единицы в старший разряд.

Алгоритм выполнения этого задания можно представить в виде двух блок-схем.

Для игрока, изображающего самый младший разряд, действия должны соответствовать следующей схеме — см. *рис*. 4 на с. 46.

Для остальных игроков схема действий показана на *рис*. 5.

Ясно, что участник игры, получающий информацию последним, никому ее не передает.

В заключение остается добавить, что, как показывает опыт, подобные конкурсы заражают азартом не только участников команд, но и их болельщиков. Зрители тоже могут быть вовлечены в действо. Для этого они приходят на конкурс с заранее приготовленными плакатами, на которых известные пословицы переделаны в “компьютерные”. Например, “Бит байт бережет”, “Береженого антивирус бережет”, “Не все полезно, что в комп пролезло”. Жюри определяет лучший плакат и награждает его автора (авторов).

КНИЖНЫЙ ШКАФ

Д.М. Златопольский
Системы счисления.
Учебные и занимательные материалы.
М.: ЛЕНАНД, 2015

В книге приведены задачи, фокусы, головоломки и другие увлекательнейшие материалы, связанные с недесятичными системами счисления. Ее материалы можно использовать на уроках, в качестве домашних заданий, на кружках и факультативах, во внеклассной работе.

Книга состоит из 18 глав и содержит 13 приложений. В ней приводятся: задачи разного уровня сложности; методика решения типовых задач на системы счисления, представленных в Едином государственном экзамене по информатике; арифметические и геометрические прогрессии чисел в недесятичных системах; логические и сдвиговые операции; основные принципы создания так называемых “помехоустойчивых” кодов; математические фокусы, головоломки, игры с числами в недесятичных системах счисления.

Все задания, представленные в книге, имеют развивающее значение для интеллекта, формируют общеучебные навыки и способствуют повышению интереса учащихся к математике и информатике. Ко всем заданиям даны ответы и разъяснения.

В приложениях описываются различные методы (в том числе малоизвестные) перевода из одной системы счисления в другую целых чисел и правильных дробей, программы (с методикой их разработки) решения задач, связанных с системами счисления, решением головоломок и демонстрацией фокусов, рассмотренных ранее, а также представлены материалы исторического характера (такие материалы имеются и в основной части книги в виде “врезок”).

Если вам нужны задачи разного уровня сложности по теме “Системы счисления” для уроков и домашних заданий, головоломки, фокусы и игры, связанные с этой темой, материалы для внеклассной работы и проектной деятельности учащихся, то эта книга — для вас. Она, безусловно, будет полезна также учащимся, увлекающимся указанными предметами.

МИР ИНТЕРНЕТА

Хотя раздел "В мир информатики" предназначен для учащихся, редакция понимает, что первыми читателями журнала являются взрослые. Именно вам, уважаемые коллеги, предназначена публикуемая ниже статья. Она подготовлена по материалам одноименной статьи в журнале "Потенциал" № 8 за 2014 год (автор — В.С. Белимова).

В Интернете родились .ДЕТИ

Интернет — несомненное благо и одно из самых выдающихся и полезных изобретений XX века. Но одинаково ли он дружелюбен ко всем пользователям? Насколько стабилен и полезен? С момента своего возникновения Интернет прошел несколько стадий развития и теперь стоит на пороге очередных заметных перемен. Сегодня сложились условия, когда для своего развития Интернет вынужден быть более человечным и дружелюбным.

Подростки и дети — вторая по активности возрастная группа пользователей Всемирной сети. Но много ли информации, учитывающей их интересы, встречается на просторах Интернета? Могут ли дети сами формировать облик современной виртуальности? Как правило, нет.

Общество до самого последнего времени предпочитало просто не замечать детей в Интернете. Часто на стадии начала использования сайта пользователям (среди которых могут быть дети) предлагается заявить о совершеннолетии простым нажатием кнопки. Существуют также специальные программы, которые устанавливаются на домашние и школьные компьютеры, цель которых — радикально ограничить доступ к Сети.

Игнорирование, побуждение ко лжи или ограда виртуального гетто — вот что встречают в Интернете дети в ответ на свой искренний интерес.

Ситуация усугубляется тем, что в своем нынешнем виде Всемирная сеть не защищает ребенка от агрессии и криминальных посягательств со стороны сверстников и взрослых.

Глазами детей

Представьте себе, что вам запретили пользоваться популярными социальными сетями на срок от года до восьми лет.

Представьте, что интернет-СМИ поглупели и стали невообразимо жестокими: теперь малоинтересные сайты "разукрашены" фотографиями, способными ужаснуть вас во глубины души.

Представьте, что родные разрешают пользоваться Интернетом только под присмотром, а Всемирная сеть почти не помогает развить необходимые компетенции.

Так выглядит Интернет глазами детей и подростков.

А вот цитаты, полученные в рамках проекта EU Kids Online, в ходе которого было опрошено почти 10 тысяч юных интернет-пользователей из европейских стран. Дети и подростки 9–16 лет сами рассказали о том, что их беспокоит.

"Facebook показывает страшные вещи, даже если кликаешь на что-то, что не выглядит страшным".

Девочка, 9 лет, Великобритания

"Видео, на которых подростки обижают больных детей, потом они выкладывают это на YouTube".

Девочка, 9 лет, Италия

"Когда незнакомцы пишут мне в Интернете или когда порносайты открываются, хотя я на них не кликаю".

Мальчик, 10 лет, Австрия

"Дети могут испытывать стресс, когда что-то, о чем они прочитали в Интернете и считали правдой, оказывается ложью".

Девочка, 11 лет, Эстония

"Когда кто-нибудь отправляет мне сообщение вроде «я тебя убью» или «я заберу все твои деньги»".

Мальчик, 12 лет, Австрия

"Я нечаянно на что-то подписался в Интернете, и моей маме пришлось заплатить почти 100 евро".

Мальчик, 12 лет, Германия

Как оказалось, детей шокируют сцены жестокости, убийств, издевательств над животными и страшные

сюжеты из новостных хроник. Еще бы, ведь в большинстве случаев это не постановочные кадры, а жестокая реальность, которая вызывает отторжение у любого нормального взрослого человека.

Что делать?

IT-компании и интернет-провайдеры разрабатывают все новые и новые программы для фильтрации, эксперты готовят “черные” и “белые” списки сайтов, депутаты принимают закон “О защите детей от информации, которая причиняет вред их здоровью и развитию”. Все эти меры направлены на одну и ту же благородную цель: очистить Интернет от контента, вредного для неокрепшей детской психики.

Но Интернет — живая среда (почти как дикая природа!) и не поддается жесткому регулированию. Он не стоит на месте, постоянно развивается и стремительно растет. По данным аналитиков, за 2013 год Сеть выросла приблизительно на треть, и сейчас в Интернете более миллиарда (!) сайтов.

Остановить этот процесс невозможно. Интернет давно перестал быть далекой виртуальной реальностью, связь между онлайном и офлайном практически стерта. А эксперты прогнозируют, что скоро наступит “эра Интернета вещей”.

Поэтому запретить что-то в Интернете — это все равно что птицам запретить петь, морю запретить приливы и отливы, Солнцу запретить светить. Изменить большой Интернет под детские потребности невозможно, да и не нужно. Эксперты единодушны в том, что все попытки регулирования Интернета делаются бессистемно. С другой стороны, реализовать полную техническую изоляцию сложно и политически неправильно.

Интерактивность и виртуальность Интернета привлекают детей всех возрастов. Но специализированных ресурсов пока что нет. Социальные сети при регистрации спрашивают возраст — то есть если человеку меньше 14, то он не может войти на сайт. Но многих ли это останавливает? Они просто приписывают себе года!

Получается, что взрослые учат детей врать, вместо того чтобы объяснять, рассказывать об опасностях, показывать, что можно делать, а что лучше в целях безопасности не делать. Запретами ничего не решишь.

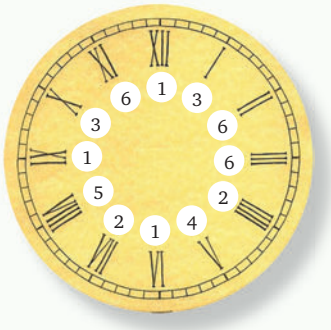
Что же делать? Ответ на этот вопрос дает домен верхнего уровня .ДЕТИ, предназначенный для использования детьми и подростками. Он был делегирован России 27 февраля 2014 года. Домен .ДЕТИ призван способствовать развитию познавательного, интересного и безопасного Интернета для несовершеннолетних пользователей и стать платформой, объединяющей русскоязычные сайты для детей и родителей. Миссия домена заключается в содействии повышению качества использования Всемирной сети Интернет детьми и подростками путем создания “интернет-пространства доверия”, консолидирующего качественный и привлекательный, развлекательный и образовательный интернет-контент, и делающего пребывание детей и подростков в сети Интернет комфортным и безопасным. Координацией деятельности домена является Фонд “Разумный Интернет”. По состоянию на январь 2015 года, в домене .ДЕТИ зарегистрировано более 760 сайтов.

