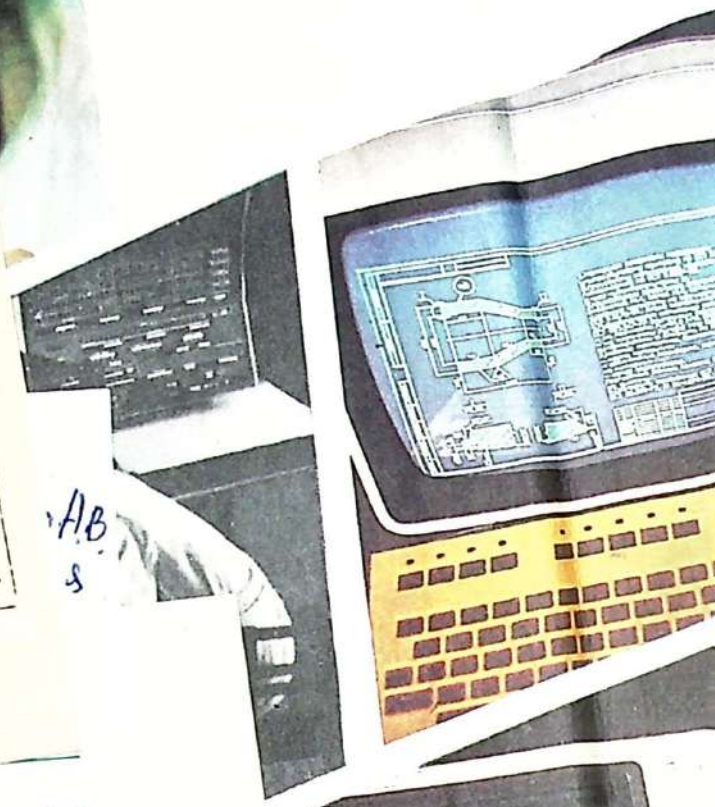
О профессиях, производстве и людях труда

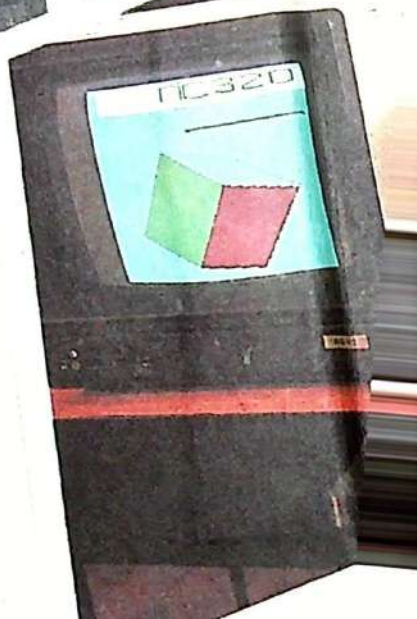
А.В. Нестеренко

ЭВМ И ПРОФЕССИЯ ПРОГРАММИСТА

возвращен
указанного здесь срока

Количество предыдущих выдач _____



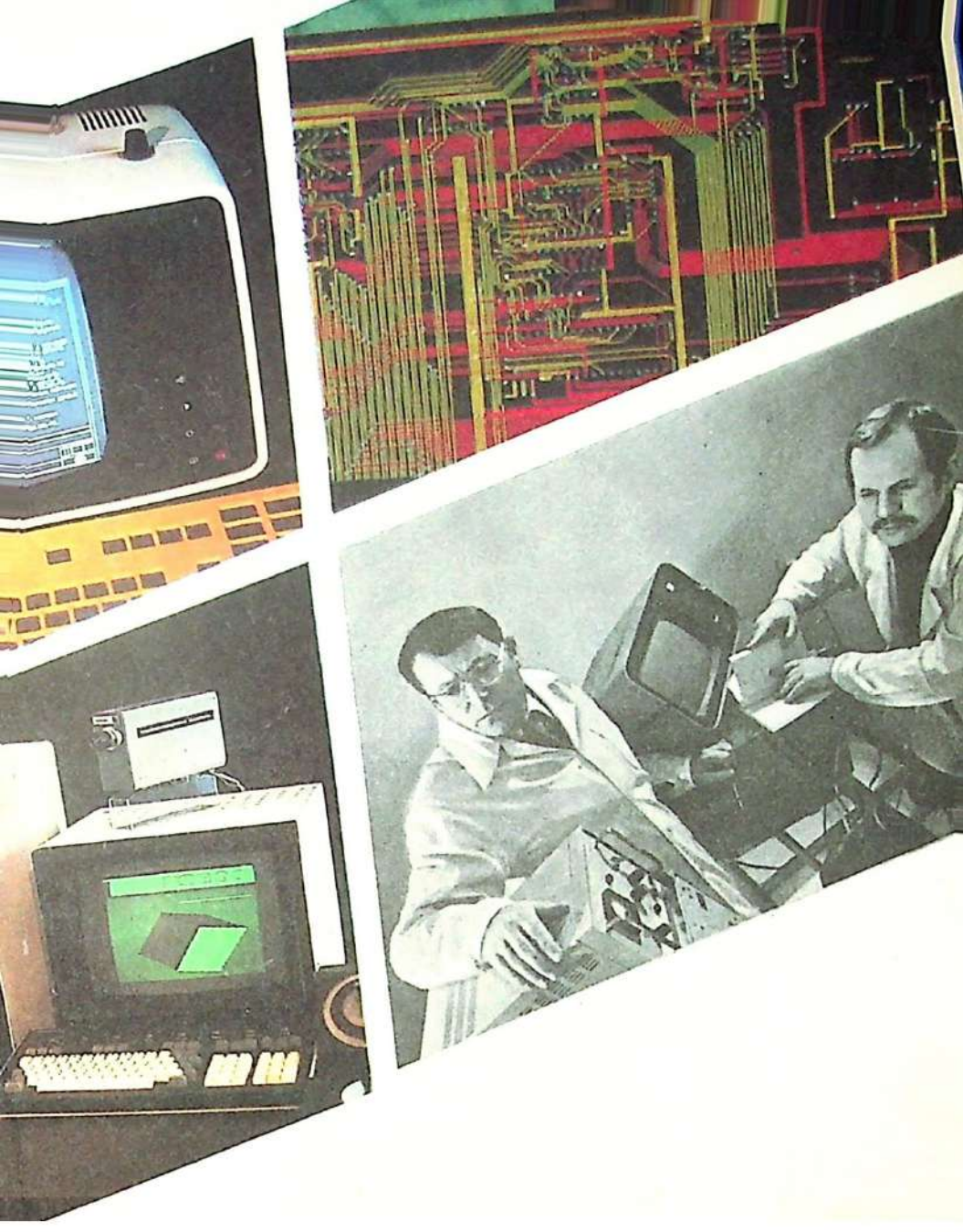


Т-4. Зак. № 2005

возвращен
указанного здесь срока

Количество предыдущих выдач	

Т-4. Зак. № 2005



профессиях, производстве и людях труда

А.В.Нестеренко

ЭВМ И ПРОФЕССИЯ ПРОГРАММИСТА

КНИГА ДЛЯ УЧАЩИХСЯ
СТАРШИХ КЛАССОВ
СРЕДНЕЙ ШКОЛЫ

МОСКВА
«ПРОСВЕЩЕНИЕ»
1990

Рецензенты:
заведующий кабинетом информатики
и вычислительной техники МГИУУ Н. Д. Угринович,
кандидат технических наук В. М. Савинков

Нестеренко А. В.

Н56 ЭВМ и профессия программиста: Кн. для учащихся ст. классов сред. шк.— М.: Просвещение, 1990.—160 с.: ил.— (О профессиях, производстве и людях труда).— ISBN 5-09-001322-5

В книге рассказывается о вычислительной технике и ее применении.

Читатель познакомится с профессией программиста, связанной с проектированием ЭВМ, созданием программных средств, обеспечением функционирования и развитием средств ВТ.

Книга адресуется учащимся старших классов, стоящим перед проблемой выбора профессии.

Н $\frac{4306020000-292}{103(03)-90}$ 276—90 ББК 32.973

ISBN 5-09-001322-5

ОГЛАВЛЕНИЕ

Тщетно надеяться написать книгу, которую можно дать молодому человеку со словами: «Прочти ее, и ты сможешь мыслить эффективно». Но когда люди пытаются понять, выявить и обойти непреодолимую сложность, я верю, что им можно помочь.

Э. Дейкстра

К ЧИТАТЕЛЮ

Ты уже знаешь таблицу умножения, знаком с законом Ома и молекулярным строением вещества. Сегодня в круг твоих знаний входят такие понятия, как *компьютер* и *информатика*. Ты прочтешь еще немало учебников и книг об идеях и открытиях и, став взрослым, будешь обладать большим запасом знаний об окружающем мире, знаний, накопленных человечеством за тысячелетия. Чем больше их объем, тем крупнее и надежнее то судно, на котором ты выйдешь в плавание по «Великому океану жизни». Куда направить корабль, кем быть — этот вопрос неизбежен. Ответом на него может быть совет: «использовать гибкость и выносливость молодости на поиски такой области деятельности, которая, обладая достаточным внутренним содержанием и достаточной реальной ценностью, обеспечит тебе возможность плодотворно работать в избранном направлении на протяжении всей жизни» (Н. Винер).

Одной из таких областей, безусловно, является вычислительная техника. Великие ученые, изобретатели проложили человечеству путь от первобытных орудий труда до вычислительных машин. Хотя электронно-вычислительная машина и проста в своей основе, но множество ее элементов и сложность их взаимосвязей затрудняют изучение ЭВМ. Для многих она так и остается «черным ящиком». И лишь самые смелые и умелые, рискнувшие заглянуть в этот «ящик», могут увидеть и познать ее «внутреннюю жизнь», подчиненную строгим законам логики и таких наук, как физика, химия, математика. Но пусть не огорчаются те, для кого она так и останется тайной. Работать с ЭВМ сможет человек любой профессии, если он овладеет знаниями и навыками общения с вычислительной машиной даже в минимальном объеме. Так же доступен и процесс составления программ для ЭВМ, который, как утверждают многоопытные программисты, не только дает экономические результаты, но и доставляет эстетические переживания, во многом близкие тем, которые испытывают при сочинении стихов или музыки.

Сфера применения вычислительной техники обширна и практически охватывает все области человеческой деятельности. Куда бы ни занесли твое судно ветра и течения жизни, всегда ты встретишь под

«созвездием Информатики» спасительный остров ЭВМ. Мир ЭВМ, сегодня еще полный тайн и чудес, завтра поможет тебе овладеть профессией, расширить круг твоих знаний, станет верным помощником в твоей трудовой деятельности.

Данная книга расскажет тебе о профессии программиста, без которой не обходится разработка ЭВМ, обеспечение ее функционирования, создание программных средств для различного применения.

Читать ее можно по-разному. Любознательным и усидчивым рекомендуем сначала ознакомиться с содержанием книги, а затем приступить к обстоятельному чтению. Те, кто всегда спешит, могут пропустить третью и четвертую главы. Но если ты выбрал для себя в будущем профессию программиста, то именно эти главы будут тебе особенно полезны. Третья и четвертая главы могут показаться трудными при первоначальном чтении, но не надо отчаиваться. К чтению этих страниц советуем вернуться немного позже. Если у тебя возникнет такое желание и ты попытаешься овладеть текстом снова, значит, можно надеяться, что ты захочешь стать программистом.

Отправляясь в путешествие по страницам книги, возьми в напутствие слова родоначальника вычислительной техники Блеза Паскаля: «Мы ничего не поймем, если будем читать слишком быстро или слишком медленно».

Мы живем еще очень рано,
На самой полоске зари.

Н. Асеев

ГЛАВА 1. ОТ КАМЕННОГО ТОПОРА ДО ЭВМ

Вся деятельность человека на протяжении многих веков была направлена на освоение природы, и началась она с присвоения ее готовых продуктов: корней, растений, животных, солнечного тепла. Однако со временем человек научился не только брать готовое у природы, но и воздействовать на нее же, чтобы получать еще больше продуктов. Люди стали возделывать землю, приручать и разводить животных, строить мастерские, фабрики, заводы, гидроэлектростанции, прокладывать железные дороги и космические трассы. В результате на Земле, некогда сплошь покрытой лесами и морями, возникли новообразования, которые русский ученый В. И. Вернадский назвал *ноосферой*.

Создавая ноосферу, человек одновременно использовал виды и свойства материи. Но на разных этапах этого процесса каждая категория материи осваивалась неравномерно. Вначале преимущество отдавалось освоению вещества, затем энергии и, наконец, *информации*. И в науке, т. е. при изучении природы и накоплении знаний о ней, тоже отмечаются периоды, которые оказываются связаны с развитием какого-то определенного вида материи. В результате такой неравномерности в ноосфере можно выделить три составляющие: *техносферу, эргосферу* и *инфосферу*, где образование техносферы связано с изучением вещества, эргосферы — с изучением энергии, инфосферы — информации.

1.1. ТРУД

Наиболее длительной оказалась деятельность человека по образованию техносферы. С этим этапом связано создание материальных благ, развите производства, машин и техники, формирование научного представления о единой материальной природе мира.

Истоки этого процесса берут начало в глубокой древности, когда первобытный человек взял в руки простое орудие — кусок кремня — и начал трудиться: колоть, резать, т. е. изменять объект труда, который к тому же надо было и перемещать, например к жилищу. Великий философ и мыслитель Ф. Бэкон сказал: «Голая рука и предоставленный самому себе разум не имеют большой силы. Дело совершается орудиями и вспоможениями, которые нужны не меньше разуму, чем руке».

Для того чтобы была возможность трудиться, человеку требовались определенная энергия и обдумывание производимых действий. И таким образом в процессе развития трудовой деятельности человека определились 5 функций труда: транспортная, технологическая, энергетическая, контрольно-управляющая и логическая. Если ты посмотришь на таблицу 1, то увидишь взаимосвязь этих функций на примере изготовления какой-либо детали.

Таблица 1

Функция труда человека			Содержание	Пример
Физическая	Обеспечивающая	Транспортная	Перемещение объекта труда	Грузчик
		Технологическая	Изменение объекта труда	Кузнец
	Рабочая	Энергетическая	Обеспечение других функций энергией	Замах руки с молотом
Умственная		Контрольно-управляющая	Контроль и направление труда	Определение места удара молотом по заготовке
		Логическая	Создание идеи, мысли, образа	Формирование представления о будущей детали

Но вскоре человеку потребовалась такая работа, выполнять которую вручную ему стало трудно и даже невозможно. Например, при строительстве ритуального сооружения или жилища необходимо было поднимать и перемещать тяжести, а для изготовления каких-либо орудий — сверлить отверстия путем быстрых вращений. Эти действия не согласовывались с естественными возможностями человека, а применение одних орудий труда оказывалось недостаточным; начиналось изобретение машин, которые более эффективно заменяли человека в процессе труда. При этом было необходимо дополнительно обеспечивать производство освещением, вентиляцией, звукопоглощением и многим другим, что вместе с орудиями труда и машинами образовало *средства труда,* которые и составляют сегодня понятие *техника.*

Процесс труда, воздействуя на объект труда, направлен на изменение физических и химических свойств этого объекта и преобразует его энергию и вещество, что влечет за собой изменение размеров, объема, массы и др. Таким образом, ты видишь, что физика, химия и математика — это те «три кита», на которые опирается процесс освоения человеком природы.