Более подробно о домене будет рассказано в следующих выпусках журнала.

ЗАДАЧНИК

Календарь из часов

Хотя 2015 год начался несколько месяцев назад, давайте сделаем календарь на этот год, но календарь — необычный, похожий на часы. Нарисуем на бумаге круг и впишем в него римские числа I, II, …, XII, как на циферблате часов[1]:



[1] Конечно, можно вписать и числа с арабскими цифрами.

Рядом с каждым числом (ближе к центру) запишем числа в квадратиках так, как показано на рисунке. Все — календарь готов! Преимуществом нашего календаря над обычным является его компактность. В последнем записаны 365 чисел, большинство из которых двузначные, а в созданном нами — всего 24 числа (и большинство однозначных).

С помощью нашего календаря можно решать разные задачи. Рассмотрим их.

Будем считать, что римские числа на циферблате — это номера месяцев (I — январь, II — февраль, …, XII — декабрь). Примем также следующие порядковые номера дней недели:

День недели	Номер
Понедельник	1
Вторник	2
…	
Суббота	6
Воскресенье	0 (условно)

Задача 1

Требуется узнать, какой день недели будет 26 октября 2015 года.

Решение

Октябрь — 10-й месяц. Против римского числа X стоит 3. Прибавляем к дате (26) число в квадратике (3): 26 + 3 = 29. Определяем остаток от деления 29 на 7 — получим 1. Это и есть порядковый номер дня недели. Итак, 26 октября 2015 года будет понедельник.

Задача 2

Вася родился 17 мая. В какой день недели он будет праздновать свой день рождения в 2015 году?

Решение

Май — 5-й месяц. Против числа V стоит 4. Прибавляем к дате (17) число в квадратике (4): 17 + 4 = 21. В остатке от деления 21 на 7 получим 0. Остаток и есть порядковый номер дня недели. Значит, день рождения Васи будет в воскресенье.

Задача 3

Какие числа соответствуют вторникам в августе 2015 года?

Решение

К числам 0, 7, 14, 21 и 28 прибавляем порядковый номер дня недели (2), а затем вычитаем число, стоящее в квадратике рядом с месяцем (для августа — 5):

0 + 2 – 5 = –3
7 + 2 – 5 = 4
14 + 2 – 5 = 11
21 + 2 – 5 = 18
28 + 2 – 5 = 25

Следовательно, вторники приходятся на 4-, 11-, 18-е и 25 августа.

Задача 4

Занятия в кружке по программированию в апреле 2015 года назначены на четверг. Определить числа, в которые будут проходить занятия кружка.

Решение

К числам 0, 7, 14, 21 и 28 прибавляем порядковый номер дня недели (4), а затем вычитаем число, стоящее в квадратике рядом с месяцем (для апреля — 2):

0 + 4 – 2 = 2
7 + 4 – 2 = 9
14 + 4 – 2 = 16
21 + 4 – 2 = 23
28 + 4 – 2 = 30

Итак, занятия состоятся 2-, 9-, 16-, 23-го и 30 апреля.

Задания для самостоятельной работы

1. Подготовив описанный календарь, определите:

а) в какой день недели будет день вашего рождения в 2015 году;

б) какие числа соответствуют субботам в августе 2015 года.

2. Подумайте, как составлен описанный календарь.

Указание по выполнению. Проанализируйте дни недели, соответствующие первому числу каждого месяца.

3. Будет ли действовать созданный календарь в другие года, например, в 2016-м? Можно ли на основе календаря 2015 года составить календарь на следующий год? Как, имея календарь на 2014 год, получить аналогичный календарь на 2015 год?

Литература

1. *Розетуллер В.М.* Часы-календарь. / Квант, 1973, № 12.

ШКОЛА ПРОГРАММИРОВАНИЯ

Когда человек хочет передвинуть гору, он начинает с того, что убирает маленькие камни.

Китайская мудрость

Решаем задачи о днях недели

Разработаем две программы (как принято в нашем издании, — на школьном алгоритмическом языке):

1) которая по заданной дате года определяет соответствующий день недели;

2) которая по заданным месяцу и дню недели находит соответствующие числа месяца.

При этом используем методику, описанную в статье “Календарь из часов”.

В первой программе информацию о заданной дате будем хранить с помощью двух переменных величин:

день — день месяца;

месяц — номер месяца.

Искомую величину обозначим *день_недели*, а числа в квадратиках (см. рисунок в указанной статье) будем хранить в массиве *числа* из 12 элементов.

Кроме того, для решения задачи понадобится рассчитать порядковый номер искомого дня недели. Имя переменной в программе — *номер*.

Вся программа имеет вид:

```
алг День_недели
нач цел день, месяц, n, номер
    лит день_недели,
    цел таб числа[1:12]
|Заполнение массива числа (см. рисунок выше)
числа[1] := 3
числа[2] := 6
…
числа[11] := 6
числа[12] := 1
|Ввод информации о дне года
вывод нс, "Введите день месяца "
```

```
ввод день
вывод нс, "Введите номера месяца "
ввод месяц
номер := mod(день + числа[месяц], 7)
выбор
  при номер = 1: день_недели := "понедельник"
  при номер = 2: день_недели := "вторник"
  …
  при номер = 6: день_недели := "суббота"
иначе
  день_недели := "воскресенье"
все
|Вывод ответа
вывод нс, "Это день – ", день_недели
кон
```

Вторая программа предназначена для решения задач, аналогичных задачам 3 и 4, рассмотренным в статье “Календарь из часов”. В ней исходными данными являются:

1) номер месяца (имя величины — *месяц*);

2) номер дня недели (*номер_дня_недели*),

а искомая величина *день* — дни (даты) месяца.

Прежде чем представлять программу, заметим, что в ней рассматриваются числа 0, 7, 14, 21 и 28 (для этого можно использовать оператор цикла с параметром). Обратим внимание также на то, что искомые даты месяца выводятся не безусловно, а только при определенном условии.

```
алг День_недели
нач цел месяц, номер_дня_недели, день, i,
цел таб числа[1:12]
|Заполнение массива числа (см. выше)
числа[1] := 3
числа[2] := 6
…
числа[12] := 1
|Ввод исходных данных
вывод нс, "Введите номера месяца "
ввод месяц
вывод нс, "Введите номер дня недели
           0 – воскресенье,
           1 – понедельник, …,
           6 – суббота"
ввод номер_дня_недели
нц для i от 1 до 5
  день := (i – 1) * 7 +
          номер_дня_недели – числа[месяц]
  если день > 0
    то
      вывод нс, день
  все
кц
кон
```

Задание для самостоятельной работы

Подготовив описанные программы на языке программирования, который вы изучаете, определите:

а) на какой день недели в 2015 году приходится 26 октября;

б) в какой день недели будет день вашего рождения в 2015 году;

в) какие числа соответствуют субботам в августе 2015 года.

Программы и ответы присылайте в редакцию.

MICROSOFT EXCEL УГЛУБЛЕННО

Решаем задачи о днях недели с помощью электронной таблицы

Решим также обе рассмотренные задачи средствами электронной таблицы Microsoft Excel или аналогичной.

1. Задача определения дня недели

Ячейки в верхней части листа оформим для ввода исходных данных и вывода результата:

	A	B
1	**Определение дня недели по дате**	
2	Введите число месяца	26
3	Введите номер месяца	10
4	Соответствующий день недели –	
5		

Остальные значения носят вспомогательный характер, и для их расчета можно использовать ячейки вне зоны видимости листа. Таких значений — два.

Во-первых, для расчета номера искомого дня недели нужно знать число в квадратике на нашем календаре, которое записано рядом с заданным номером месяца. Это число можно определить с использованием функции ВЫБОР. Эта функция имеет следующий общий вид:

ВЫБОР(номер_индекса; значение1; значение2;…),

— где *значение1*, *значение2*, … — аргументы-значения, из которых выбирается значение функции, а *номер_индекса* — это номер выбираемого аргумента-значения. Аргумент *номер_индекса* должен быть числом от 1 до 29, формулой или ссылкой на ячейку, содержащую число в диапазоне от 1 до 29. Если номер_индекса равен 1, то функция ВЫБОР возвращает значение 1; если он равен 2, то функция ВЫБОР возвращает значение 2 — и так далее.