1.2. МАШИНЫ

Итак, человеку потребовалось изобрести машины, развитие которых пошло по пути создания вспоможений для выполнения отдельных функций труда. Постепенно машины сформировались в отдельные группы: транспортные, технологические, энергетические и управляю-

щие. Назначение первых трех групп соответствует функциям таблицы 1, а управляющая группа машин облегчает труд человека при выполнении контрольно-управляющих и логических функций.

Существующие группы машин объединяются в единые агрегаты, выполняющие сразу несколько функций; например, локомотив обеспечивает транспортную и энергетическую, металлообрабатывающий станок — технологическую и энергетическую, а промышленный *робот* выполняет все 5 функций: *энергетическую* (полученная электрическая энергия преобразуется в механическую), *транспортную* (робот сам закладывает заготовки, а затем снимает и складирует готовые детали), *технологическую* (обрабатывает заготовки), *контрольно-управляющую* и *логическую,* т. е. при изменении свойств материала какой-либо заготовки робот определяет наивыгоднейшие условия обработки, и все это он делает по заданному шаблону или чертежу. Две последние функции выполняются с помощью *вычислительной машины,* которая может быть встроена в него или связана с ним извне. Являясь управляющим устройством, вычислительная машина обрабатывает поступающую информацию, определяющую выполнение указанных функций. Несмотря на то, что робот выполняет все 5 функций труда, он, конечно, не заменяет полностью человека. Если поступит информация, не предусмотренная *программой,* которая закладывается в вычислительную машину, то робот в лучшем случае остановится «в глубокой задумчивости», из которой его может вывести только человек.

Итак, **вычислительная машина!** Призванная выполнять важнейшие функции труда при освоении природы, она сама стала продуктом этого освоения. Своим рождением вычислительная машина обязана физике, химии и математике. Как же переплелась судьба ее изобретения с развитием этих наук? Об этом мы с тобой сейчас и поговорим.

1.3. НАУКА

Начнем с химии. Из школьного курса ты знаешь, что ей принадлежит освоение вещества и использование его свойств для создания материальных благ. «Чего довольно самой природе, человеку мало!» — это высказывание Сенеки может служить девизом химической технологии, позволяющей получать то, что природа не дает нам в готовом виде. И прежде всего это относится к металлам и их сплавам. Добывание огня и наблюдения за процессом горения привели еще 4 тыс. лет назад к производству железа из руд. С открытием в XVIII в. основных законов химии (например, сохранение массы вещества), а также атомно-молекулярной теории нашим соотечественником М. В. Ломоносовым, французом Лавуазье, англичанином Дальтоном химическая технология стала стремительно развиваться.

Новый этап в развитии химии начался с достижений физики, связанных с появлением полупроводников в первой половине XX в. В начале 60-х гг. была разработана технология производства интегральных микросхем, объединяющих в одном миниатюрном кристалле тысячи

полупроводниковых приборов. Интегральная технология основывается на достижениях фотохимии, фотолитографии, электрохимии; использует процессы выращивания кристаллов. Интегральные микросхемы (как ты уже, наверное, знаешь) широко применяются в современной вычислительной технике.

Не закончился еще и «железный век». Изучение оксидов металлов привело к появлению ферритов, сочетающих магнитные свойства вещества с электрической полупроводимостью, что тоже нашло применение в вычислительной технике. На очереди освоение еще более эффективной основы элементов этих машин — магнитоодноосных кристаллов и пленок.

Этот перечень достижений науки в освоении вещества можно было бы продолжить. Но обратимся пока к физике, так как именно эта наука «дала энергию» вычислительной машине.

Широкое использование энергии началось в XVIII в. с создания первой паровой машины. Но надо отметить, что еще в первобытном периоде человек умел получать и использовать огонь, что ему пригодилось в дальнейшем при получении металла в рудоплавильной печи. Но и этого оказалось мало; стали необходимы воздуходувные мехи, использующие механическую энергию. Затем человек все чаще обращается к энергии ветра и воды: появляются мельничные ветряные двигатели и приводы от водяного колеса.

Поворотным пунктом в использовании энергии стало открытие электрического тока и электромагнетизма (XVIII–XIX вв.). Затем последовало открытие атома, что привело к развитию электронной теории, на основании которой английский ученый Д. Флеминг в 1904 г. изобрел двухэлектродную электронную лампу (диод), а чуть позже (1906 г.) американский ученый Ли де Форест создал триод — лампу, где потоком электронов можно управлять с помощью третьего электрода — сетки.

Успехи физики и электротехники дали новый толчок в развитии средств связи. Появление возможности использовать для передачи *информации* электрические сигналы привело к открытию новых направлений: радио, телеграфии, телефонии, а затем и телевидения. Здесь неоценимый вклад внесли русские ученые и инженеры, как, например, П. Шиллинг, А. С. Попов, П. М. Голубицкий, Б. Л. Розинг, М. А. Бонч-Бруевич. Знания законов физики и опыты по разработке технических средств связи явились важным фактором создания ЭВМ. Теперь нам остается рассмотреть, какова роль математики в изобретении электронно-вычислительной машины.

Математика — это наука, корни которой идут из глубины веков. Как только люди стали общаться между собой, они стали *считать*. Самыми первыми инструментами счета, не потерявшими своей ценности и теперь, были пальцы на руках, а также палочки и камешки. Слово «камешки» на латинском языке читается как *calculi,* а их перебрасывание при счете — *calculare,* что означает «считать».

Затем расчеты (например, расчет с покупателем) стали фиксировать на глиняных табличках, бересте, папирусе, что привело к появлению

систем счисления. А в дальнейшем это предопределило разработку *правил оперирования числами.* Еще в III в. до н. э. математик Евклид, автор математического трактата «Начала», в геометрической форме изложил правило получения наибольшего общего делителя двух натуральных чисел. Древнегреческий ученый Эратосфен (III—II вв. до н. э.) предложил способ получения простых чисел. *Правила четырех арифметических действий над числами* разработал в IX в. узбекский математик Ал-Хорезми. В Европе эти правила стали называть алгоритмами (от латинской формы написания имени автора — *Algorithmi).* Хотя в дальнейшем многие стали пользоваться этим понятием, но точного его определения долгое время не существовало. И только в начале XX в. алгоритм стал объектом математического изучения. Благодаря работам английского математика А. М. Тьюринга, американца Е. Поста, советских ученых А. А. Маркова и А. М. Колмогорова понятие алгоритма стало базовым понятием вычислительной техники.

Но этим вклад математики в создание ЭВМ не ограничился. С середины XX в. стали разрабатываться различные способы описания алгоритмов, например с помощью специальных *языков,* которые называются *алгоритмическими, и графовых схем* — графического изображения алгоритма. Как ты знаешь, построение алгоритма подчинено определенным законам *логики,* которая, являясь наукой о построении правильных умозаключений, имеет многовековую историю. Начала логики изложены в работах Аристотеля, и в таком виде она развивалась как одно из направлений философии. Существенного практического применения логика тогда не имела. Но попытка английского математика Дж. Буля в XIX столетии изобразить ее в виде алгебраической системы и изучать теми же методами, как и другие разделы математики, оказалась полезной на практике. В результате сформировалось новое математическое направление — *алгебра логики,* ставшее фундаментальной основой вычислительной техники.

Надо сказать, что одновременно, в 70-х гг. XIX в., немецким математиком Г. Кантором проводились исследования, относящиеся к сравнению бесконечных чисел по величине. Для решения этой проблемы Кантор выдвинул ряд идей, которые привели к созданию самостоятельной математической дисциплины — *теории множеств.* С развитием этой теории объектом алгебры логики стали функции и различные *операции* над ними. Практическое значение для вычислительной техники приобрел определенный класс функций, у которых значения равны всего двум величинам — 0 и 1, что соответствует двум логическим понятиям — «ложь» и «истина». Впоследствии это направление алгебры логики получило название *булевой алгебры.*

Широкое применение алгебра логики нашла и в *абстрактной теории автоматов,* значительный вклад в которую внесли советские ученые В. М. Глушков и А. А. Летичевский, и именно она стала основой для теории проектирования ЭВМ и *теории программирования.*

Таким образом, дорогой читатель, ты, вероятно, понял, что без физики, химии и математики изучение основ вычислительной техники практически невозможно.

1.4. ИНФОРМАЦИЯ

Накопление человечеством опыта и знаний при освоении природы совмещалось с освоением информации. Именно этот процесс привел к образованию инфосферы.

Информация в переводе с латинского языка означает: разъяснение, изложение чего-либо или сведения о чем-либо. Такое понятие, как *обработка информации*, появилось совсем недавно, но обрабатывать информацию люди начали еще в древние времена.

Сначала из поколения в поколение информация передавалась *устно*. Это были сведения о профессиональных навыках, например о приемах охоты, обработки охотничьих трофеев, способах земледелия и др. Но затем информацию стали фиксировать в виде *графических образов* окружающего мира. Так, первые наскальные рисунки, изображающие животных, растения, людей, как известно, появились примерно 20—30 тыс. лет назад.

Начатый поиск более современных способов фиксирования информации привел к появлению *письменности*. Вначале люди записывали расчеты с покупателями, а затем написали и первое слово. На чем только они не писали! В Индии — на пальмовых листьях, в Вавилоне — на глиняных плитках, на Руси пользовались берестой. Как видим, *письменность* — это новый шаг человечества в области хранения и передачи информации. Однако *первым революционным явлением* в этой сфере стало изобретение печатного станка, благодаря которому появилась книга и, таким образом, стало возможно массовое тиражирование профессиональных знаний, зафиксированных на материальном носителе.

Сегодня потоки книг, сливаясь с потоками технической документации и многотомной справочной литературы, образуют океаны информации. Эту информацию необходимо *хранить* и *передавать* потребителю, для чего нужен мобильный и емкий *носитель*.

Но книга является неудобным, сложным, дорогим, а главное, «медленным» носителем информации. Вся многогранность содержания раскрывается человеку при перелистывании, чтении и рассматривании книги. Таким образом, она не может непосредственно влиять на производственный процесс. Сначала человеку необходимо найти нужную ему книгу, освоить накопленные в ней знания, которые позже смогут дать толчок дальнейшему развитию производства. Хранение книг требует громадных зданий и специальных климатических условий, а их доставка потребителю сопряжена с дорогостоящим размножением во множестве экземпляров и объемными транспортными перевозками. Книга как носитель информации сегодня уже отстает от стремительного продвижения человечества по пути освоения природы. Прогресс в этой деятельности, обусловленный в первую очередь развитием ком-

муникаций, т. е. связью между людьми, требует расширения влияния инфосферы на техносферу и эргосферу.

Этому требованию отвечает *второе революционное изобретение* XX в.— электронная вычислительная машина (ЭВМ). Она-то и является носителем информации и средством доставки ее потребителю. В совокупности с линиями связи, такими, как проводная, радио-, космическая и оптическая, ЭВМ делает человеку доступной и мобильной любую часть гигантского объема информации, которая без непосредственного воздействия на человека сможет влиять на работу производственного оборудования, например на станки с программным управлением, роботы, действующие уже сейчас. Вступают в строй автоматизированные линии, целые автоматизированные производства.

Отсюда, конечно, не следует, что в будущем ЭВМ вытеснит из обихода книгу. Можно ли представить нашу жизнь без томика стихов или прозы? Ведь книга не просто «носитель информации», она — часть нашего духовного мира. Уже сейчас, передавая информацию в машинную память, люди освобождают полки книжных хранилищ от технической документации и справочной литературы. В будущем, по словам В. М. Глушкова, «... электронные вычислительные машины будут кладовыми не только технических и научных знаний человечества, но и всего, что было создано им за многие века своего существования; они станут огромной и вечной памятью его».

Хотя *информационный взрыв* и прозвучал в первой половине XX в., однако эффективного устройства обработки информации все еще не было.

1.5. КИБЕРНЕТИКА И ЭВМ

Отважным мореплавателям и мудрым астрономам требовалось быстрое проведение точных расчетов, связанных с логарифмами и тригонометрическими функциями. Проведение таких операций без помощи специальных устройств уже в XVII в. оказывалось им не под силу. Правда, еще в 2600 г. до н. э. применялось счетное устройство, называемое *абак*. Оно представляло собой ящик со струнами, по которому передвигались кости. Однако подобное устройство оказывалось несовершенным для задач, которые становились все сложнее. Прошло несколько тысяч лет, прежде чем были построены новые счетные устройства. Такая длительная пауза была вызвана повсеместным применением римской системы счисления. И только принятие арабской системы послужило толчком к разработке усовершенствованных счетных устройств.

В период между 1614 и 1617 гг. шотландский математик Джон Непер разработал метод упрощенного умножения и деления, основанный на применении счетных пластинок и таблиц прямого умножения. Непер создал механическое устройство, содержащее костяные пластинки с отпечатанными на них цифрами. В определенном сочетании эти пластинки позволяли выполнять прямое умножение. Настоящий

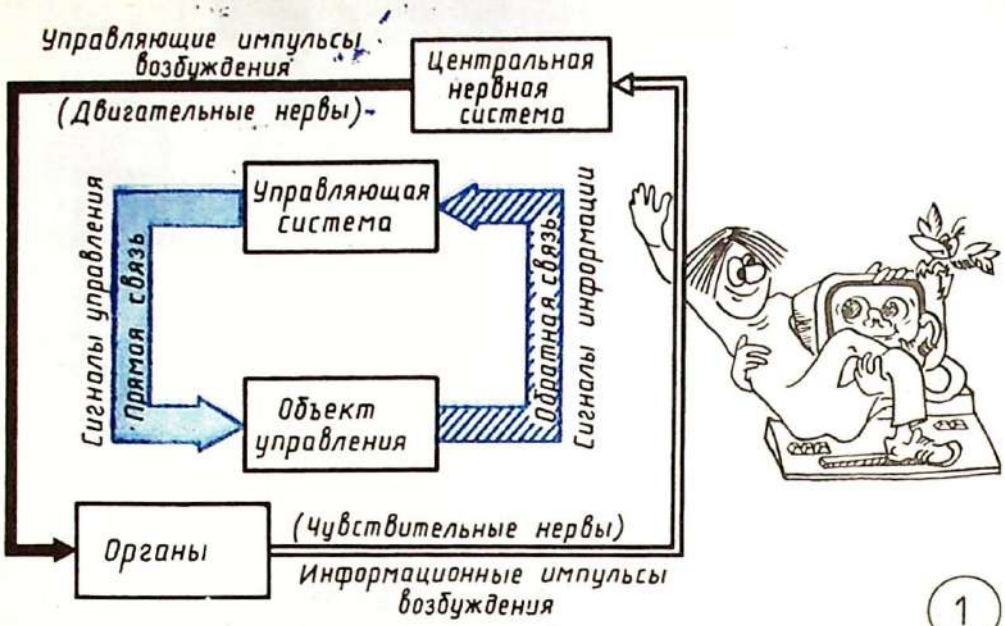
переворот произвело изобретение юного нормандца Блеза Паскаля. Он — один из самых знаменитых людей в истории человечества, в которую вошел как великий физик и математик. Имя Паскаля связано и с зарождением вычислительной техники. В конце 1640 г. он задумал построить машину, чтобы освободиться от расчетов «с помощью пера и жетонов». Основной замысел, как писал сам Паскаль, заключался в следующем: «...каждое колесо или стержень некоторого разряда, совершая движение на десять арифметических цифр, заставляет двигаться следующее на одну цифру». Реализация этой идеи потребовала пяти лет напряженной работы автора, приведшей к созданию «паскалева колеса», как говорили современники, производившего четыре действия над пятизначными числами. Затем производство таких машин было налажено, и до настоящего времени их сохранилось восемь штук. Но, по словам Паскаля, «вещи всегда лучше, когда они совсем еще новые».

В 1694 г. Лейбниц разработал машину, выполнявшую все известные в то время математические операции — четыре основных арифметических действия и извлечение квадратного корня.

И наконец в 1878 г. шведский инженер В. Т. Однер разработал усовершенствованное устройство — арифмометр, который лег в основу многих других счетных устройств. Одно из них, имеющее имя «Феликс», можно и сегодня встретить в учреждениях.

Промышленная революция XIX в., связанная с резким увеличением экономических расчетов и стремительным развитием точных наук, требовала построения надежных быстродействующих счетных машин, способных выполнять действия с большими числами, особенно при изучении и проектировании каких-либо *сложных систем*. Вообще системы окружающего нас мира состоят из большого числа взаимосвязанных элементов, между которыми циркулирует информация. А сложные системы содержат большое число таких элементов и значительные объемы информации, циркулирующей между ними.

Советский математик Г. Н. Поваров делит все системы на 4 группы в зависимости от числа элементов: малые — $10...10^3$ элементов, сложные — $10^4...10^7$, ультрасложные — $10^7...10^{30}$ и суперсистемы — $10^{30}...10^{200}$ элементов. В качестве примера сложной системы (системы второй группы) он приводит автоматическую телефонную станцию, третьей — организм высших животных и человека, четвертой — Вселенную. Освоение сложных систем шло как в науке, так и в производстве. В науке все больший интерес проявлялся к микромиру — элементарному строению вещества и к мегамиру — изучению законов Вселенной, где ученые столкнулись со сверхмалыми и сверхбольшими числами, обрабатывать которые традиционными методами стало весьма затруднительно. Да и развитие производства выдвигало все новые требования к машинам и механизмам. Устаревающая техническая система со всем набором функций включалась конструкторами в новую как подсистема. Шло время, и бывшая «новая» система становилась подсистемой следующей, более совершенной, но и более сложной. Так росло число элементов систем и, соответственно, число связей между ними.



1

Несмотря на различие всевозможных систем как в живом мире, так и в технике, в них есть много общего, позволяющего для их исследования применять одни и те же *модели*. Одну из таких моделей составляют управляющая система и объект управления (рис. 1), объединенные *каналами прямой и обратной связи,* по которым передаются управляющая информация и ответная на нее. Впервые на эту общность систем обратил внимание американский математик Норберт Винер. Это был ученый с широким кругозором и ярко выраженным влечением к новому. Придя в математику из философии, он занимался также физикой, биологией, медициной, думал о связи и содружестве разных дисциплин. Все это помогло ему увидеть скрытое родство между рядом важных областей и наметить проект их объединения в рамках новой общей науки, которую в 1947 г. американский ученый назвал *кибернетикой*. В 1948 г. в Париже и одновременно в Кембридже вышла книга Винера «Кибернетика, или Управление и связь в животном и машине» (*«Cybernetics or control and communication in the animal and machine»*). С этого времени и идет отсчет эры кибернетики. В книге он пишет: «Было решено назвать всю теорию управления и связи в машинах и живых организмах кибернетикой, от греческого *æυβερνητης* — «кормчий»... Мы хотели также отметить, что судовые рулевые машины действительно были одними из самых первых хорошо разработанных устройств с обратной связью». Справедливости ради надо отметить, что выбирая термин «кибернетика», Винер пытался создать неологизм, не подозревая, наверное, что этот термин применяли еще Платон и Ампер. И это, пожалуй, не случайное совпадение, а иллюстрация длительного, но верного пути к истине.

Когда эта книга вышла в свет, первая ЭВМ уже была создана. Но еще в 1940 г., размышляя над теорией связи и проблемой повышения

скорости вычислений, Винер предопределил основу дальнейшего развития вычислительных машин. Вот как он описывает этот период: «Исходя из своих общих идей, я рассматривал автоматические вычислительные машины также как одну из форм систем связи, поскольку основное внимание здесь уделяется передаче сигналов, а не мощностей. С моей точки зрения, такие системы представляют собой просто последовательность переключающих устройств, соединенных между собой таким образом, что информация, содержащаяся в сигнале на выходе каких-то из них, используется в последующих устройствах в качестве входного сигнала и сигнала управления.

Ясно, что такие переключающие устройства могли реализовываться и в форме зубчатых колес или других подобных механизмов, и в форме механических или электрических реле, использующих электронные лампы или другие средства электроники... Мне казалось, что наиболее удобными будут переключающие устройства, которые осуществляют выбор между двумя, а не десятью возможностями...»

Действительно, к 1940 г. сошлись, с одной стороны, потребности в скоростном вычислителе, с другой — возможности его реализации, основанные на достижениях связи, электроники, математики.

В начале 40-х гг. американский инженер болгарского происхождения Д. Атанасов, а затем американский ученый Д. Моучли предложили применение радиоламп для выполнения вычислений, а в 1946 г. первая электронная вычислительная машина начала действовать.

Посмотри рисунок на цветной вклейке I — на нем выделены некоторые результаты освоения наукой соответствующих категорий материи, ставших основой создания ЭВМ. А вычислительная машина, в свою очередь, становится неотъемлемым звеном всех видов машин, определяя сегодняшний день техники. Замыкая на нашей кибернетической модели обратную связь техники с наукой, можно только догадываться о тех гигантских свершениях, которые подарит нам наука, вооруженная современной техникой. И двигателем этого прогресса будет ЭВМ.

1.6. ПЕРСОНАЛЬНЫЕ ЭВМ И СЕТИ

Прогресс в человеческой деятельности таков, что и большой ЭВМ становится не под силу перерабатывать быстро, как говорят, оперативно, ту информацию, которая циркулирует даже в небольшой системе управления. Естественно, на помощь привлекаются дополнительные ЭВМ, а чтобы повысить оперативность обработки и создать максимальные удобства для работы, эти машины устанавливаются непосредственно на рабочем месте. Так родились *профессиональные* персональные ЭВМ. Множество подобных рабочих мест, оснащенных ПЭВМ, связаны между собой и с главной большой ЭВМ, образуя единую сеть. Такая сеть, расположенная в рамках одного учреждения или предприятия, получила название *локальная вычислительная сеть* (ЛВС). Два родственных предприятия, находясь на большом расстоянии друг

от друга, могут, используя линии связи, объединить свои ЛВС в территориально-распределенную сеть ЭВМ. И наконец, ЭВМ, находящиеся в разных городах страны и даже в разных странах, связываются в глобальные сети по кабельным, радио- и космическим каналам.

Что дает сеть ЭВМ? Прежде всего это возможность индивидуального доступа к информации, хранящейся в различных уголках города или страны, причем практически мгновенного доступа в рамках ЛВС и с малыми задержками в крупных сетях. Кроме того, значительно ускоряется доставка корреспонденции — писем, телеграмм, если используется электронная почта. Фактически с появлением персональных ЭВМ и их сетей миллионам людей предоставляется возможность иметь в индивидуальном пользовании «станок для обработки информации».

1.7. СИСТЕМЫ УПРАВЛЕНИЯ

Вслед за учебниками и играми наступает удивительная и главная пора жизни, называемая простым словом *работа*. Труд человека, с описания которого мы начали путешествие по страницам этой книги, неразрывно связан с этим словом. Но куда бы ни привело тебя призвание — на завод, в институт, в конструкторское бюро, в медицину или просвещение — повсюду надежным помощником и другом тебе будут ЭВМ — большие и малые, микроЭВМ и те, которые будут еще меньше. На рисунке цветной вклейки II,*а* (серые прямоугольники) отмечены основные сферы человеческой деятельности, где уже сегодня применяются персональные ЭВМ. Эти сферы отражают этапы создания материальных благ.

Впереди всегда идет наука с множеством научно-технических расчетов и экспериментов.

После того как *идея* «созрела» и ее обосновали возможностью и необходимостью практической реализации, приступают к проектированию будущего изделия. Когда же готовы чертежи и документация, можно приступать к производству изделия. Заготовка превращается в готовое изделие в ходе технологического процесса, который осуществляется на предприятии — фабрике или заводе. Станками и машинами, участвующими в этих процессах, управляют люди. Их необходимо всему этому обучать. И начинается такое обучение в школе, а затем продолжается в училищах, техникумах и институтах. *Обучение* — это передача знаний от предков к потомкам. Основным носителем знаний является текстовая информация — множество книг, журналов, брошюр и др. Часть такой информации сейчас переносится в магнитную память ЭВМ. На всем пути, начиная с зарождения идеи и кончая выпуском работающего образца изделия, основную роль играет человек. С каждым годом все более усложняется труд человека, поэтому и появились различные приспособления, машины и механизмы. Сегодня «информационный бум», требующий громадного объема манипуляций с числовыми и текстовыми данными, определил создание *автоматизированных*

систем управления. Слово «автоматизированная» означает, что в состав системы входит ЭВМ, но основным «элементом» такой системы по-прежнему является человек.

С помощью таких систем человек непосредственно участвует в процессе управления производством. Но людям всегда хотелось найти себе механических помощников, которые имели бы мускулы, глаза, уши и даже мозг для того, чтобы полностью переложить на их «плечи» тяжелую, монотонную работу и работу в опасных условиях. Создание таких устройств велось долго и безуспешно. И только появление особых устройств — *микропроцессоров* сделало эту мечту явью. Воплотилась она в роботах. *Робот* — это не только устройство, способное лишь хватать, толкать или катить, но им может быть и стиральная машина с микропроцессорным управлением. Главное в том, что эти устройства обладают элементами «интеллекта» за счет программного управления, реализуемого микропроцессором. Изменение программы, заложенной в устройство, может в корне изменить его поведение. Возможность гибкой передачи механизму логики действий, сначала простых, а затем более сложных, является величайшим открытием современности.

На рисунке цветной вклейки II,*а* (желтые прямоугольники) приведены основные направления применения микропроцессоров. Прежде всего это промышленная сфера. Активно внедряются микропроцессоры на транспорте и в связи. Не оставлена без внимания и бытовая техника, где за счет программного управления значительно расширяются ее возможности.

Чтобы понять, какое место занимает ЭВМ в названных системах управления, надо знать возможности вычислительной машины, правила обращения с нею как с инструментом обработки информации. Путь к этому пониманию лежит только через знания устройства ЭВМ, а также принципов программирования. Давай потратим несколько часов на чтение следующих страниц и попробуем разобраться в этих вопросах. Эти страницы — не учебник по программированию или электронике. Они содержат лишь те сведения, с которыми проще приступать к чтению настоящих учебных пособий.

4 — для всех 4?
Все ли семерки похожи?

. . .

Читает ли бабочка книгу
своих порхающих крыльев?
Какой алфавит изучила
пчела для чтения карт?
Какими числами множит
муравей погибших собратьев?

Пабло Неруда

ГЛАВА 2. ЭВМ: АРИФМЕТИКА И АЛГЕБРА

2.1. КАК СЧИТАЕТ ЭВМ?

2.1.1. Прежде всего — позиция...

Системы счисления делятся на позиционные и непозиционные.

Древние египтяне применяли систему счисления, состоящую из набора символов, изображавших распространенные предметы быта. Совокупность этих символов представляла число. Расположение их в числе не имело значения, отсюда и появилось название *непозиционная система*. К таким системам относится и римская, в которой впервые все величины представлялись с помощью прямолинейных отрезков. Людям приходилось либо рисовать громоздкие строки повторяющихся символов, либо увеличивать алфавит этих символов. Это и явилось общим недостатком непозиционных систем счисления. В римской системе для записи больших чисел над символами основного алфавита ставилась черточка, которая обозначала: число умножается на 1000. Но все эти «маленькие хитрости» были бы бессильны перед проблемой записи очень больших чисел, с которыми сегодня проходится иметь дело вычислительным машинам.

Выход из этого положения был найден, как только стали применять *позиционные системы*. В такой системе счисления число представляется в виде определенной последовательности нескольких цифр. Место каждой цифры в числе называется *позицией*. Первая известная нам система, основанная на позиционном принципе,— шестидесятичная вавилонская. Цифры в ней были двух видов, одним из которых обозначались единицы, другим — десятки. При определении числа учитывали, что цифры в каждом следующем *разряде* были в 60 раз больше той же самой цифры из предыдущего разряда. Запись числа была не однозначной, так как не было цифры для определения нуля. Следы вавилонской системы сохранились и до наших дней в способах измерения и записи величин углов и времени.

Однако наибольшую ценность для нас имеет индо-арабская система, где имеется ограниченное число значащих цифр — всего 9,

а также символ 0 (нуль). Индийцы первыми использовали нуль для указания позиционной значимости величины в строке цифр. Эта *система* получила название *десятичной,* так как в ней было десять цифр.

В эпоху вычислительной техники получили практическое применение восьмеричная, шестнадцатеричная и двоичная системы счисления, которые являются ее основой.

Итак, позиционная система! В ней каждой позиции присваивается определенный *вес* b_i, где b — *основание* системы счисления.

Например, четырехпозиционное число можно представить следующим образом:

$$D = d_3b_3 + d_2b_2 + d_1b_1 + d_0b_0,$$

где d_i соответствует цифре.