Следовательно, в нашем случае формула для числа, соответствующего номеру месяца в ячейке B3, такая:

=ВЫБОР(B3;3;6;6;2;4;1;2;5;1;3;6;1)

Эту формулу запишем, как условились, вне зоны видимости листа — например, в ячейке P2.

Вторая вспомогательная величина — номер искомого дня недели.

Согласно статье в рубрике “Задачник”, формула для расчета этого номера имеет вид:

=ОСТАТ(B2+P2;7)

Ее запишем в ячейке P3.

После этого искомое значение дня недели в ячейке B4 может быть найдено по номеру дня также с применением функции ВЫБОР. Правда, здесь имеется “подводный камень” — воскресенье, условно соответствующее номеру 0, не может быть определено (в функции ВЫБОР аргумент номер_индекса не может быть меньше 1). Выходом является использование функции ЕСЛИ по следующей логике:

```
если номер дня недели = 0
  то
     искомый день недели – воскресенье
иначе
     используем функцию ВЫБОР
все
```

То есть в одной из “ветвей” функции ЕСЛИ следует использовать функцию ВЫБОР.

Соответствующую формулу запишите самостоятельно.

Дополнительные задания для самостоятельной работы

1. Оформив лист по описанной методике, определите:

а) на какой день недели в 2015 году приходится 26 октября;

б) в какой день недели будет день вашего рождения в 2015 году.

2. Подготовьте лист для решения обсуждаемой задачи для случая, когда дата года задается не как два числовых значения, а как одно значение типа дата, например:

	A	B
1	**Определение дня недели по дате**	
2	Введите дату 2015 года	26.10.2015
3		
4	Соответствующий день недели –	
5		

Для этого вспомните статью “Работа с датами в электронных таблицах” в выпуске “В мир информатики” № 202 (“Информатика” № 11/2014).

2. Задача определения чисел месяца

Верхнюю часть листа оформим так:

	A	B
1	**Определение дат в месяце**	
2	Введите номер месяца	
3	Введите номер дня недели (0 – воскресенье, 1 – понедельник, …, 6 – суббота)	
4		
5	Даты в этом месяце:	
6		
7	…	

Искомые значения должны выводиться в ячейках A6:A10 (возможно, не во всех, если их меньше пяти).

Как и в предыдущей задаче, вспомогательные величины будем определять в ячейках вне зоны видимости листа. Например, число в квадратике на календаре, которое записано рядом с заданным номером месяца, — в ячейке P3. Здесь также используется функция ВЫБОР.

В столбце O можно записать пять значений, а в столбце P — рассчитать возможные даты с использованием значений из столбца O и из ячеек P3 и B3:

	A	B	…	O	P
…					
5	Даты в этом месяце:				
6				0	
7				7	
8				14	
9				21	
10				28	

Из возможных значений (в столбце P) в ячейках A6:A10 должны выводиться только положительные.

Задание для самостоятельной работы

Оформив лист по описанной методике, определите, какие числа соответствуют субботам в августе 2015 года.

Результаты самостоятельной работы по трем последним статьям (можно не все) присылайте в редакцию. Фамилии всех приславших будут опубликованы.

ЗАДАЧНИК

Ответы, решения, разъяснения к заданиям, опубликованным в разделе “В мир информатики” в ноябре 2014 года

(*Продолжение. Начало см. в предыдущем выпуске*)

Числовые ребусы в троичной системе

Ответы

1. 2 + 11 = 20. **2.** 10 + 10 = 20. **3.** 1 + 20 = 21.

Правильные ответы представили:

— Бульбова Лидия, средняя школа г. Пионерский Калининградской обл., учитель **Багрова О.А.**;

— Воронова Анжелика и Хомутов Андрей, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Довгань Алексей и Назаренко Елена, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Казанец Елена, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Коробов Сергей, Марков Алексей и Яснов Федор, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Крысанов Виктор, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Рыжов Антон, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Стороженко Степан, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Чукмасова Надежда, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**

Задача "Яблоки"

Напомним условие: "После того как каждое из 52 яблок разрезали пополам, стало 134 половинки. В какой системе счисления велся счет при этом?"

Решение

Данные условия можно записать так:

$$\begin{array}{cccc} & 5 & 2 \\ + & 5 & 2 \\ \hline 1 & 3 & 4 \end{array}$$

Анализ показывает, что $5 + 5 = 13$ только в семеричной системе счисления ($q + 3 = 10,\ q = 7$).

Ответ: в семеричной.

Правильные ответы представили:

— Антонов Антон и Удалова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Воронова Анжелика, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Довгань Алексей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Казанец Елена и Прохорова Александра, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Карданова Аминат, Коломина Нонна, Медяникова Аделина, Остроухова Валерия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Коробов Сергей, Марков Алексей и Яснов Федор, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Крысанов Виктор и Щукарева Анна, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Незванцев Сергей, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Рыжов Антон, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**

Кроссворд

Ответы

По горизонтали: 1. Три. 4. Ор. 5. Контроль. 7. Тип. 10. Дно. 11. Процессор. 13. Диалог. 14. Опрос. 15. Статус. 16. Клон. 17. Тело. 19. Растр. 22. Состав. 24. Форматирование. 25. Стиль. 26. Арена.

По вертикали: 1. Транзистор. 2. Икс. 3. Процент. 6. Листинг. 8. Протокол. 9. Бодо. 12. Регистр. 16. Клавиша. 18. Основа. 20. Адрес. 21. Трафик. 23. Тишь.

Ответы прислали:

— Андреев Владислав, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Владимиров Константин, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Власова Ольга и Гурова Илона, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Иванов Николай и Смолов Станислав, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Новиков Сергей, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Филиппенко Михаил, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Чернова Ксения, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**;

— Шалимов Владимир, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Шпаченко Алексей, г. Пенза, школа № 512, учитель **Гаврилова М.И.**

Три вопроса (рубрика "Поиск информации")

Ответы

1. Уильям Боинг построил авиазавод в городе Сиэтл.

2. В высшей лиге чемпионата СССР по футболу 1979 года со счетом 5:2 закончился матч между командами "Спартак" и ЦСКА.

3. Собака, о которой главный герой романа Владимира Набокова "Лолита" рассказывал, что "…мы едва не сбили навязчивую пригородную собаку (из тех, что устраивают засады автомобилям)", была породы сеттер.

Ответы представили:

— Барановская Татьяна и Жукова Ирина, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Деревянченко Дарья, г. Ярославль, школа № 33, учитель **Ярцева О.В.**;

— Донникова Анна, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Евграфова Ксения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Есипова Мария, Круглякова Мария и Яснова Дарья, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

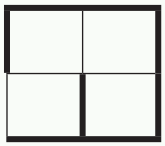
— Иванова Ксения и Мухина Светлана, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Маркина Ирина, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Хорькова Анна, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**

Головоломка “Робот в лабиринте”

Напомним, что необходимо было дать исполнителю Робот такую последовательность команд, по которой он вышел бы из лабиринта, показанного ниже, где бы ни находился. (Положение Робота в лабиринте неизвестно.) Он понимает четыре команды: вверх, вправо, вниз и влево — и, исполняя их, смещается на одну клетку в соответствующем направлении, если ему ничего не мешает, и ничего не делает — если мешает стена.



Ответ: вверх, влево, вниз, влево.

Ответы прислали:

— Абаев Иван, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Андреев Владислав, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Абдувахидова Алина, Абдувахидова Софья, Галкина Эвелина, Киселев Владислав, Милушкин Дмитрий, Строкин Константин и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Бойко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Глушаков Андрей и Листова Елена, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Евграфова Ксения, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Михайлов Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**

Головоломка “Неисправный калькулятор”

Напомним, что требовалось определить значения (оба числа, над которыми проводится действие, а также результат), которые должны быть отображены на калькуляторе. На нем цифры изображаются с помощью так называемых “7-сегментных” индикаторов, но калькулятор — неисправный, и у него горят не все сегменты.

Ответ: $35 \times 54 = 1890$.

Правильный ответ прислали:

— Бойко Степан, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Гируцкий Павел, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Глушаков Андрей, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Живило Андрей, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Зубов Владислав, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Макаров Василий, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Михайлов Иван, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**

Задача “Банкир и конверты”

Напомним условие: “Некоему банкиру нужно было встретиться с важным клиентом, которому он должен выдать наличными заранее неизвестную сумму от 1 до 1 000 000 у.е. Чтобы не тратить время на отсчитывание денег, банкир дал указание своим кассирам заготовить некоторое количество конвертов с деньгами, на которых написаны содержащиеся в них суммы, чтобы потом просто отдать клиенту один или несколько конвертов, в которых и будет содержаться требуемая им сумма.