Вес b_i увеличивается от позиции к позиции справа налево пропорционально. В качестве такой пропорции выступает *степень* основания. Таким образом, веса в позиционной системе приобретают вид b^i, ..., b^2, b^1, b^0. Вышеприведенный пример тогда примет вид:

$$D = d_3b^3 + d_2b^2 + d_1b^1 + d_0b^0.$$

Если d_i есть множество десятичных цифр, а основание $b = 10$, то значение числа D вычислится так:

$$D = 5 \cdot 10^3 + 4 \cdot 10^2 + 8 \cdot 10^1 + 3 \cdot 10^0 = 5483.$$

Для того чтобы представлять дробные числа, применяется отрицательный показатель степени основания:

$$D = d_{-1}b^{-1} + d_{-2}b^{-2} = 1 \cdot 10^{-1} + 5 \cdot 10^{-2} = 0.15.$$

В общем виде число в позиционной системе счисления записывается и вычисляется так:

$$D = d_{p-1}b^{p-1} + d_{p-2}b^{p-2} + \ldots + d_1b^1 + d_0b^0.\ d_{-1}b^{-1} + d_{-2}b^{-2} + \ldots + d_{-n}b^{-n} = \sum_{i=-n}^{p-1} d_ib^i,$$

где p — число цифр, расположенных слева от точки, а n — число цифр, расположенных справа.

Пример для десятичной системы:

$$D = d_2b^2 + d_1b^1 + d_0b^0.\ d_{-1}b^{-1} + d_{-2}b^{-2} =$$
$$= 4 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0.\ 1 \cdot 10^{-1} + 5 \cdot 10^{-2} = 423.\ 15_{10}.$$

Пример для двоичной системы ($b = 2$):

$$D = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0.\ 1 \cdot 2^{-1} + 0 \cdot 2^{-2} = 101.\ 1_{(2)} = 5.\ 5_{10}.$$

В целом числе предполагается, что точка находится справа от *правой крайней цифры.* Возможные нули в крайних левых и правых позициях числа не влияют на величину числа и поэтому, как правило, не отображаются. Действительно, число 423.15 равно числу 000423.150.

Такие нули иногда называют незначащими. Крайняя левая цифра в числе называется *цифрой старшего разряда*, а крайняя правая — *цифрой младшего разряда*.

2.1.2. И всего две цифры

Столь привычная для нас десятичная система оказалась неудобной для электронной вычислительной машины. Если в механических вычислительных устройствах, использующих десятичную систему, достаточно просто можно было применить элемент с множеством состояний (колесо с девятью зубьями), то в электронных наиболее просто и надежно реализуются элементы с двумя состояниями. Поэтому естественным явилось применение в ЭВМ *двоичной системы счисления*, т. е. системы, где основание $b=2$.

В этой системе всего две цифры. Каждая цифра называется *двоичной* (от английского *binary digit* — двоичная цифра). Сокращение этого выражения привело к появлению термина *бит*, ставшего названием разряда двоичного числа. Веса разрядов в двоичной системе изменяются по степеням двойки. Поскольку вес каждого разряда умножается либо на 0, либо на 1, то в результате значение числа определяется как сумма соответствующих значений степеней двойки. Ниже в таблице 2 показаны значения весов позиций для 8-разрядного числа.

Таблица 2

Номер разряда	7	6	5	4	3	2	1	0
Степень двойки	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Значение позиций	128	64	32	16	8	4	2	1

Если какой-либо разряд двоичного числа равен 1, то он называется *значащим разрядом*. В таблице 3 показан пример накопления суммарного значения числа за счет значащих разрядов.

Таблица 3

Двоичное число	1	0	0	1	0	0	0	1
Значение позиции	128	64	32	16	8	4	2	1
Значение, входящее в сумму	128			16				1
Суммарное значение	145							

Нетрудно догадаться, что максимальное значение двоичного числа ограничено числом его разрядов и определяется по формуле $M=2^n-1$, где n — число разрядов. В вычислительной технике эти

числа разрядов имеют фиксированные значения — 4, 8, 16, 32, а соответствующие им числа будут иметь следующие максимальные значения:

Число разрядов	Максимальное значение числа
4	15
8	255
16	65535
32	4294967295

Сложить, чтобы вычесть. Арифметические действия, выполняемые в двоичной системе, подчиняются тем же основным правилам, что и в десятичной системе. Только в двоичной системе перенос единиц в старший разряд возникает чаще, чем в десятичной. Вот как выглядит сложение в двоичной системе:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0 + 1$ — *перенос в старший разряд.*

В таблице 4 приведены правила десятичного сложения, а в таблице 5 — двоичного. Сопоставление этих двух таблиц показывает, насколько проще сложение двоичного числа, чем десятичного. Отсюда и реализация в ЭВМ устройства, производящего сложение, очевидно, гораздо проще для двоичной системы.

Таблица 4

+	0	1	2	3	4	5	6	7	8	9	Перенос
0	0	1	2	3	4	5	6	7	8	9	
1	1	2	3	4	5	6	7	8	9	0	1
2	2	3	4	5	6	7	8	9	0	1	1
3	3	4	5	6	7	8	9	0	1	2	1
4	4	5	6	7	8	9	0	1	2	3	1
5	5	6	7	8	9	0	1	2	3	4	1
6	6	7	8	9	0	1	2	3	4	5	1
7	7	8	9	0	1	2	3	4	5	6	1
8	8	9	0	1	2	3	4	5	6	7	1
9	9	0	1	2	3	4	5	6	7	8	1
Перенос		1	1	1	1	1	1	1	1	1	

Таблица 5

+	0	1	Перенос
0	0	1	
1	1	0	1
Перенос		1	

Следующий пример показывает сложение десятичных и соответствующих им двоичных чисел:

Десятичное	Двоичное	Десятичное	Двоичное
2	10	23	10111
+7	+111	+8	+100
9	1001	31	11111

Для упрощения аппаратных средств современных вычислительных машин их арифметические устройства не содержат специальных схем для выполнения вычитания. Эта операция производится тем же устройством, которое производит сложение, и называется *сумматором*. Но для этого вычитаемое должно быть преобразовано предварительно из прямого кода, с которым мы познакомились выше, в специальный код. Ведь в десятичной системе также приходится преобразовывать числа. Сравни: 13 − 5 и 13 + (−5). Такой обратный код в двоичной системе получают путем изменения в числе нулей на единицы, единиц — на нули. Эта операция носит название *инвертирование*. Например, инвертирование числа 0101 даст число 1010. Опыт выполнения операций над числами в обратном коде показал, что они требуют ряда дополнительных преобразований, неизбежно ведущих к усложнению аппаратных средств. Поэтому широкого распространения этот код не получил.

При выполнении арифметических действий результат может получиться не только положительным, но и отрицательным. Как же представить знак «—» в схемах машины, если в них фиксируется лишь два состояния — 0 или 1? Договорились знак числа определять самым левым, старшим битом. Если число положительное, то знаковый разряд равен 0, а если отрицательное — 1. Решение о выделении знакового разряда, конечно, сказалось на максимальных величинах представляемых чисел. Максимальное положительное число в 16 разрядах равно +32767, а отрицательное −32768. Но вернемся к кодам.

Оказалось, что наиболее удобно оперировать двоичными числами в *дополнительном* коде. Единственное усложнение заключается в том, что необходимо для получения дополнительного кода прибавить единицу к *инвертированному* числу, т. е. к *обратному коду* этого числа. Взгляни на таблицу 6. В ней приведены десятичные числа и их двоичные представления в различных формах (для упрощения взяты 4-разрядные числа). Интересным в этой таблице является вот что. Если начать счет с числа 1000(−8) и двигаться вниз по столбцам, то в дополнительном коде каждое последующее число получается прибавлением единицы к предыдущему числу без учета переноса за пределы 4-го разряда. Так просто эту операцию в прямом и обратном кодах не осуществить. Эта особенность дополнительного кода и явилась причиной предпочтительного применения его в современных мини- и микроЭВМ, а также во многих больших ЭВМ.

Итак, числа, представленные в дополнительном коде, складываются по правилам двоичного сложения, но без учета каких-либо переносов

Таблица 6

Десятичное число	Прямой код	Обратный код	Дополнительный код
−8	−	−	1000
−7	1111	1000	1001
−6	1110	1001	1010
−5	1101	1010	1011
−4	1100	1011	1100
−3	1011	1100	1101
−2	1010	1101	1110
−1	1001	1110	1111
0	1000 0000	1111 0000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111

за пределы старшего разряда. Рассмотрим это на следующих примерах:

+2	0010	−2	1110
+	+	+	+
+5	0101	−6	1010
+7	0111	−8	1000
+5	0101	+3	0011
+	+	+	+
−4	1100	−7	1001
+1	0001	−4	1100

Еще одним достоинством дополнительного кода является то, что нуль, в отличие от прямого и обратного кодов, представляется одним кодом. Наличие 0 в знаковом разряде при представлении нуля определяет его как величину положительную, что согласуется с математической теорией чисел и соглашениями, принятыми во всех языках программирования.

Подытоживая наше знакомство с дополнительным кодом, обобщим величину десятичного значения числа в дополнительном коде. Так как вес старшего, т. е. значащего, разряда в данном случае равен -2^{n-1}, а не $+2^{n-1}$, как в прямом коде, то диапазон представления чисел находится от $-(2^{n-1})$ до $+(2^{n-1}-1)$.

Умножение возможно без таблицы, а деление — еще проще. Способ умножения десятичных чисел, которым ты владеешь в совершенстве благодаря постоянной практике начиная с первого класса, применим и для умножения двоичных чисел. Здесь все происходит проще хотя бы потому, что не надо помнить таблицу умножения.

Достаточно знать следующие простые правила умножения двоичных разрядов.

Множимое		Множитель		Произведение
0	×	0	=	0
0	×	1	=	0
1	×	0	=	0
1	×	1	=	1

Другими словами, процедура умножения сводится к записи нулей, если разряд множителя равен 0, и записи множимого, если разряд множителя равен 1.

Двоичное деление основано на методе, знакомом тебе по десятичному делению, т. е. сводится к выполнению операций умножения и вычитания. Выполнение основной процедуры — выбор числа, кратного делителю и предназначенного для уменьшения делимого, здесь проще, так как таким числом может быть либо 0, либо сам делитель.

Для деления чисел со знаками в дополнительном коде существует несколько методов. Простейший из них заключается в преобразовании этих чисел в положительные, в выполнении деления, как для чисел без знака; последующем восстановлении знака результата.

2.1.3. Восемь, шестнадцать... и все-таки десять

При наладке аппаратных средств ЭВМ или создании новой программы часто возникает необходимость «заглянуть внутрь» памяти машины, чтобы оценить ее текущее состояние. Но там все заполнено длинными последовательностями нулей и единиц двоичных чисел. Эти последовательности очень неудобны для восприятия человеком, привыкшим к более короткой записи десятичных чисел. Кроме того, естественные возможности человеческого мышления не позволяют оценить быстро и точно величину числа, представленного, например, комбинацией из 16 нулей и единиц. Для облегчения восприятия двоичного числа решили разбивать его на группы разрядов, например по 3 или 4 разряда. Эта идея оказалась очень удачной, так как последовательность из 3 бит имеет 8 комбинаций, а последовательность из 4 бит — 16 комбинаций. Числа 8 и 16 являются степенями двойки, поэтому легко находить соответствие с двоичными числами. Развивая эту идею, пришли к выводу, что группы разрядов можно закодировать, сократив при этом длину последовательности знаков. Для кодировки трех битов (триад) требуется 8 цифр, и поэтому взяли цифры от 0 до 7 десятичной системы. Для кодировки же четырех битов (тетрад) необходимо 16 знаков; для этого взяли 10 цифр десятичной системы и 6 букв латинского алфавита: *A, B, C, D, E, F*. Полученные системы, имеющие основания 8 и 16, назвали соответственно *восьмеричной* и *шестнадцатеричной*. В таблице 7 приведены числа в десятичной, восьмеричной и шестнадцатеричной системах и соответствующие группы бит двоичной системы.

Таблица 7

Десятичное число	Восьмеричное число	Последовательность из 3 бит	Шестнадцатеричное число	Последовательность из 4 бит
0	0	000 000	0	0000
1	1	000 001	1	0001
2	2	000 010	2	0010
3	3	000 011	3	0011
4	4	000 100	4	0100
5	5	000 101	5	0101
6	6	000 110	6	0110
7	7	000 111	7	0111
8	10	001 000	8	1000
9	11	001 001	9	1001
10	12	001 010	A	1010
11	13	001 011	B	1011
12	14	001 100	C	1100
13	15	001 101	D	1101
14	16	001 110	E	1110
15	17	001 111	F	1111

16-разрядное двоичное число со знаковым разрядом можно представить 6-разрядным восьмеричным, причем старший разряд в нем будет принимать значения лишь 0 или 1. В шестнадцатеричной системе такое число займет 4 разряда.

Легкость преобразования двоичных чисел в восьмеричные и шестнадцатеричные видна из следующего примера:

1100001111010110

1100 0011 1101 0110	1 100 001 111 010 110
C 3 *D* 6	1 4 1 7 2 6

Из этого примера следует, что для преобразования двоичного числа в восьмеричное необходимо двоичную последовательность разбить на триады справа налево и каждую группу заменить соответствующей восьмеричной цифрой. Аналогично поступаем и при преобразовании в шестнадцатеричный код, только двоичную последовательность разбиваем на тетрады и для замены используем шестнадцатеричные знаки.

Так же просто осуществляется и обратное преобразование. Для этого каждую цифру восьмеричного или шестнадцатеричного числа заменяют группой из 3 или 4 бит. Например:

A *B* 5 1	1 7 7 2 0 4
1010 1011 0101 0001	1 111 111 010 000 100

Арифметические операции над числами в восьмеричной и шестнадцатеричной системах проводятся по тем же правилам, что и в десятичной системе. Только надо помнить, что если имеет место *перенос*, то переносится не 10, а 8 или 16.

Сейчас ты можешь считать, что освоил четыре системы счисления: «машинную» — *двоичную*, «человеческую» — *десятичную* и две промежу-

точные — *восьмеричную* и *шестнадцатеричную*. Каждая из них применяется в различных процессах, связанных с ЭВМ: двоичная — для организации машинных операций по преобразованию информации, восьмеричная и шестнадцатеричная — для представления машинных кодов в удобном виде при работе профессиональных пользователей — программистов и аппаратчиков. Десятичная же система остается для представления результатов деятельности ЭВМ, отображаемых на устройствах печати и экранах видеотерминалов, а также для ввода данных в ЭВМ. Поэтому в машине постоянно происходят процессы преобразования чисел из десятичной системы в двоичную и наоборот. Да и человеку, имеющему дело с ЭВМ, часто приходится прибегать к преобразованиям чисел между двоичной, восьмеричной, шестнадцатеричной и десятичной системами.

Для перевода из десятичной системы в другую систему обычно применяется *метод последовательного деления* исходного числа на основание системы счисления, в которую переводится число. Полученный остаток после первого деления является младшим разрядом нового числа. Образовавшееся частное снова делится на это основание. Из остатка получаем следующий разряд нового числа и т. д. Давай переведем десятичное число 212 в двоичную систему:

```
_212 | 2
 212   106 | 2
   0   106   53 | 2
         0    52   26 | 2
              1    26   13 | 2
                   0    12   6 | 2
                        1    6   3 | 2
                             0    2   1 | 2
                                  1    0
```

$212_{(10)} = 11010100_{(2)}$ (старший разряд)

А теперь переведем десятичное число 31318 в восьмеричную систему:

```
_31318 | 8
 31312   3914 | 8
     6   3912   489 | 8
             2   488   61 | 8
                   1    56   7 | 8
                        5    0
```

$31318_{(10)} = 75126_{(8)}$ (старший разряд)

Перевод из одной системы в другую дробных чисел производится по правилу, требующему не делить, а умножать дробную часть на величину основания нового числа. В качестве примера преобразуем десятичное число 2638.75 в шестнадцатеричную систему. Это действие произведем в два этапа — сначала для целой, а затем для дробной части:

$$
\begin{array}{r|l}
2638 & 16 \\
\hline
2624 & 164 \quad | \; 16 \\
\hline
14 & 160 \quad 10 \quad | \; 16 \\
\hline
 & 4 \quad 0 \\
\hline
 & 10
\end{array}
$$

(старший разряд целой части)

$\frac{75}{100} \cdot 16 = 12$ $\qquad 2638.75_{(10)} = \mathrm{A4E.C}_{(16)}$

2.1.4. Куда поставить точку?

Слева, справа, посредине... При рассмотрении систем счисления мы оперировали в основном целыми числами, т. е. числами, у которых точка, отделяющая целую часть числа от дробной, располагается справа от правого крайнего разряда. Такое представление чисел подходит для счета каких-либо объектов. В инженерных и научных расчетах не обойтись без учета дробных чисел. Тогда точку можно располагать левее от крайних правых разрядов, добиваясь при этом необходимой точности вычислений. Так, в 16-разрядном двоичном числе расположение точки справа от левого крайнего разряда даст максимальную точность при вычислении положительных значений синуса:

$$0.000000000000000_2 = 0_{(10)}$$
$$0.100000000000000_2 = 0.5_{(10)}$$
$$1.000000000000000_2 = 1.0_{(10)}$$

В общем случае положение точки в числе может быть любым, но в дальнейших операциях неизменным. Такое представление числа называется *представлением в формате с фиксированной точкой.*

Сложение и вычитание чисел с фиксированной точкой производится по правилам обычного двоичного сложения и вычитания, так как результат операции не влияет на положение точки. Однако при выполнении умножения и деления необходимо осуществлять коррекцию положения точки. Рассмотрим два примера, помня, что веса разрядов, расположенных справа от двоичной точки, являются отрицательными степенями двойки:

$$
\begin{array}{r}
x \cdot 2^{-3} \\
+ \\
\underline{y \cdot 2^{-3}} \\
(x + y)\, 2^{-3}
\end{array}
\qquad\qquad
\begin{array}{r}
x \cdot 2^{-5} \\
\times \\
\underline{y \cdot 2^{-5}} \\
((xy)2^{-5})\, 2^{-5} = xy \cdot 2^{-10}
\end{array}
$$

Наличие дополнительных вычислений при представлении дробных чисел в *формате с фиксированной точкой* затрудняет расчеты на ЭВМ, но если это все же необходимо, то программист должен сам следить за положением точки: выполнять операции отдельно для целой части числа и для дробной, а затем сводить их в единое результирующее число.

А лучше — без точки. Оба недостатка *формата с фиксированной точкой* (слежение за положением точки и сравнительно небольшой диапазон представляемых чисел) устраняются представлением чисел в *формате с плавающей точкой*. В этом формате разряды числа разбиваются на два поля, имеющие названия *мантисса* и *порядок*. Если обозначить мантиссу буквой M, а порядок букв — P, то величина числа $X = \pm M \pm P$. Эта запись является двоичным эквивалентом знакомой тебе формы записи десятичных чисел $X = M \cdot 10^E$, например: $200 = 2 \cdot 10^2$, $36000000000 = 36 \cdot 10^9$. Структура 16-разрядного числа в представлении с плавающей точкой и примеры даны в таблице 8.

Таблица 8

15	14 10	9	8 0	Результат
Знак порядка	Модуль порядка	Знак мантиссы	Модуль мантиссы	
Пример				
0	00000	0	000000000	$=0 \cdot 2^0$
0	00000	1	000000001	$=-1 \cdot 2^0$
1	00100	0	010001100	$=140 \cdot 2^{-4}$
0	11111	0	111111111	$=511 \cdot 2^{31}$

Из последнего примера видно, что всего 16 разрядов могут представить очень большие числа. Но, отобрав шесть разрядов под порядок, мы уменьшили точность представления чисел. Так, в приведенной структуре единица отстоит от ближайшего дробного числа на 2^{-10}, тогда как в формате с фиксированной точкой — на 2^{-17}. Интересной особенностью формата с плавающей точкой является возможность представления одного числа различными комбинациями значений мантиссы и порядка. Так, например, нуль в этом формате может быть записан 64 способами (мантисса равна нулю, а порядок принимает любое значение). Другие числа могут иметь до девяти представлений, например: $32 = 1 \cdot 2^5 = 2 \cdot 2^4 = 4 \cdot 2^3 = 8 \cdot 2^2 = 16 \cdot 2^1 = 32 \cdot 2^0$; $2560 = 5 \cdot 2^9 = 10 \cdot 2^8 = 20 \cdot 2^7 = 40 \cdot 2^6 = 80 \cdot 2^5 = \ldots = 1280 \cdot 2^1$. Несмотря на это, представление чисел в формате с плавающей точкой оказалось достаточно удобным для обработки на ЭВМ больших и дробных чисел, хотя при этом пришлось пойти на некоторые дополнения. Так, например, чтобы увеличить точность числа для его представления отводят два, а иногда и четыре 16-разрядных поля. Вообще же в различных вычислительных машинах используются отличающиеся друг от друга форматы с плавающей точкой, но основаны они на едином принципе представления: порядок и мантисса.

Для выполнения арифметических операций над числами в формате с плавающей точкой используются точные правила, зависящие от конкретной реализации в конкретной ЭВМ, но содержащие общий подход. Так, сложение и вычитание чисел с плавающей точкой сводится к выравниванию позиций точек с тем, чтобы оба числа имели одинаковый порядок, а затем производится сложение или вычитание мантисс. Для умножения и деления выравнивание позиций точек не требуется; производится лишь сложение (при умножении) или вычитание (при делении) порядков и умножение или деление мантисс.

На ЭВМ, ориентированных для выполнения большого количества операций с числами в формате с плавающей точкой, имеются специальные аппаратные средства, автоматически реализующие порядок действий при арифметических вычислениях и преобразованиях таких чисел.

2.1.5. Символы и коды

Исследуя системы счисления, мы как бы заглянули в вычислительную машину, где с огромной скоростью преобразуются электрические сигналы, которые для удобства назвали единицами и нулями. Наконец операции по обработке необходимой для нас информации завершены, и она должна быть выдана или на бумажную ленту, или прямо на экран *видеотерминала*. Здесь уже машинные нули и единицы предстают в *символьном* начертании цифр и букв, складывающихся в определенные последовательности чисел, слов и других знаков, т. е. в текстовую информацию. Устройства печати и видеотерминалы также имеют электронную «начинку» и могут распознавать лишь все те же машинные нули и единицы, которые они преобразуют в привычные для нас символы, рисуемые на бумаге механическим устройством или на экране потоком электронов. Таким образом, как ты уже, видимо, догадался, за каждым символом кроется определенная последовательность двоичных разрядов — *битов*. Договорились о порядке применения этих последовательностей. Наиболее широко используется рекомендованный международными организациями код ASCII *(American Standard Code for Information Interchange* — стандартный американский код для обмена информацией). Этому коду соответствует отечественный стандартный код, получивший название КОИ-7. Вообще надо знать, что на все коды имеются государственные стандарты. На КОИ-7 дан ГОСТ 13052-74, где каждому символу соответствует 7-битовая последовательность; таким образом, может быть представлено 128 различных символов, которые и приведены в таблице 9. Используя эти коды, можно, например, строку СУММА = 25 представить так: 1110011 1110101 1101101 1101101 1100001 0100000 0111101 0100000 0110010 0110101. Некоторые 7-битовые коды служат не для представления отображаемых символов, а для управления устройствами отображения и обеспечения достоверного обмена по линиям связи этих устройств с ЭВМ. Так, например, код 0001010 (12_8) используется для возврата курсора (указателя) на экране видеотерминала к началу строки, а код 0000100 (4_8) сообщает о конце передаваемого текста.

Таблица 9

		Старшие разряды	7	0	0	0	0	1	1	1	1
			6	0	0	1	1	0	0	1	1
			5	0	1	0	1	0	1	0	1
0	0	0	0	Служебные символы		Пробел	0	@	*P*	Ю	П
0	0	0	1			!	1	*A*	*Q*	А	Я
0	0	1	0			»	2	*B*	*R*	Б	Р
0	0	1	1			#	3	*C*	*S*	Ц	С
0	1	0	0			¤	4	*D*	*T*	Д	Т
0	1	0	1			%	5	*E*	*U*	Е	У
0	1	1	0			&	6	*F*	*Y*	Ф	Ж
0	1	1	1			'	7	*G*	*W*	Г	В
1	0	0	0			(	8	*H*	*X*	Х	Ь
1	0	0	1			)	9	*I*	*Y*	И	Ы
1	0	1	0			*	:	*J*	*Z*	Й	З
1	0	1	1			+	;	*K*	[	К	Ш
1	1	0	0			,	<	*L*	\	Л	Э
1	1	0	1			−	=	*M*	]	М	Щ
1	1	1	0			.	>	*N*	¬	Н	Ч
1	1	1	1			/	?	*O*	_	О	Забой
4	3	2	1								
Младшие разряды											

Международным стандартным кодом является также код *EBCDIC (Extended Binary Coded Decimal Interchange Code* — расширенный двоично-кодированный десятичный код для обмена информацией). Ему соответствует отечественный стандартный код ДКОИ (двоичный код для обмена и обработки информации, ГОСТ 19768-74). Отличительной особенностью этого кода является 8-битовое представление символов. Зная код КОИ-7, может показаться, что использование 8 бит неэффективно, но во многих случаях это дает ряд преимуществ, так как именно 8 бит позволяет кодировать 256 различных символов. Дополнительные коды можно использовать для представления символов верхнего и нижнего регистров печатающего устройства, а также многих специальных символов. Допускается перекодировка представления цифр в упакованный формат, когда в 8-битовом поле кодируется две цифры — по 4 бита на цифру, что дает экономию памяти и увеличивает скорость передачи данных.

Двоичное кодирование символов облегчает всевозможные манипуляции с ними. Так, чтобы получить строку 123456789 в коде КОИ-7, достаточно прибавить последовательно к числу 0110000 единицу девять раз.

2.2 КАК РАССЧИТЫВАЮТ СХЕМЫ ДЛЯ ЭВМ?

2.2.1. Высказывания и сентенции

Великого английского писателя В. Шекспира можно поставить в один ряд с первооткрывателями основ электронной вычислительной техники. Его знаменитая фраза «быть или не быть» — логическое высказывание, и построено оно по схеме ИЛИ — НЕ.

Но обо всем расскажем по порядку. Выясним сначала, что такое высказывание.

Под *высказыванием* понимается повествовательное предложение, которое может быть либо истинным, либо ложным, но не тем и другим одновременно. Исходя из значения слова *не,* можно убедиться, что если такое высказывание, как «3 является простым числом», истинно, то его отрицание — «3 не является простым числом» — ложно, и наоборот, если высказывание «4 является простым числом» ложно, то «4 не является простым числом» — истинно.

Понятия *истинно* (по-английски *true)* и *ложно (false)* удобнее заменить значениями 1 и 0. (Здесь и далее будут встречаться английские слова, так как они широко используются в профессиональном программировании.) Это не арифметические понятия «единица» или «нуль». Логический 0 не означает количества, т. е. «ничто», а индицирует (отмечает) одно из состояний какого-либо объекта, если этот объект может находиться попеременно в двух состояниях. Тогда логическая 1 индицирует другое его состояние. Это относится к рассмотренным нами высказываниям, а также к некоторым электронным или механическим устройствам. Такие устройства с двумя устойчивыми состояниями получили название *бистабильные,* к которым относятся, например, *реле.*

Применение реле стало революционным новшеством в развитии вычислительной техники. (Термин *реле* появился во Франции в средние века. Этим словом называли замену уставших лошадей свежими на почтовой станции.)

Электромагнитное устройство, переключающее одну цепь на другую, также назвали *реле.* Для вычислительной техники важным является то, что одно положение контактов реле можно воспринимать как состояние, соответствующее единице, а другое — нулю.

Релейный принцип получил развитие в машине *Mark-1,* созданной Г. Айкеном в США (1944 г.). Однако скорость релейных машин не отвечала растущим требованиям. Та же *Mark-1* выполняла умножение двух чисел только за 6 с. Особенно остро этот недостаток ощущался в военном деле. Шла вторая мировая война, которая ускорила разработку быстродействующих вычислительных устройств, широко применявшихся в зенитной артиллерии. Основу для таких устройств заложил еще в 1940 г. Н. Винер. Он сформулировал требования, которым должны удовлетворять эти устройства:

1. Быть цифровыми, как в обычном арифмометре.
2. Состоять не из механических частей, а из электронных ламп.
3. Использовать двоичную, а не десятичную систему счисления.

4. Последовательность действий планировать самой машиной так, чтобы человек не вмешивался в процесс решения задачи с момента введения исходных данных до съема окончательных результатов.

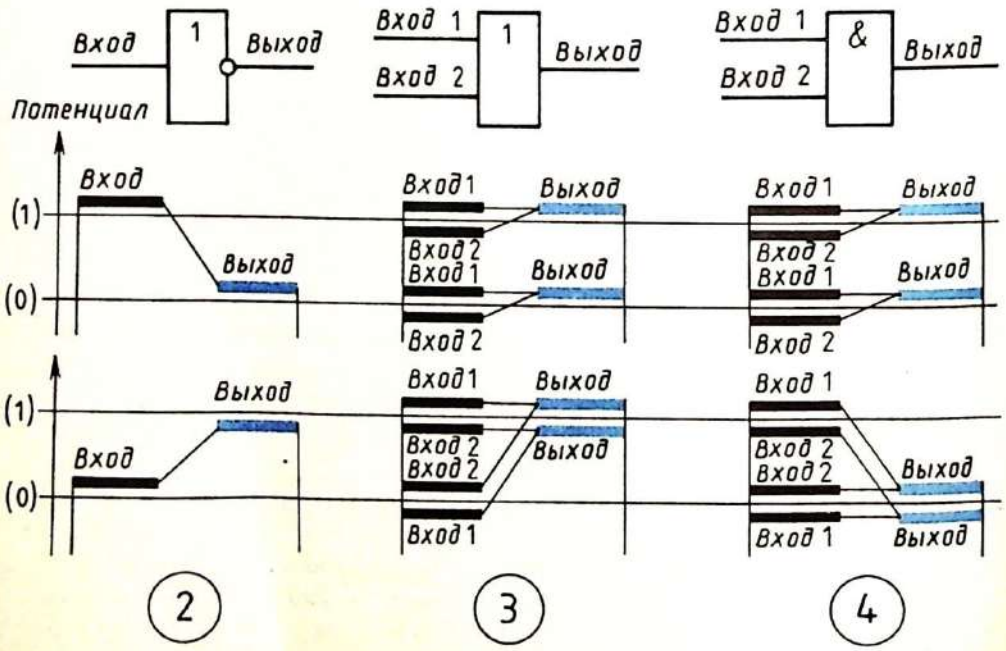
5. Иметь устройства для записи и хранения результатов.