1. Какое наименьшее количество конвертов необходимо иметь?

2. Какова будет в этом случае полная сумма во всех конвертах?

3. Как, используя подготовленные конверты, набрать требуемую сумму?”

Ответы прислали:

— Бородкин Сергей, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Дегтярь Анатолий и Новиченко Владимир, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**;

— Иванов Николай, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Карданова Аминат, Коломина Нонна, Медяникова Аделина, Остроухова Валерия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.**;

— Милушкин Дмитрий, Рыжов Антон и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Незванцев Сергей, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Новиков Сергей и Хромченкова Елизавета, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Торопов Александр, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Филиппенко Михаил, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**

Решение

Конечно, можно просто заготовить конверты со всеми суммами от 1 до 1 000 000. Но где взять столько конвертов? (Да и зачем тратить деньги на них? ☺)

Есть более рациональный подход. Надо положить в первый конверт 1 у.е., а в каждый следующий класть вдвое большую сумму, чем в предыдущий. Тогда, например, в 5-м конверте будет 16 у.е., в 6-м — 32 и т.д. Проведя расчеты, например, на калькуляторе, можно установить, что в 18-м конверте будет 131 072 у.е., …, в 21-м — 1 048 576 у.е., но он уже явно не понадобится, а вот 20-й конверт с 1 048 576/2 = 524 288 у.е. может и пригодиться. Общее количество денег во всех конвертах составит 1 048 575.

Набрать же любую требуемую сумму можно так:

1) перевести число-сумму в двоичную систему счисления;

2) взять конверты, номера которых совпадают с номерами тех разрядов двоичной записи, в которых имеются единицы (нумерация разрядов — справа налево, начиная с 1).

Например, для набора суммы 231 820 у.е. следует использовать конверты с номерами 1, 3, 4, 8, 9, 12, 16, 17 и 18 ($231\,820_{10} = 111000100110001100_2$). Удобно, не правда ли? Это еще один пример полезности двоичной системы счисления ☺.

Новые задачи

1. При проведении описанных расчетов некоторые вычисления являются явно лишними и их можно не проводить.

Если в k-м конверте сумма денег составляет s, то в каком конверте будет сумма:

1) в 4 раза большая;

2) в 8 раз большая;

3) в 2^s раз большая;

4) равная s^2?

2. Установите, для каких значений сумм, не превышающих 25, можно рассчитать требуемое минимальное число конвертов “в уме”.

3. Как рассчитать сумму $1 + 2 + 4 + \ldots + 2^k$, не используя многочисленные операции сложения? Например, чему равна сумма $1 + 2 + + 2^2 + \ldots + 2^9$, если $2^{10} = 1024$?

Решение задания 4-го тура конкурса № 111 “Переправы”

Напомним, что предлагалось решить следующую задачу: “Две семьи (в каждой муж, жена и сын) хотят переправиться через реку. Есть двухместная лодка. Грести может всего один человек — один из мужей. Сыновья могут быть на берегу только вместе с кем-нибудь из взрослых. Женщины боятся быть на берегу, если там нет лиц мужского пола. Как им всем переправиться на другой берег?”.

Решение

Обозначим членов семей первыми буквами, умеющего грести — большой буквой, остальных — малыми. Схема переправы такая:

Мм →,
М ←,
Мж→,
М ←,
Мс →,
Мм ←,
Мж →,
М ←,
Мс →,
М ←,
Мм →.

Ответы представили:

— Абдувахидова Алина, Абдувахидова Софья и Хозин Марат, Владимирская обл., г. Струнино, школа № 11, учитель **Волков Ю.П.**;

— Багрова Анастасия, Батурин Илья, Дикий Данил, Ионкин Илья, Кацубо Алексей, Лифенцев Владислав, Луцук Максим, Мисюра Алексей, Пак Александра, Панасенко Дарья, Приходько Геннадий, Сысоев Александр и Шикалович Ростислав, средняя школа г. Пионерский Калининградской обл., учитель **Багрова О.А.**;

— Байкова Римма, Дубинина Анна и Левченко Ирина, средняя школа села Восточное Нижегородской обл., учитель **Долгова Г.А.**;

— Василенко Татьяна и Ухин Станислав, средняя школа села Горелово Тамбовской обл., учитель **Шитова Л.А.**;

— Васина Светлана, Макаренко Виталий и Хомутова Евгения, средняя школа деревни Муравьево, Вологодская обл., учитель **Муравьева О.В.**;

— Владимиров Виталий и Яковлева Анастасия, основная школа села Именево, Республика Чувашия, Красноармейский р-н, учитель **Тимофеева И.А.**;

— Водальчук Михаил, гимназия г. Шелехова, Иркутская обл., учитель **Водальчук С.А.**;

— Гагарин Валерий и Кириллов Иван, Республика Карелия, г. Сегежа, школа № 6, учитель **Соколова В.Н.**;

— Дибров Сергей, средняя школа поселка Осиновка, Алтайский край, учитель **Евдокимова А.И.**;

— Захарова Юлия, Смоленская обл., г. Демидов, школа № 1, учитель **Кордина Н.Е.**;

— Иванов Алексей, Свердловская обл., Красноуфимский р-н, Тавринская средняя школа, учитель **Ярцев В.А.**;

— Калинина Ирина, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**;

— Карданова Аминат, Коломина Нонна, Медяникова Аделина, Остроухова Валерия и Уткина Ксения, Ставропольский край, Кочубеевский р-н, станица Барсуковская, школа № 6, учитель **Рябченко Н.Р.** (все перечисленные читатели правильно выполнили также задания 3-го тура конкурса);

— Красненков Александр и Пискунова Полина, средняя школа поселка Ерофей Павлович, Амурская обл., Сковородинский р-н, учитель **Краснёнкова Л.А.**;

— Ледин Роман, г. Ростов-на-Дону, лицей № 56, учитель **Ли В.М.**;

— Лежнева Александра, г. Пенза, школа № 512, учитель **Гаврилова М.И.**;

— Ломтев Павел и Рыжиков Антон, средняя школа села Сердар, Республика Марий Эл, учитель **Чернова Л.И.**;

— Перешнев Алексей, г. Архангельск, школа № 9, учитель **Дудина Т.В.** (Алексей также правильно решил обе задачи 3-го тура конкурса);

— Цуцков Илья, Ардатовский коммерческо-технический техникум, поселок Ардатов Нижегородской обл., преподаватель **Зудин В.П.**;

— Чернова Ксения, Республика Карелия, поселок Надвоицы, школа № 1, учитель **Каликина Т.В.**;

— Щегольков Дмитрий, средняя школа поселка Новопетровский Московской обл., учитель **Артамонова В.В.**

Задания первых четырех туров конкурса № 111 “Переправы” правильно выполнили и также стали его участниками Авдеев Даниил, Попов Тимур и Черноусова Ирина, г. Воронеж, лицей № 2, учитель **Комбарова С.И.**

Напомним, что конкурс проводится в несколько туров, а его итоги будут подводиться с учетом всех туров в целом.

Три министра

Министры иностранных дел России, США и Китая обсудили за закрытыми дверями проекты международного соглашения, представленные каждой из стран. Отвечая затем на вопрос журналистов: “Чей именно проект был принят?”, министры дали такие ответы:

— Россия: “Проект не наш, проект не США”;

— США: “Проект не России, проект Китая”;

— Китай: “Проект не наш, проект России”.

Один из них (самый откровенный) оба раза говорил правду; второй (самый скрытный) оба раза говорил неправду, третий (осторожный) один раз сказал правду, а другой раз — неправду. Определите, представителями каких стран являются откровенный, скрытный и осторожный министры.

Круг и Квадрат

Жили две фигуры — Круг и Квадрат. На улице, где они жили, стояли три дома: с окном и трубой; с окном, но без трубы; с трубой, но без окна. Круг и Квадрат жили в домиках с окнами, Квадрат любил тепло и часто топил печку. Кто в каком домике жил?

Задача предназначена для учащихся 1–7-х классов.

КРЕПКИЙ ОРЕШЕК

В этой рубрике, как всегда, проводится разбор задач, решение которых вызвало трудности.

Задачи из сентябрьского выпуска “В мир информатики”

Задача “Точные квадраты”

Напомним, что следовало ответить на вопрос, при каких основаниях систем счисления k являются точными квадратами числа:

1) 121_k; 2) $12\,321_k$.

Ответ

1) учитывая, что точный квадрат есть произведение некоторого числа на самого себя, рассмотрим такое произведение для числа 11 (записанного в некоторой системе счисления):

$$\begin{array}{r} \times\ 1\ 1 \\ \hline 1\ 1 \\ 1\ 1\ \\ \hline 1\ ?\ 1 \end{array}$$

Далее возникает вопрос: “В какой системе счисления можно получить результат умножения, равный 121 (числу из условия), то есть когда 1 + 1 = 2?” Ответ здесь такой — при любом основании системы $k > 2$. Полученное условие и является решением задачи;

2) при любом $k > 3$, так как при этом

$$12\,321_k = 111^2{}_k.$$

Была предложена также новая задача: “Определите, при каких основаниях систем счисления k является точным квадратом число $123\,454\,321_k$?”

Указание по выполнению

Найдите квадрат числа 11111:

$$\begin{array}{r} \times\ 1\ 1\ 1\ 1\ 1 \\ 1\ 1\ 1\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 1 \end{array}$$

…

Задача “ID-номер”

Напомним условие: “При заключении договора на предоставление услуг каждому абоненту — пользователю сети присваивается уникальный 12-значный идентификационный номер — так называемый “ID-номер”, позволяющий идентифицировать пользователя не только по, например, его имени (логину) или IP-адресу. ID-номер предназначен также для пополнения баланса пользователя через различные платежные системы.

Предположим, что ID-номер представляет собой последовательность символов и используется посимвольное кодирование. При этом каждый символ кодируется минимально возможным количеством бит, а весь номер кодируется минимально возможным количеством байт. Исследуя количество требуемой памяти для описанного кодирования, получили зависимость:

Количество символов в номере	1	2	3	4	5	6	7	8	9	10	…
Количество байт для кодирования номера	1	2	2	3	4	4	5	5	6	7	

Как бы вы описали алфавит для такого кодирования ID-номера (количество символов и т.д.)?".

Приведем начало анализа.

Прежде всего можно понять, что для кодирования одного символа требуется меньше одного байта, — при одном байте (восьми битах) количество байт для кодирования всего номера было бы равно количеству символов в номере. С другой стороны, один символ кодируется более чем четырьмя битами, так как при четырех битах таблица имела бы вид:

Количество символов в номере	1	2	3	4	5	6	7	8	9	10
Количество байт для кодирования номера	1	1	2	2	3	3	4	4	5	5

Предлагаем читателям исследовать оставшиеся возможные варианты кодирования одного символа и найти подходящее значение, а также число используемых для кодирования символов.

Задача "Перестановка двойки"

Напомним условие: "Некоторое число оканчивается на 2. Если эту цифру перенести в начало числа, то оно удвоится. Найдите наименьшее такое число".

Приведем начало решения.

Пусть искомое число имеет вид $x2$, или $10x + 2$, где x — некоторое число. Соответствующее искомому число с двойкой в начале назовем "новое число".

Ясно, что оно (*новое число*) оканчивается на 4 (оно в два раза больше искомого, которое оканчивается на 2). Однако проверка всех таких чисел на соответствие условию — дело очень трудоемкое. Забегая вперед, скажем, что для нахождения искомого числа придется проверить более чем 10^{10} (!) чисел.

Будем рассуждать так. Ясно, что *новое число* имеет вид $2x$. Рассмотрим варианты.

1. Если x — однозначное число, то *новое число* равно $20 + x$. Тогда, согласно условию, можем записать:

$2 \cdot (10x + 2) = 20x + 4 = 20 + x$, или $19x = 16$.

Но для такого уравнения подходящих целых чисел x нет.

2. Если x — двузначное число, то *новое число* равно $200 + x$. Тогда имеем:

$2 \cdot (10x + 2) = 200 + x$, или $19x = 196$.

В этом случае также нет подходящих значений.

3. Аналогично при трехзначном x:

$20x + 4 = 2000 + x$, или $19x = 1996$.

Итак, для решения задачи нам нужно найти минимальное *целое* x, являющееся корнем линейных уравнений вида $19x = 16$, $19x = 196$, $19x = 1996$, …, $19x = 1(99…9)6$. Для этого мы можем использовать электронную таблицу Microsoft Excel или подобную программу (хотя, опять забегая вперед, скажем, что для "ручного" поиска понадобится проверить всего 10 чисел; но лучше пусть все сделает компьютер ☺). Общий вид листа такой:

	A	B	C
1		Правая часть уравнения	*x*
2	20	16	
3	200	196	
4	2000	1996	
…			
10	2000000000	1999999996	
11	2E+10	19999999996	
12	2E+11	2E+11	
…			
21	2E+20	2E+20	

Найдите соответствующее значение x.

Задачи из ноябрьского выпуска "В мир информатики"

Задача "Кто сказал правду?"

Напомним условие: "Билет на проезд в общественном транспорте считается счастливым, если в его шестизначном номере сумма первых трех цифр равна сумме последних трех цифр.

Как-то между тремя приятелями состоялся такой разговор:

— Однажды мне попался счастливый билет, у которого каждая цифра, начиная со второй, была либо вдвое больше, либо вдвое меньше предыдущей, — заявил Петя.

— А мне, помню, достался счастливый билет, у которого каждая цифра, начиная со второй, была либо вдвое больше, либо втрое меньше предыдущей, — сообщил Коля.

— А у моего счастливого билета каждая цифра, начиная со второй, была либо вдвое больше, либо вчетверо меньше предыдущей, — сказал Вася.

Чьи слова могли быть правдой?".

Анализ

Мы считаем, что билета с номером 000000 не существует, иначе задача теряет смысл. Поэтому ни одна из цифр билета не должна быть равна 0.

Вася мог сказать правду, например, если у него был билет 124124. А вот возможность правдивого

утверждения остальных ребят требует более серьезного анализа.

Предположим, что Коля сказал правду: каждая цифра, начиная со второй, либо вдвое больше, либо втрое меньше предыдущей.

Тогда, если первая цифра не делится на 3, то каждая следующая цифра должна быть вдвое больше предыдущей, и последняя цифра равна первой, умноженной на $2^5 = 32$, но это явно больше 10.

Если первая цифра 3 или 6, то, идя к последней цифре (делаем пять умножений на 2 или делений на 3), мы можем максимум один раз поделить на 3, но тогда последняя цифра будет не меньше чем $3/3 \cdot 16 = 16$, что опять же больше 10.

Вариант, при котором первая цифра — 9, исследуйте самостоятельно.

Предположим, что Петя сказал правду: каждая цифра, начиная со второй, либо вдвое больше, либо вдвое меньше. Посмотрим на остатки всех цифр при делении на 3.

Пусть первая цифра делится на 3 (то есть равна 3, 6 или 9). Таких билетов всего два: 363636, 636363, и ни один из них не счастливый.

Вариант, при котором первая цифра не делится на 3, также предлагаем читателям исследовать самостоятельно. Найдите также второй номер, который соответствует утверждению Васи.

Редакция благодарит Владислава Киселева и Марата Хозина, учеников школы № 11 г. Струнино Владимирской обл. (учитель **Волков Ю.П.**), приславших правильный ответ. Владислав и Марат будут награждены дипломами. Поздравляем!

Задача “Кодирование чисел”

Напомним условие: “Для кодирования натуральных чисел с помощью буквенных последовательностей был предложен следующий принцип, основанный на использовании латинских букв: A, B, C и D.

Числам 1, 2, 3 и 4 ставятся в соответствие указанные четыре буквы.

Последующим 16 числам ставятся в соответствие двухбуквенные коды в следующем порядке: 5 = AA, 6 = AB, 7 = AC, 8 = AD, 9 = BA, 10 = BB, …, 18 = DB, 19 = DC, 20 = DD. Аналогично для последующих чисел используются трехбуквенные коды (от 21 = AAA до 84 = DDD), четырехбуквенные и т.д. Укажите буквенный код числа 295.

В ответе нужно записать последовательность из латинских букв. Решение получить, не выписывая все 295 чисел”.

Начало решения

По данным условия можно составить таблицу:

Длина кода, символов	Коды	Всего чисел в группе	Числа в группе	
			От	До
1	A, B, C, D	4	1	4
2	AA, AB, …, DD	16	5	20
3	AAA, AAB, …, DDD	64	21	84
4	AAAA, AAAB, …, DDDD	256	85	340

Из нее видно, что код числа 295 — 4-буквенный, а его порядковый номер в группе равен 295 – 85 + + 1 = 211. В группе чисел с 4-буквенными кодами можно выделить четыре подгруппы:

1) начинающихся на A — 64 числа;
2) начинающихся на B — 64 числа;
3) начинающихся на C — 64 числа;
4) начинающихся на D — 64 числа.