Последующее развитие вычислительной техники показало, что эти требования Винера были верными.

А сейчас рассмотрим — как же представляются нули и единицы в электронных схемах? Договорились отличать эти состояния по различным уровням электрического напряжения, прикладываемого к участку схемы. Например, единице соответствует потенциал в 5 В, а нулю — 0 В по отношению к заземленному участку схемы. Электронные схемы, преобразующие сигналы только двух фиксированных уровней напряжения, получили название *логических* элементов. Эти схемы состоят из набора полупроводниковых элементов, в основном транзисторов. В свою очередь, из логических элементов собираются более сложные устройства — счетчики, сумматоры, триггеры и др. Из последних формируются целые узлы ЭВМ.

Логический элемент на схемах изображают прямоугольником. Линии с левой стороны прямоугольника показывают входы, а с правой — выходы элемента. Внутри прямоугольника изображается указатель логической функции, выполняемой данным элементом. Такими указателями могут быть различные значки; иногда ими служат слова-связки, присутствующие в логических высказываниях. Это известные нам *не, или,* а также *и*. Эти связки называют *сентенциональными*. Они часто встречаются в рассуждениях людей, в устном или письменном изложении для связывания повествовательных предложений.

Предложение со словом *не* (английское *no)* называется *отрицанием*



первоначального предложения. Например, «4 не есть простое число» — это отрицание такого предложения, как «4 есть простое число», «Петров не готов ловить комаров» — отрицание того, что Петров готов ловить комаров.

Функцию отрицания в электронике выполняет логический элемент НЕ — *инвертор* (рис. 2). Инвертор имеет один вход и один выход и устроен так, что если подать на его вход потенциал, соответствующий единице (например, 5 В), то на его выходе будет потенциал, соответствующий нулю (в данном случае 0 В). Работа инвертора описывается таблицей 10.

Таблица 10

Потенциал на входе соответствует	Потенциал на выходе соответствует
0	1
1	0

Слово *или (or)* служит для связки двух простых предложений в сложное. Такая связка в логике называется *дизъюнкцией* или *логическим сложением*. Например, предложение «Игра со спичками или неосторожное обращение с электронагревательными приборами приводит к пожару» является дизъюнкцией предложений «Игра со спичками приводит к пожару» и «Неосторожное обращение с электронагревательными приборами приводит к пожару». Не будем при этом обращать внимание на грамматические особенности языка, приведшие к объединению в дизъюнктивном предложении слов «приводит к пожару», имеющихся в исходных предложениях.

Логический элемент ИЛИ называют дизъюнктором. Он имеет два входа (рис. 3). Работа его задается таблицей 11.

Таблица 11

Потенциал на входах соответствует		Потенциал на выходе соответствует
0	0	0
0	1	1
1	0	1
1	1	1

Слово *и (and)* применяется для соединения двух простых предложений в сложное. Называется это соединение *конъюнкцией*. «В камине гаснет огонек и свечка нагорела» — конъюнкция, предложенная А. С. Пушкиным.

Конъюнктор (логический элемент И) обозначается как на рисунке 4, а его работа соответствует таблице 12.

Помимо указанных логических элементов существует еще множество их, которые выполняют более сложные логические преобразования. Эти преобразования являются комбинациями простейших логических операций. К числу таких элементов относятся, например, элементы ИЛИ — НЕ, И — НЕ, И — ИЛИ — НЕ.

Таблица 12

Потенциал на входах соответствует		Потенциал на выходе соответствует
0	0	0
0	1	0
1	0	0
1	1	1

2.2.2. Таблицы и алгебра

Анализ комбинационных устройств проще всего производить с помощью специального математического аппарата — *алгебры логики*, разработанной (как мы уже говорили) английским математиком Дж. Булем. В честь его имени этот аппарат, оперирующий только с двумя величинами — логическая 1 (истина) и логический 0 (ложь), называют булевой алгеброй, а сами логические переменные — булевыми переменными. Если заменить предложения, участвующие в высказывании, буквами A и B, то выражения

$$\overline{A},\ AB,\ A+B$$

будут соответственно отрицанием предложения A, конъюнкцией и дизъюнкцией предложений A и B. Здесь черта над буквой — это НЕ, точка, которую чаще всего опускают — И, плюс — ИЛИ.

В алгебре логики применяются таблицы истинности, при помощи которых можно определять истинностное значение любого высказывания для всех возможных случаев значений истинности составляющих его высказываний. Таблицы истинности имеют в логике такое же значение, как таблица умножения в арифметике, а запоминаются гораздо легче, в чем ты можешь немедленно убедиться, если разберешь таблицу 13.

Таблица 13

Исходное высказывание		Отрицание $\bar{A}$	Конъюнкция AB	Дизъюнкция A + B
A	B			
0	0	1	0	0
0	1		0	1
1	0	0	0	1
1	1		1	1

При анализе электронных логических элементов входные величины обозначают через X1, X2 и т. д., выходные через Y. *Таблица истинности* для элемента ИЛИ — НЕ (рис. 5), соответствующая логической функции $Y=\overline{X1+X2}$, представлена таблицей 14, а для элемента И — НЕ (рис. 6) функции $Y=\overline{X1\ X2}$ — таблицей 15.

Таблицы истинности являются таблицами, описывающими работу логических элементов. Но чем сложнее функция, соответствующая логическому элементу, тем труднее построить таблицу истинности.

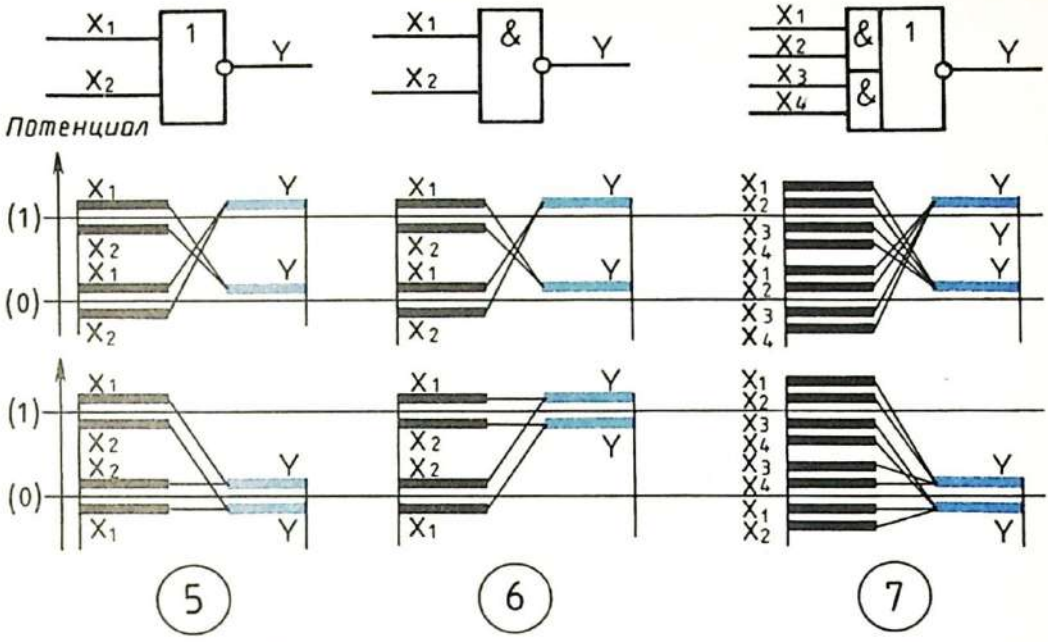


Таблица 14

X1	X2	Y
0	0	1
0	1	0
1	0	0
1	1	0

Таблица 15

X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

Для элемента И — ИЛИ — НЕ (рис. 7), имеющего четыре входа и реализующего функцию $Y = \overline{X1\ X2 + X3\ X4}$, таблица будет достаточно большой — из 16 строк, в чем ты можешь при желании убедиться.

Сложные сочетания высказываний получили название *логических формул*. С ростом числа входных переменных одну и ту же логическую функцию можно выразить различными формулами, которые тоже представлены в таблицах 16 и 17.

Таблица 16

X1	X2	X3	Y
1	1	1	1
1	1	0	0
0	1	1	0
0	1	0	0

Таблица 17

X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

Им соответствуют функции $Y = X1\ X2\ X3$ и $Y = X1\ X2$. За формулами электронных логических схем стоят реальные электронные элементы. Поэтому разработчики логических схем стремятся выбирать формулы, содержащие меньшее число элементов, что ведет к удешевлению и повышению надежности узлов ЭВМ. Поэтому для упрощения начальных формул проводятся специальные действия, приводящие к

более удобному виду. Эти преобразования и позволяет сделать алгебра логики. В основе ее лежат некоторые законы, часть из которых аналогична соответствующим законам обычной алгебры. Например, *переместительный закон* для дизъюнкции и конъюнкции имеет вид:

$$A + B = B + A,$$

$$AB = BA,$$

а *сочетательный:*

$$(A + B) + C = A + (B + C),$$
$$(AB)\,C = A\,(BC).$$

Используя эти и некоторые другие законы, можно, как и в обычной алгебре, раскрывать скобки, выносить за скобки общий множитель и т. д. При этом принимается: действие конъюнкции предшествует действию дизъюнкции, как в обычной алгебре умножение предшествует сложению.

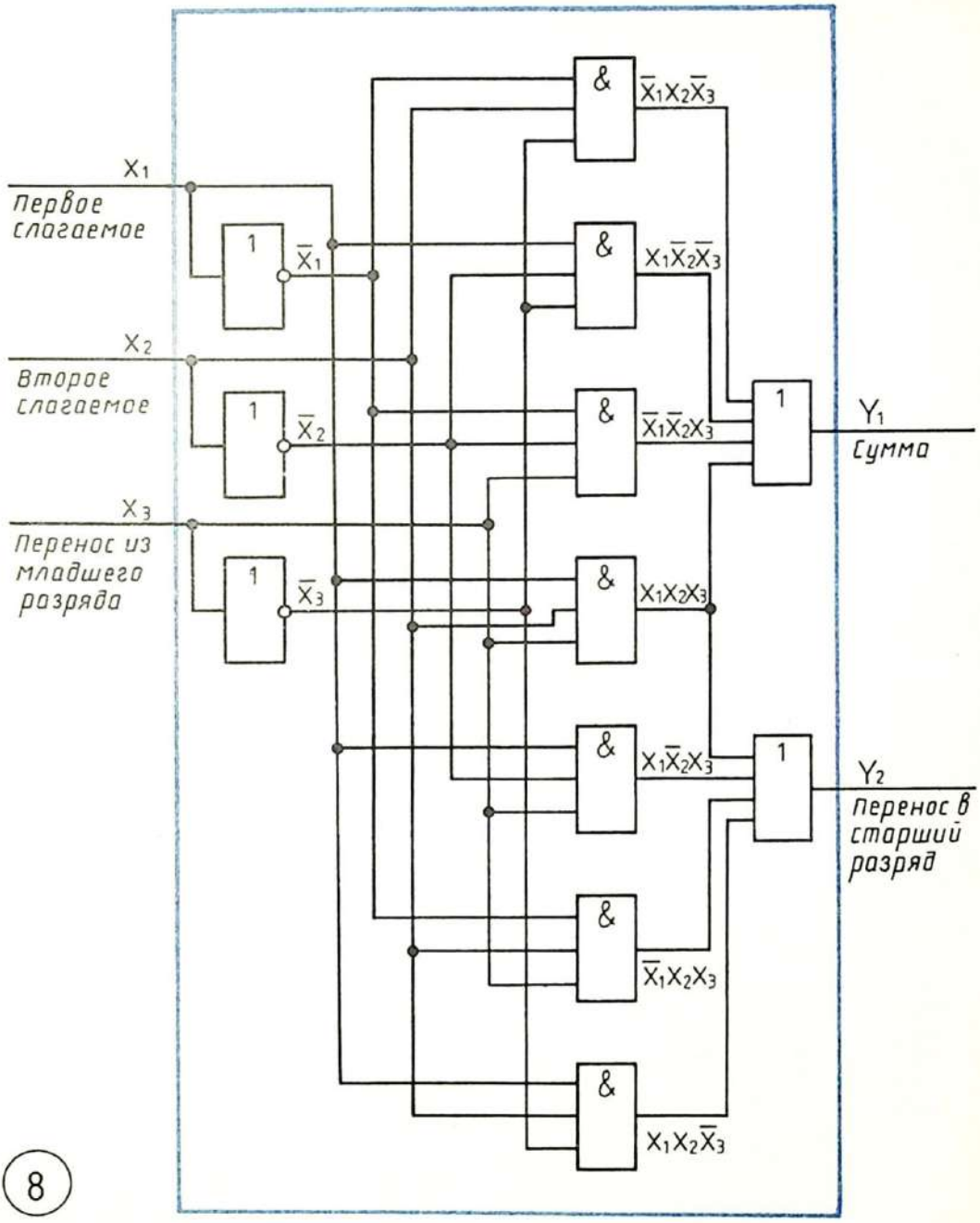
Теперь выясним, что представляет собой *синтез логической схемы.* Это — процесс, состоящий из нескольких этапов. На первом этапе строят таблицу истинности проектируемого узла по заданным условиям работы, т. е. по соответствию его входных и выходных сигналов. На втором этапе по таблице истинности записывают формулу логической функции данного узла, а затем производят преобразование (минимизацию) этой формулы, если есть в этом необходимость (третий этап). Наконец (четвертый этап) изображается логическая схема узла.

2.2.3. Логические схемы

Сумматор. Построим логическую схему устройства, предназначенного для арифметического сложения двух чисел,— *сумматора.* По известному нам правилу сложения многоразрядных двоичных чисел каждый разряд суммы формируется из разрядов слагаемых и переноса из младшего разряда; кроме того, формируется перенос в старший разряд. Простейшим сумматором является сумматор одноразрядный. Таблицу истинности такого устройства см. в таблице 18.