Наше число — в последней подгруппе (в трех первых всего 192 числа). Значит, первая буква кода — D.

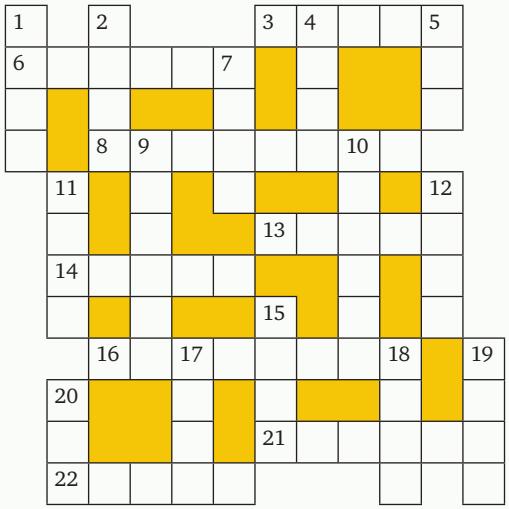
Чтобы найти вторую букву кода, надо рассмотреть четыре части найденной подгруппы. Номер числа 295 в подгруппе: 211 – 193 + 1 = 19. Соответствующую вторую и остальные буквы кода найдите самостоятельно.

Антон Рыжов, ученик школы № 11 г. Струнино Владимирской обл. (учитель **Волков Ю.П.**), приславший правильный ответ с обоснованием, будет награжден дипломом. Молодец!

Ответы и решения рассмотренных задач присылайте в редакцию.

“ЛОМАЕМ” ГОЛОВУ

Кроссворд



Решите, пожалуйста, кроссворд.

По горизонтали

3. Один из контактов транзистора типа **МОП** (**М**еталл — **О**ксид — **П**олупроводник).

6. Рассмотрение и при необходимости обработка всех элементов массива.

8. Изображение чего-нибудь, рассказ о ком-нибудь (о чем-нибудь) в письменной или устной форме.

13. Поименованная группа файлов, объединенных по какому-то признаку.

14. Процесс написания текста на компьютере.

16. Совокупность четко определенных правил для решения задачи за определенное число шагов.

21. Указатель места на экране монитора.

22. Так иногда называют столбец в таблице.

По вертикали

1. Непрошенное рекламное сообщение, сетевой “мусор”.

2. Язык программирования (как правило, для начинающих изучать последнее).

4. Жаргонное название результата обработки изображения на устройстве ввода информации в компьютер.

5. Число в системе условных обозначений символов.

7. Разновидность носителя информации.

9. Название клавиши.

10. Ввоз товаров из-за рубежа, а также вставка в документ приложений Windows объектов из других приложений.

11. Часть экрана, занимаемая приложением или документом Windows.

12. …данных.

15. Синоним слова “дорожка” (участка магнитного диска).

17. Конечное число точек на плоскости, соединенных отрезками кривых линий.

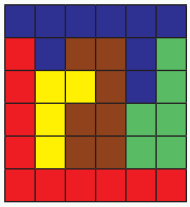
18. Устройство для соединения двух участков локальной сети.

19. Знак препинания.

20. Элемент языка разметки гипертекста.

Сложить квадрат ☺

Из пяти разноцветных фигур сложен квадрат 6 × 6. Используя эти же фигуры, получите квадрат 6 × 6 другим способом. Переворачивать фигуры обратной стороной вверх нельзя, а поворачивать можно.



Может сложиться впечатление, что решения нет. Однако оно есть!

Автор задачи — Игорь Акулич

Восемь вопросов и один термин

Ответив на восемь вопросов и вписав ответы в столбцы, в выделенном горизонтальном ряду прочтите слово — одно из основных понятий информатики.

1	2	3	4	5	6	7	8

1. Основной структурный элемент презентации, создаваемой с помощью программы Microsoft PowerPoint.

2. Устройство с органами управления, с помощью которых оператор воздействует на управляемые объекты.

3. Размер (высота) шрифта.

4. Последовательность букв и цифр, ограниченная с обоих концов пробелами, запятыми, точками, дефисами и т.п.

5. Программа, обладающая способностью к самовоспроизведению.

6. Совокупность характеристик символа или абзаца.

7. Непрерывная последовательность данных.

8. Результат арифметической операции.

Ответы (9 слов, найденное — с комментариями к нему) присылайте в редакцию.

Еще один “АМУР”ный числовой ребус

В нескольких последних выпусках “В мир информатики” были опубликованы числовые ребусы, в которых фигурировало слово-число **АМУР**. Решите, пожалуйста, еще один:

АМУР × **Р** = **М*****УАР**,

в котором звездочкой (“*”) может быть любая цифра.

ПОИСК ИНФОРМАЦИИ

Пять вопросов

1. Какая императрица учредила орден Святого Александра Невского?

2. Какой музыкальный инструмент слушают герои фильма “Старомодная комедия”?

3. Какой камень астрологи считают лучшим для получения тайной информации, так как он “способен” помочь сконцентрировать третий глаз? (Кроме того, мистики утверждают, что этот каменный талисман позволяет вступать в телепатический контакт.)

4. В романе “Утраченный символ” Дэн Браун описывает, как его героиня Кэтрин Соломон разрабатывает принципиально новое направление в науке, позволяющее не только взвесить человеческую душу, но и изменить существующий порядок вещей с помощью коллективных медитаций. Свои исследования она проводит в помещении с уникальным оборудованием. Какую конфигурацию напоминает ее лаборатория? Как называет Кэтрин свою лабораторию?

Циклоида

Помните оранжевые пластмассовые катафоты — светоотражатели, прикрепляющиеся к спицам велосипедного колеса? Прикрепим катафот к самому ободу колеса и проследим за его траекторией при движении велосипеда. Кривая линия, по которой будет перемещаться катафот, в математике называется “циклоида” (от греческого слова κυκλοειδής — *круглый*). Итак, циклоида — это траектория некоторой фиксированной точки окружности, катящейся без скольжения по прямой (см. *рис.* 1). Окружность (в случае велосипеда — колесо) называется “производящей окружностью”.

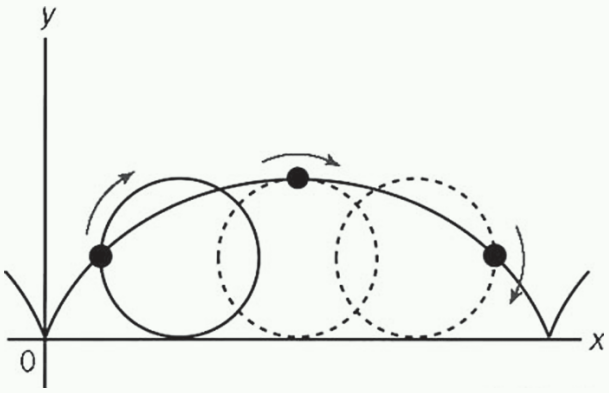


Рис. 1

У циклоиды масса любопытнейших свойств. Оказывается, например, что циклоида является кривой наибыстрейшего спуска. Иначе говоря, скатываясь по снежной горке, профиль которой выполнен в виде перевернутой циклоиды, мы окажемся у основания горки быстрее, чем в случае другой формы горки. Кроме того, циклоида является такой кривой, по которой должна двигаться тяжелая материальная точка, чтобы период ее колебаний не зависел от амплитуды колебаний. Используя это свойство, в XVII веке Христиан Гюйгенс (голландский механик, физик, математик, астроном и изобретатель) сконструировал часы, изображенные на *рис.* 2. Любопытно, что траектория конца маятника, как и ограничивающие его боковые “щеки”, представляет собой циклоиду.