Таблица 18

Вход			Выход	
Слагаемое		Перенос из младшего разряда	Сумма	Перенос в старший разряд
X1	X2	X3	Y1	Y2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



8

Соответствующий логический элемент должен иметь три входа и два выхода. Логические функции для него запишутся так:

$$Y1 = \overline{X1}\ X2\ \overline{X3} + X1\ \overline{X2}\ \overline{X3} + \overline{X1}\ \overline{X2}\ X3 + X1\ X2\ X3,$$

$$Y2 = X1\ X2\ \overline{X3} + \overline{X1}\ X2\ X3 + X1\ \overline{X2}\ X3 + X1\ X2\ X3.$$

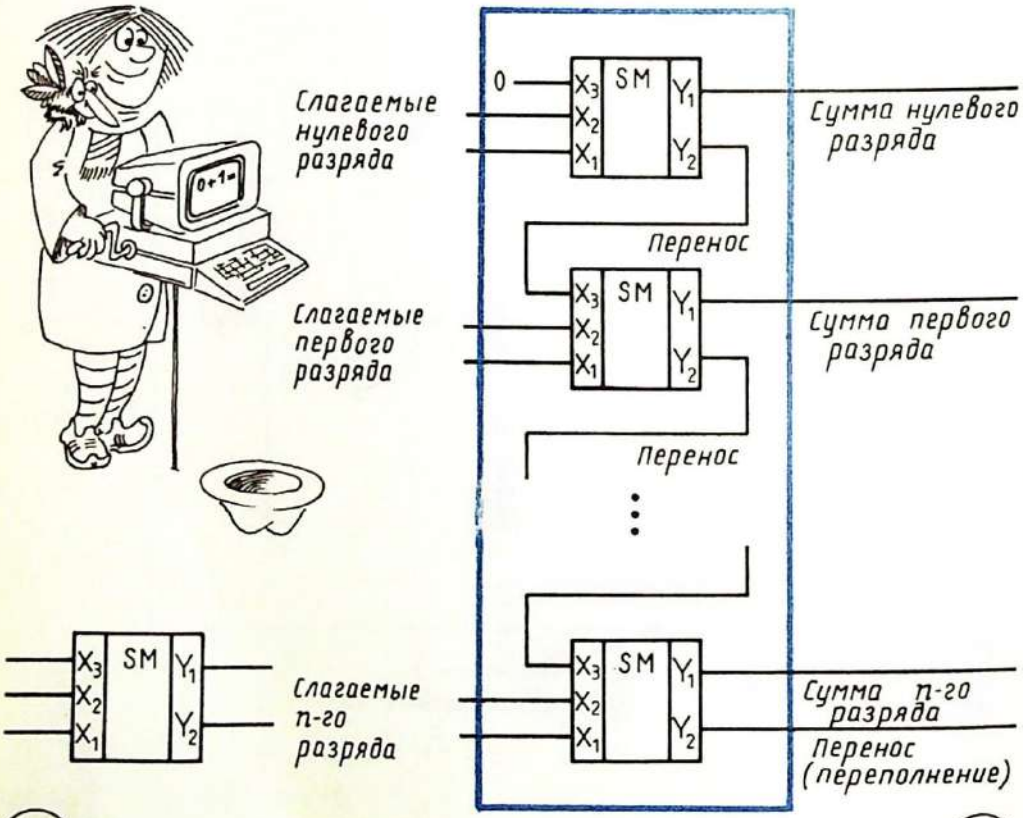
По этим функциям можно построить одноразрядный сумматор на элементах И и ИЛИ (рис. 8). Попытка минимизации приведенных фор-

мул не дает более удобного представления, поэтому мы не будем здесь ее проводить.

То, что изображено на рисунке 8, может быть заменено одним прямоугольником, как показано на рисунке 9, с пометкой *SM* — сумматор. Теперь для сложения двух многоразрядных двоичных чисел соединим последовательно *n* одноразрядных сумматоров и получим схему, приведенную на рисунке 10. На вход X1 сумматора нулевого разряда подается постоянный 0. Если в результате сложения на выходе Y2 последнего сумматора *n*-го разряда появляется 1, то происходит переполнение сумматора. Этот выход обычно заводят в специальную схему ЭВМ, где фиксируется факт переполнения, который может быть проанализирован программистом. Но об этом мы узнаем несколько позже.

Схемы, приведенные на рисунках 8 и 10, реализуются отдельными микросхемами, выпускаемыми в виде одно-, двух- и четырехразрядных сумматоров. Соединяя их между собой, получают сумматоры с требуемой разрядностью.

Кодер-декодер. Для связи с внешними устройствами, как мы уже знаем, необходимо преобразовывать числа из одной формы представления в другую. Например, при вводе с клавиатуры дисплея десятичных цифр 0...9 сигнал, поступающий от нажатой клавиши, необходимо пре-



9

10

образовать в двоичный код. Устройство, преобразующее одиночный сигнал в двоичный код, называется *шифратором* (кодером). Посмотри на таблицу 19, описывающую работу шифратора десятичных цифр.

Таблица 19

Вход (сигнал от нажатой клавиши)										Выход (двоичный код)			
0	1	2	3	4	5	6	7	8	9				
X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y4	Y3	Y2	Y1
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

При нажатии любой клавиши возникает сигнал, соответствующий 1 (одновременное нажатие нескольких клавиш запрещено). На выходе Y1 единица появляется при нажатии нечетной клавиши (X1, X3, X5), т. е. $Y1 = X1 + X3 + X5 + X7 + X9$. Для остальных выходов функции имеют вид:

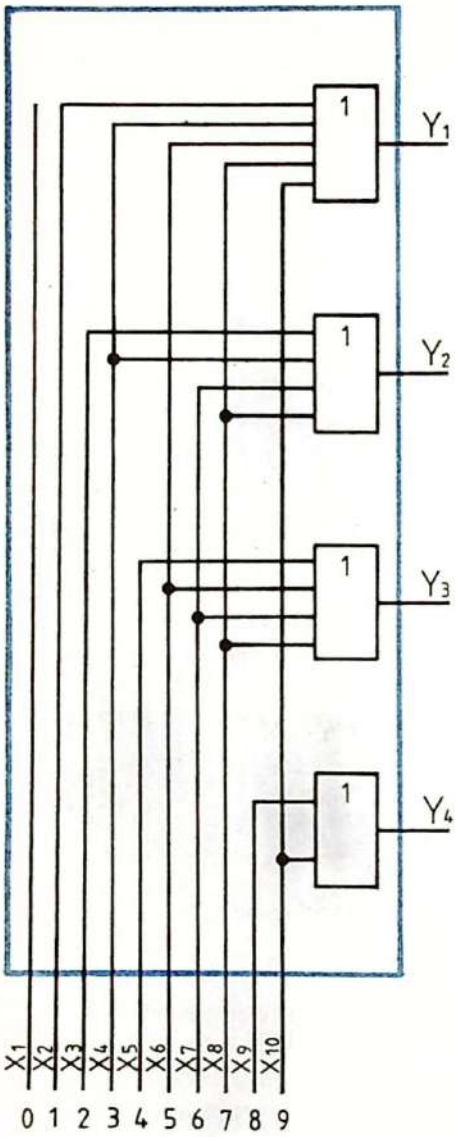
$$Y2 = X2 + X3 + X6 + X7,$$
$$Y3 = X4 + X5 + X6 + X7,$$
$$Y4 = X8 + X9.$$

Соответствующая логическая схема шифратора показана на рисунке 11.

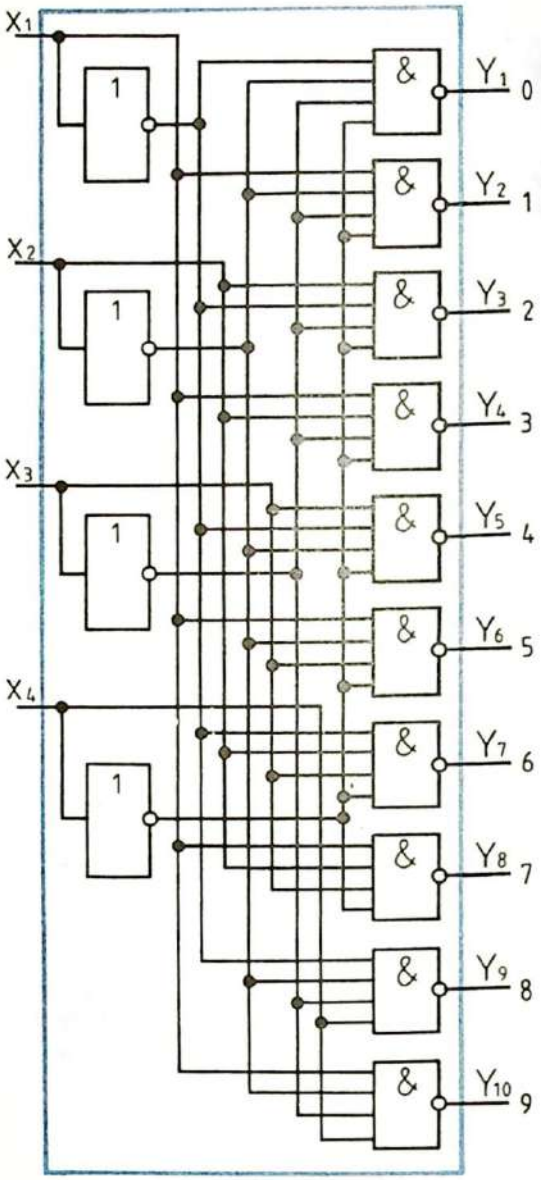
Процедура преобразования двоичного кода в единичный сигнал выполняется *дешифратором (декодером)*. На его входы поступает двоичный код. При этом на одном из выходов возникает единичный сигнал. Этот сигнал может быть подан, например, на схему, управляющую свечением экрана дисплея, для отображения соответствующей десятичной цифры. Таблица 20 дает представление о работе дешифратора.

Логические функции для каждого выхода запишутся так:

$$Y1 = \overline{X1}\ \overline{X2}\ \overline{X3}\ \overline{X4},$$
$$Y2 = X1\ \overline{X2}\ \overline{X3}\ \overline{X4},$$
$$Y3 = \overline{X1}\ X2\ \overline{X3}\ \overline{X4},$$
$$Y4 = X1\ X2\ \overline{X3}\ \overline{X4},$$
$$Y5 = \overline{X1}\ \overline{X2}\ X3\ \overline{X4},$$
$$Y6 = X1\ \overline{X2}\ X3\ \overline{X4},$$
$$Y7 = \overline{X1}\ X2\ X3\ \overline{X4},$$
$$Y8 = X1\ X2\ X3\ \overline{X4},$$



11



12

$$Y9 = \overline{X1}\ \overline{X2}\ \overline{X3}\ X4,$$
$$Y10 = X1\ \overline{X2}\ \overline{X3}\ X4.$$

Схема дешифратора, построенная из элементов ИЛИ — НЕ и И — НЕ, показана на рисунке 12.

Триггер и регистр. Сумматор, шифратор, дешифратор — это устройства, преобразующие информацию. Но ЭВМ может запоминать данные. Простейшим устройством, осуществляющим запоминание данных, является триггер. Изобретение его относится еще к 1918 г., и сде-

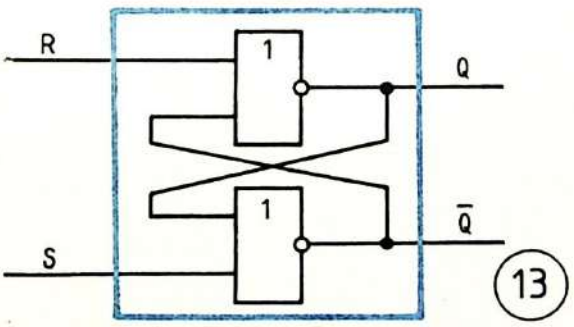
Таблица 20

Вход (двоичный код)				Выход (сигнал, соответствующий десятичной цифре 0 1 2 3 4 5 6 7 8 9)									
X4	X3	X2	X1	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

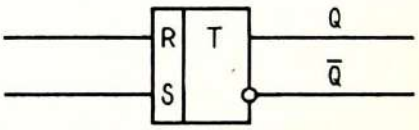
лано оно М. А. Бонч-Бруевичем, руководителем созданной по декрету В. И. Ленина Нижегородской лаборатории связи.

Триггер имеет два устойчивых состояния, в которые поочередно переходит под воздействием входных сигналов при записи информации. Существует множество типов триггеров. Один из них, так называемый *RS*-триггер, построенный на двух элементах ИЛИ—НЕ, приведен на рисунке 13. Выход одного элемента подключен к входу другого; это обеспечивает триггеру два устойчивых состояния и, соответственно, возможность хранения информации в виде 1 или 0. Вход R называют входом установки триггера в нулевое состояние, а вход S — в единичное. Триггер имеет два выхода. Из них Q — это прямой выход, а $\overline{Q}$ — инверсный, так как на $\overline{Q}$ потенциал всегда соответствует состоянию, которое противоположно состоянию выхода Q. Таблица 21 соответствует таблице истинности триггера.

Предположим, что на выходах триггера уже есть определенная комбинация сигналов, а именно: $Q=0$, а $\overline{Q}=1$. Тогда при подаче на входы нулей состояние триггера не изменится (см. 1-ю строку табл. 21), т. е. он «помнит» записанный ранее 0 ($Q=0$). Так же триггер хранит и единицу — это видно по 2-й строке таблицы. Если на вход S подавать 1, то, независимо от предыдущего состояния, на выходе Q всегда будет 1 (см. 3-ю и 4-ю строки табл. 21). Аналогично, если подавать 1 на вход R, то на выходе Q будет 0, что следует из последних двух строк таблицы.



13



14

Таблица 21

Вход			Выход			
			состояние до установки		состояние после установки	
№ п/п	R	S	Q	$\bar{Q}$	Q	$\bar{Q}$
1	0	0	0	1	0	1
2	0	0	1	0	1	0
3	0	1	0	1	1	0
4	0	1	1	0	1	0
5	1	0	0	1	0	1
6	1	0	1	0	0	1

В этой таблице отсутствуют две строки, соответствующие случаям, когда оба входа равны 1. Дело в том, что тогда состояние выходов триггера будет неопределенным и зависеть от электрических и временны́х характеристик самого устройства. Поэтому такая комбинация на входах является запрещенной.

На схемах триггер обозначают прямоугольником, помеченным буквой T (см. рис. 14).

Триггер помнит только один двоичный разряд. Для хранения многоразрядных чисел *триггеры объединяют в устройство,* называемое *регистром.* Регистр — важнейший элемент ЭВМ, и разговор о нем пойдет особый, но немного позже.