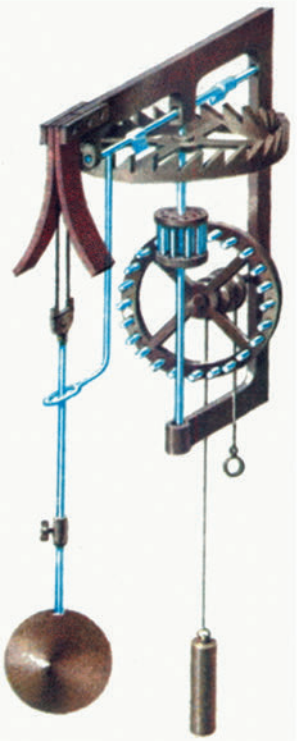


Рис. 2

Обращали ли вы внимание на то, куда полетит камень, застрявший в протекторе велосипеда или мотоцикла, когда выскочит из него, — против направления движения или по направлению? Как известно, свободное движение тела начинается по касательной к той траектории, по которой оно двигалось. Касательная к циклоиде всегда направлена по направлению движения и проходит через верхнюю точку производящей окружности. По направлению движения полетит и наш камушек. Наверняка многие знают, что будет, если кататься по лужам на велосипеде без заднего крыла. Мокрая полоска на спине этих читателей и была житейским подтверждением только что полученного результата.

Теперь перейдем к информатике ☺ и опишем методику получения изображения циклоиды на экране монитора компьютера.

Как и для других замечательных кривых (см. [1–3]), для решения задачи используем так называемые “параметрические уравнения” линии. Для циклоиды они такие:

$x = rt - r\sin(t)$;

$y = r - r\cos(t)$.

Напомним, что параметрическими такие уравнения называются потому, что они определяют значения координат x и y каждой точки кривой в зависимости от некоторого параметра, в нашем случае от параметра t. Кроме того, в уравнениях есть величина r — радиус производящей окружности.

Из *рис.* 1 видно, что циклоида состоит из одинаковых участков, то есть, как говорят в математике, “является периодической функцией по оси абсцисс”. Период равен $2\pi r$ (почему — подумайте самостоятельно). Давайте получим два участка кривой. Это значит, что при $r = 1$ значения параметра t должны меняться от 0 до 4π.

Сначала применим для построения циклоиды электронную таблицу (Microsoft Excel или подобную программу). С учетом только что сделанных рассуждений лист должен быть оформлен так (необходимые формулы на основе параметрических уравнений кривой запишите самостоятельно):

	A	B	C	…	S	T
1	t	x	y		r =	1
2	0	0,00000	0,00000			
3	0,05	0,00002	0,00125			
4	0,1	0,00017	0,00500			
5	0,15	0,00056	0,01123			
…						
251	12,45	12,56611	0,00676			
252	12,5	12,56632	0,00220			
253	12,55	12,56637	0,00013			

Рис. 3

По полученным расчетным данным можно построить график (тип диаграммы — **Точечная с гладкими кривыми**). При этом следует учесть, что на диаграммах и графиках электронных таблиц масштаб по осям x и y в общем случае не совпадает, в ре-

зультате чего вид полученной линии будет отличаться от показанного на *рис.* 1. Сделать масштаб одинаковым можно, меняя размеры области диаграммы.

Теперь решим задачу, разработав программу (как принято в нашем издании — на школьном алгоритмическом языке). Так как в этом языке (и в ряде других) величина — параметр цикла должна быть целого типа, то для расчета значений параметра t в диапазоне от 0 до 4π приходится "хитрить". Можно вместо вещественных значений использовать целые от 0 до 1256 ($4\pi \cdot 100$), а внутри тела цикла делить их на 100:

```
нц для i от 1 до 1256
  |Получаем соответствующее значение t
  t := i/100
  |Рассчитываем координаты
  |соответствующей точки кривой
  …
кц
```

Обсудим, как определить координаты точек на экране, чтобы получить два участка кривой, при этом использовав всю ширину экрана и верхнюю половину его высоты (см. *рис.* 4).

Рис. 4

Согласно параметрическим уравнениям, координаты линии равны:

```
x := int(r * t - r * sin(t))
y := int(r - r * cos(t))
```

На экране найденным значениям соответствуют координаты (для двух участков кривой и при $r = 1$):

```
x_экр := int(x * максX/12.56)
y_экр := y0 - int(y * максУ/2/(2 * r))
```

— где *int* — функция, возвращающая целую часть ее вещественного аргумента, *y*0 — координаты центра экрана по вертикали, *максX* и *максУ* — соответственно, максимальное значение координат x и y в выбранном режиме работы экрана. $2r$ — максимальное значение координаты y (в данном случае; почему — подумайте самостоятельно).

Вся программа:

```
алг Циклоида
нач вещ x, y, r, t, цел y0, x_экр, y_экр
r := 1
|Устанавливаем графический режим
…
|Координаты центра экрана по вертикали
y0 := int(максУ/2)
нц для i от 1 до 1256
  |Получаем соответствующее значение t
  t := i/100
  |Рассчитываем координаты
  |соответствующей точки кривой
  x := int(r * t - r * sin(t))
  y := int(r - r * cos(t))
  x_экр := int(x * максX/12.56)
  y_экр := y0 - int(y * максУ/2/(2 * r))
  |и изображаем эту точку
  точка(x_экр, y_экр)
кц
```

Задания для самостоятельной работы

1. Использовав электронную таблицу Microsoft Excel или подобную программу, получите два участка циклоиды для случая $r = 2$.

2. Разработайте программу (на языке программирования, который вы изучаете) для построения циклоиды для того же значения r.

Указание по выполнению. Получите два участка кривой, при этом использовав всю ширину экрана и верхнюю половину его высоты.

3. Представьте себе, что катится (см. *рис.* 1) не окружность, а круг. В этом случае можно рассматривать траекторию точки, расположенной не на окружности, а на некотором расстоянии a от центра круга. Параметрические уравнения соответствующей линии:

$$x = rt - a\sin(t);$$
$$y = r - a\cos(t),$$

— где r — радиус производящего (катящегося) круга, a — расстояние от центра круга до точки, траектория движения которой рассматривается.

Разработав программу или/и использовав электронную таблицу, получите три кривые для случаев:

1) $a < r$;
2) $a = r$;
3) $a > r$.

(Последнему варианту кривой соответствует траектория движения точки колес железнодорожного транспорта, трамваев и т.п., расположенной на так называемых "ребордах" — выступающих гребнях, не дающих вагону сойти с рельсов.)

Результаты (можно не по всем заданиям) присылайте в редакцию.

В заключение — историческая справка о циклоиде. Первыми из ученых на нее обратили внимание Николай Кузанский в XV веке и Шарль де Бовель в труде 1501 года. Но серьезное исследование этой кривой началось только в XVII веке.

Название *циклоида* придумал Галилео Галилей (во Франции эту кривую сначала называли *рулеттой*). Содержательное исследование циклоиды провел современник Галилея Мерсенн. Блез Паскаль, автор одной из первых вычислительных машин, писал о циклоиде: "Рулетта является линией столь обычной, что после прямой и окружности нет более часто встречающейся линии; она так часто вычерчивается перед глазами каждого, что надо удивляться тому, как не рассмотрели ее древние… ибо это не что иное, как путь, описываемый в воздухе гвоздем колеса".

Литература

1. О кардиоиде и о сердце. / "В мир информатики" № 203 ("Информатика" № 12/2014).
2. Еще две замечательные кривые. / "В мир информатики" № 204 ("Информатика" № 1/2015).
3. Трилистник и другие. / "В мир информатики" № 205 ("Информатика" № 2/2015).

Решаем задачу “Необычный маршрут”

В прошлом учебном году в разделе “В мир информатики” была опубликована задача “Необычный маршрут”. Напомним, что в ней речь шла о перемещении мальчика Мити по такому маршруту по плоской равнине: 1 м — на юг, 2 м — на запад, 4 м — на север, 6 м — на восток, 7 м — на юг, 8 м — на запад, 10 м — на север, 12 м — на восток, 13 м — на юг, 14 м — на запад, 16 м — на север, 18 м — на восток и т.д. Требовалось установить:

1) на каком расстоянии от точки *A* будет Митя после 100 “шагов” (после прохождения 100 отрезков);

3) какой общий путь пройдет Митя за 100 “шагов”.

Ответ на первый вопрос был обсужден в рубрике “Крепкий орешек” в ноябрьском выпуске журнала за 2014 год. Было показано, что можно установить, что за каждые четыре этапа маршрута мальчик удаляется от начала А на пять метров (использована теорема Пифагора). Следовательно, после 100 “шагов” Митя будет находиться на расстоянии 125 м от исходной точки.

Новая задача

Митя вышел из точки **A** плоской равнины и прошел 1 м — на юг, 2 м — на запад, 3 м — на север, 4 м — на восток, 5 м — на юг, 6 м — на запад, 7 м — на север, 8 м — на восток, 9 м — на юг, 10 м — на запад и т.д.

На каком расстоянии от точки **A** будет Митя после:

1) 100 “шагов” (после прохождения 100 отрезков);

2) 115 “шагов”;

3) *n* “шагов”?

Получите ответы, используя электронную таблицу или разработав компьютерную программу (для ответа на третий вопрос — только программу).

Составим программу, с помощью которой можно определить, на каком расстоянии от исходной точки **A** будет находиться Митя после некоторого *n*-го шага (после прохождения *n* отрезков).

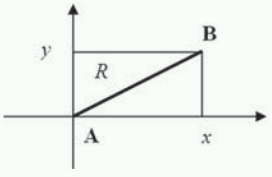
Достаточно очевидное решение такое. Если ввести прямоугольную систему координат с началом в точке **A** и после каждого шага рассчитывать координаты нового местоположения мальчика, то при известных координатах *x* и *y* точки **B**, в которой будет находиться Митя после *n*-го шага, расстояние *R* от начальной точки **A** можно рассчитать, используя теорему Пифагора: $R = \sqrt{x^2 + y^2}$ (см. рисунок ниже).

С учетом этого схема решения задачи будет следующей:

1. Ввод значения *n*;

2. Для каждого шага от 1-го до *n*-го:

2.1. Расчет длины отрезка пути на очередном шаге (назовем эту величину — *длина_шага*);



2.2. Определение координат *x* и *y* места нахождения Мити после очередного шага.

3. Расчет значения *R*.

4. Вывод ответа.

На этапе 2.2 координаты *x* и *y* после очередного шага можно рассчитывать, зная соответствующие значения координат до этого шага:

1) при перемещении на юг:

```
y := y - длина_шага
```

2) при перемещении на запад:

```
x := x - длина_шага
```

3) при перемещении на север:

```
y := y + длина_шага
```

4) при перемещении на восток:

```
x := x + длина_шага
```

А как рассчитать длину отрезка пути, проходимого на том или ином шаге? На первый взгляд кажется, что никакой закономерности в последовательности значений отрезков нет. Однако это не так. Если объединить длины отрезков каждой, так сказать, “четверки шагов”, то можно составить таблицу:

Номер “четверки шагов”	**Номер шага в данной “четверке шагов”**			
	1	*2*	*3*	*4*
1	1	2	4	6
2	7	8	10	12
3	13	14	16	18
…	…	…	…	…

Из нее видно, что в каждой строке, начиная со второй, значения в том или ином столбце на шесть больше, чем в том же столбце предыдущей строки. Учитывая это, можем утверждать, что для четверки шагов с неким номером, который обозначим *номер*_4, длина отрезка может быть определена по формуле:

$$длина1 + 6 \times (номер\_4 - 1),$$

— где *длина*1 — значение, равное 1, 2, 4 или 6 в зависимости от номера шага в данной четверке шагов.

Читатели, знакомые с понятием “прогрессия”, конечно же увидели в последнем выражении формулу для расчета члена арифметической прогрессии с номером *номер*_4 при разности, равной шести.

В программе фрагмент, относящийся к расчету длины отрезка пути на очередном шаге, может быть оформлен с использованием оператора выбора (варианта):

```
выбор
  при номер = 1:
    длина_отрезка := 1 + 6 * (номер_4 – 1)
  при номер = 2:
    длина_отрезка := 2 + 6 * (номер_4 – 1)
  при номер = 3:
    длина_отрезка := 4 + 6 * (номер_4 – 1)
  при номер = 4:
    длина_отрезка := 6 + 6 * (номер_4 – 1)
все
```

— где *номер* — номер шага, который имеет в "своей" четверке шагов шаг с общим порядковым номером *номер_шага*.

Значение последней величины может быть определено на основе анализа такой таблицы:

номер шага	Номер шага в данной "четверке шагов" (*номер*)
1	1
2	2
3	3
4	4
5	1
6	2
7	3
8	4
9	1
…	…

— из которой следует, что:

```
если остаток от деления значения
     номер_шага на 4 равен нулю
  то
     номер равен 4
  иначе
    номер равен указанному остатку
все
```

В программе на школьном алгоритмическом языке фрагмент для расчета значения *номер* имеет вид:

```
если mod(номер_шага, 4) = 0
  то
    номер := 4
  иначе
    номер := mod(номер_шага, 4)
все
```

— где `mod` — функция, возвращающая остаток от деления своего первого аргумента на второй. В других языках программирования для этого используется не функция, а специальная операция (как правило, знак этой операции также обозначается `mod`).

Менее очевидна формула для расчета номера четверки, в которую входит данный шаг. И здесь также составим таблицу для анализа:

номер шага	Номер "четверки шагов" (*номер_4*)
1	1
2	1
3	1
4	1
5	2
6	2
7	2
8	2
9	3
…	…

Из нее следует, что

```
номер_4  :=  div(номер_шага + 3, 4)
```

или

```
номер_4 : =  div(номер_шага –  1, 4) + 1
```

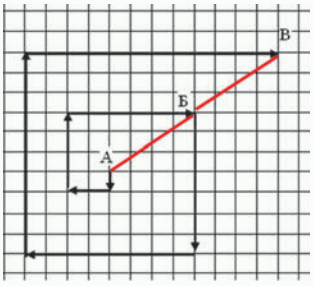
— где `div` — функция, определяющая целочисленное частное от деления своего первого аргумента на второй (в других языках программирования для этого также используется не функция, а специальная операция).

Предлагаем читателям самостоятельно убедиться в справедливости обоих вариантов.

Всю программу решения задачи (на языке программирования, который вы изучаете) соберите самостоятельно. С ее помощью определите, на каком расстоянии от исходной точки **A** будет находиться Митя после 50-го шага своего необычного маршрута.

Имеется также более рациональный (с точки зрения объема вычислений) вариант решения.

Если проанализировать первые четыре шага, то можно установить, что после них Митя будет находиться в точке **B**, от которой до точки **A** — 5 м (см. формулу для расчета *R* выше). После восьми шагов (в точке **B**) мальчик будет находиться от точки **A** на расстоянии 10 м.



Учитывая это, может быть предложена такая идея решения задачи.

Если общее число шагов *n* кратно четырем, то искомое расстояние рассчитывается следующим образом:

$$R = 5 \times \mathrm{div}(n, 4),$$

в противном случае нужно:

1) определить количество полных четверок шагов, сделанных Митей (оно равно div(*n*, 4));

2) найти координаты точки, в которой будет находиться мальчик после прохождения этого количества полных четверок шагов:

```
x :=  4 * div(n, 4)
y :=  3 * div(n, 4)
```

3) еще учесть один, два или три шага так, как это делалось в первом варианте.

Новый вариант программы на языке программирования, который вы изучаете, разработайте самостоятельно.

Результаты (можно не все) присылайте в редакцию.

Уважаемые коллеги!

Для поощрения самых активных участников конкурсов, проводимых в разделе "В мир информатики", редакция может направить вам электронный вариант диплома.

Заявку на диплом просьба прислать в редакцию электронной (адрес: vmi@1september.ru) или обычной почтой в мае–июне. Оформление дипломов будет проводиться в учебном заведении.

Общероссийский проект

Школа цифрового века

Издательский дом «ПЕРВОЕ СЕНТЯБРЯ» • digital.1september.ru

Каждый педагогический работник образовательной организации, вошедшей в проект «Школа цифрового века», получает доступ ко всем материалам проекта по принципу «все включено» (без дополнительной платы)

МАТЕРИАЛЫ ПРОЕКТА

- **24 предметно-методических журнала** по всем предметам и направлениям школьной жизни, включая журнал для родителей
- **Модульные дистанционные курсы*** из циклов «Навыки профессиональной и личной эффективности педагога» и «Инклюзивный подход в образовании» с выдачей сертификата
- **Дистанционные 36-часовые курсы**** повышения квалификации с выдачей удостоверения установленного образца
- **Методические брошюры** по всем школьным предметам

Стоимость участия образовательной организации в проекте – 6 тысяч рублей за весь учебный год. Стоимость участия не зависит от количества педагогических работников в образовательной организации

Участие образовательной организации и педагогических работников в проекте удостоверяется соответствующими документами: дипломом каждому педагогическому работнику, дипломом образовательному учреждению, дипломом руководителю образовательного учреждения. Для дошкольных организаций предусмотрен свой набор удостоверяющих документов

Срок действия проекта в 2015/16 учебном году: с 1 августа 2015 года по 30 июня 2016 года

Подробности и прием заявок
от образовательных организаций
на сайте

digital.1september.ru

---

* В течение указанного срока предоставляются без ограничения количества курсов на одного педагога.

** Предоставляется по одному курсу для одного педагога в течение одного учебного года (выбор конкретного курса – на усмотрение педагога).