Приведенные в этом параграфе логические схемы как отдельные устройства в современной вычислительной технике ты уже не встретишь. Элементной базой ЭВМ стали интегральные схемы, которые в одном корпусе имеют набор устройств, но так как логика построения таких устройств все та же, то этот параграф дает представление и о их работе.

2.3. НА ПУТИ К ПРОГРАММИРОВАНИЮ

2.3.1. «Дети, соберитесь...»

Как ты думаешь, что такое с точки зрения математики содержимое хозяйственной сумки твоей бабушки? Это *множество* продуктов. А коллектив твоего класса? Это *множество* ребят.

А замечал ли ты, что очень часто мы переносим внимание с отдельных предметов на их скопления, понимая эти скопления как что-то целое? И ты, вероятно, уже пришел к выводу, что *множество* — это любое скопление определенных предметов или объектов, воспринимаемое как единое целое. Так оно и есть. А предметы, составляющие множество, называются *элементами множества* или его *членами.* Принадлежность элемента x множеству A обозначается как $x \in A$. Если же x не есть элемент множества A, то это записывается в виде $x \notin A$. Записью $x_1, x_2, ..., x_n \in A$ пользуются для сокращения последовательности $x_1 \in A$, $x_2 \in A, ..., x_n \in A$. Если множество не содержит ни одного элемента, то оно называется *пустым* и обозначается специальным знаком $\varnothing$.

Множество может быть задано путем перечисления его элементов, заключенных в фигурные скобки, как, например:

коллектив = { *Вицин, Моргунов, Никулин* }.

В частности, множество может состоять и из одного элемента, тогда оно называется *единичным множеством*.

Любой *коллектив* класса состоит из учеников-отличников и учеников, которые успевают немного хуже. Значит, *множество* может состоять из *нескольких множеств*, а *входящие* множества называются *подмножествами* другого множества, например *пустое множество* есть подмножество любого множества. Если A — множество отличников, а B — коллектив класса, то говорят, что A включено в B, и обозначают так: $A \subset B$.

Два множества равны, если они состоят из одних и тех же элементов. Например, если $A = \{1, 2, 3,\}$, а $B = \{2, 1, 3,\}$, то $A = B$.

2.3.2. «...и станьте парами!»

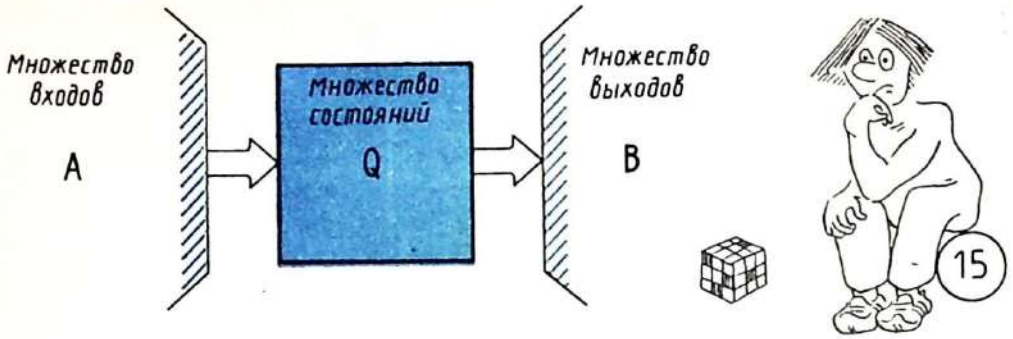
Читатель, ты, наверное, помнишь призыв: «Дети, станьте парами!» А сейчас задумайся над тем, что представляете ты и твой сосед по парте? Пару, но *упорядоченную*. А дедушка и бабушка? По-видимому, то же самое — *упорядоченную пару*. Такой парой могут быть и два предмета, например стол и стул. Выходит, что упорядоченная пара — совокупность двух предметов, расположенных в определенном порядке. А если это совокупность, то, значит, *упорядоченная пара* — это множество, точнее, *двухэлементное* множество. Упорядоченная пара символически обозначается через $\langle x, y \rangle$. Первый элемент пары x называют *первой координатой*, а второй — y — *второй координатой*.

В упорядоченных парах можно определить упорядоченные тройки, четверки и вообще упорядоченные n-ки. Упорядоченная тройка предметов, обозначаемая как $\langle x, y, z \rangle$, определяется как упорядоченная пара $\langle \langle x, y, \rangle z \rangle$. Упорядоченная n-ка предметов $x_1, x_2, \ldots, x_n$, обозначаемая как $\langle x_1, x_2, \ldots, x_n \rangle$, есть упорядоченная пара $\langle \langle x_1, x_2, \ldots, x_{n-1} \rangle, x_n \rangle$. Синонимом упорядоченной n-ки является n-мерный вектор, или *кортеж*.

2.3.3. ЭВМ — это автомат...

Электронные вычислительные устройства и их программное обеспечение характеризуются огромным числом элементов и их комбинаций. Это приводит к возникновению сложных задач выбора вариантов структуры ЭВМ при ее проектировании. Для преодоления этой сложности потребовался математический аппарат, позволяющий строить модели машин. Таким аппаратом стала *теория автоматов*.

Простейшая модель машины имеет вид, показанный на рисунке 15. На входе машины действует множество A входных сигналов, а на ее выходе множество B выходых сигналов. Под воздействием входных сигналов машина переходит из одного состояния в другое; множество таких состояний обозначено Q. Эта машина и есть автомат, определяющими элементами которого являются упорядоченные пары упорядоченных пар:



$$M = \{((\text{состояние, вход}), (\text{новое состояние, выход}))\}.$$

Элементы автомата можно представить и так:

$$M = \{((q_i,\ a_i),\ (q_j,\ b_j))\}.$$

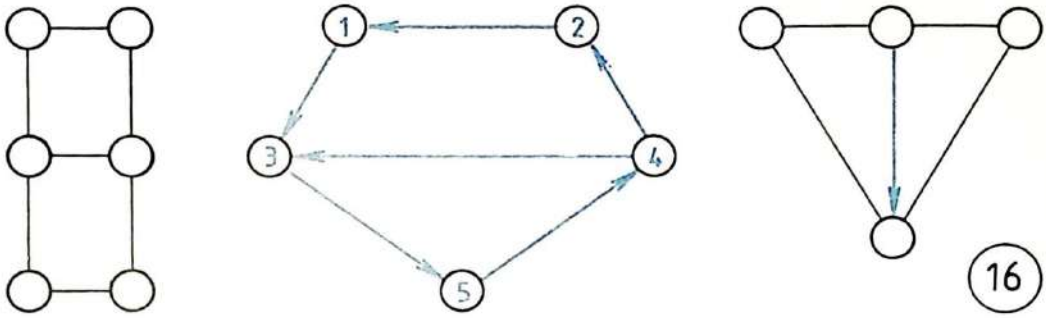
Таким образом, автомат может быть представлен списком множеств состояний, т. е. входов и выходов. Тогда один из возможных способов выявления внутренней структуры машины состоит в определении *передаточной функции* ее, т. е. в составлении математических соотношений между входами и выходами по каждому состоянию из списка состояний.

2.3.4. ... А программа — это граф

Мощным инструментом человеческого познания, наряду с естественным языком, а также письменностью и операциями с числами, является графика. Как часто нам приходится браться за карандаш и бумагу; последнюю мы покрываем линиями и точками в надежде объяснить то, что не удается выразить словами и цифрами. Поэтому интерес математики к столь универсальному средству не случаен. Оказалось, что конфигурацию из точек и линий можно определить как *абстрактное математическое понятие,* если не обращать внимание на то, какими являются линии,— прямые они или кривые, длинные или короткие. Важно лишь то, что они соединяют две точки. Такое понятие получило название *граф,* а было оно впервые предложено в 1936 г. Д. Кенигом — венгерским математиком.

Раньше применялись различные названия: карта, схема, лабиринт и др. Начало же *современной теории графов* заложил в 1937 г. американец Пойа.

Началась история графов еще раньше, со статьи Л. Эйлера, написанной в 1736 г. В этой статье Эйлер впервые предлагал графическую задачу, суть которой заключалась в том, что необходимо было «на бумаге» обойти семь мостов Кенигсберга лишь один раз, т. е. нарисовать маршрут. Интерес к графам возродился снова в XIX в. благодаря исследованиям моделей кристаллов, электрических и транспортных сетей. И наконец графы пришли в теорию алгоритмов. Первые понятия графовых схем алгоритмов и их преобразований для программирования ввели в 1956 г. советские математики А. А. Ляпунов и Ю. И. Янов.

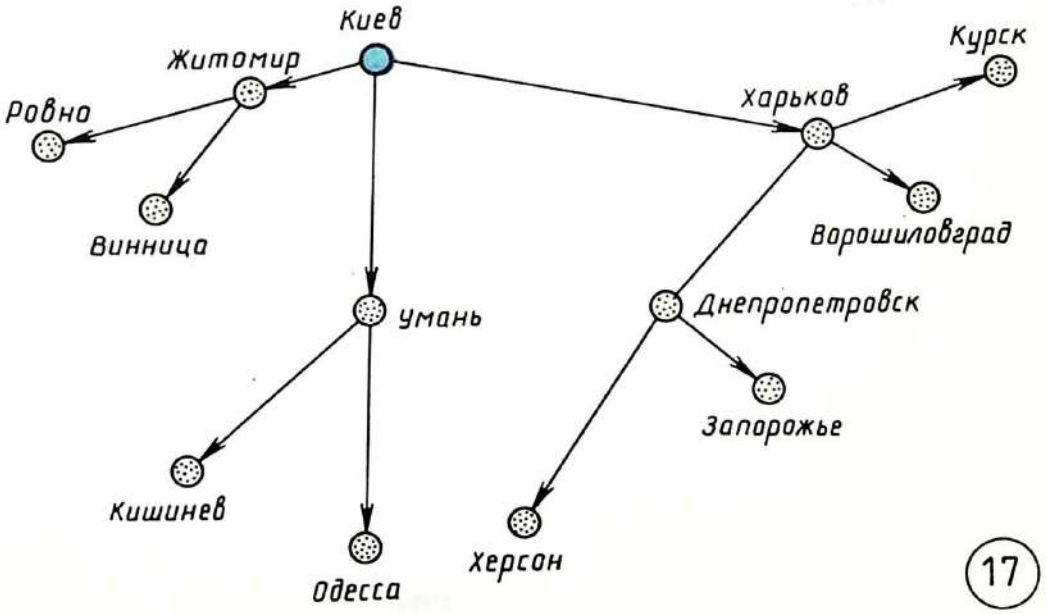


16

Каково же математическое определение графа? Пусть есть множество V точек. Тогда граф $G = G(V)$ — некоторое множество упорядоченных пар $E = \langle a, b \rangle$; $a, b \in V$, причем каждая пара указывает, какие элементы (точки) a и b соединены между собой. Такая пара называется *ребром графа,* а элементы a и b — *концевыми точками* или *вершинами.*

На схемах обычно вершины обозначаются маленькими кружочками, а *ребра* — непрерывными прямыми или кривыми линиями (рис. 16).

Существует несколько разновидностей графов. Для нас интересен, пожалуй, будет один тип графа, который имеет название *дерево.* У дерева каждое ребро имеет *ориентацию,* т. е. стрелку, показывающую, из какой вершины ребро выходит и в какую входит. Но особенностью его является то, что во все вершины, кроме одной, входит по одному ребру. Вершина, в которую не входят ребра, называется *корнем дерева,* а концевые вершины — *листьями.* Путь от корня к листьям называется *ветвью.* Обычно графическое дерево изображают корнем вверх. На рисунке 17 приведено дерево автомобильных дорог из Киева в другие города страны. Вершина, где находится Киев, будет корнем дерева,



17

а вершины, соответствующие городам Ровно, Винница, Кишинев, Одесса, Херсон, Запорожье, Ворошиловград, Курск, являются его листьями. Путь ⟨Киев, Харьков, Днепропетровск, Херсон⟩ есть ветвь дерева.

Графы типа деревьев применяются не только для географических описаний. Они широко используются при анализе многих задач в электронике и в программировании. Изменение уровней потенциалов в логической схеме или вычислительный процесс могут быть представлены в виде дерева. Корень дерева соответствует начальному состоянию системы, описываемой логической схемой или языком программирования, а вершины — это состояния, в которые система переходит в результате выполнения различных операций; листья — конечные состояния системы.

2.3.5. Язык и грамматика

В процессе общения людей возник язык, которым мы пользуемся в своей обыденной речи и на котором пишутся книги. Но людям пришлось создать и искусственные языки, например логико-математические, применяющиеся в науке, языки международного общения — эсперанто, воляпюк и др. Особый интерес для нас представляет группа так называемых формальных языков, в которую входят логико-математические языки, языки описания алгоритмов и языки программирования, создаваемые с целью построения точных и однозначных формулировок. По словам советского ученого Н. А. Криницкого, эти языки должны позволять «выводить из аксиом следствия не путем рассуждений, а с помощью операций, преобразующих символьные конструкции независимо от их смысла». Когда мы говорим о формальных языках, то имеем в виду универсальность и независимость выражений в этих языках от смысла описываемых процессов.

Для создания и изучения подобных языков необходим математический аппарат. Вполне естественным явилось то, что выбор пал на теорию множеств. Действительно, язык содержит множество символов и знаков, объединенных в множества выражений. Основой языка является алфавит. *Алфавитом* называется произвольное множество попарно различимых символов, например {а; б, в, ..., я} или {яблоко, слива, вишня}. Алфавит принято обозначать через V. Тогда строкой длиной m $(m \geqq 0)$ над алфавитом V называется *m-компонентный кортеж над множеством V*. Строка обозначается символом σ. Строка с $m = 0$ называется *пустой* и обозначается Λ. Строка записывается как $\langle x_1, x_2, \ldots, x_m \rangle$, или $x_1, x_2, \ldots, x_m$. Множество всех строк над V обозначается V^*, но оно редко применяется в конкретных задачах полностью. Обычно участвуют какие-то характерные *подмножества* этого множества. Например, если мы начнем составлять слова из 4 букв, то сразу обнаружим, что слова, состоящие только из согласных, не имеют смысла. Таким образом, свод правил, определяющий подмножества V^*, называется *грамматикой*.

Грамматика естественных языков — русского и иностранных — долго

изучается в школьном курсе. Это и понятно, так как естественные языки очень сложны; в них отражается сущность человеческого разума. Попытки описать эти языки с помощью некоторого формального аппарата сегодня еще далеки от успеха. Но в сфере создания искусственных языков, особенно в приложениях к вычислительной технике, прогресс значительно заметнее. По оценке академика А. П. Ершова, «число существующих языков программирования перевалило за несколько тысяч и, наверное, скоро будет сравнимо с количеством человеческих языков». Как быстро создавались и распространялись языки программирования, можно представить, если учесть, что к 1963 г. использовалось около 30 языков, к 1969 г.— 120, а приведенная оценка Ершова относится к середине 80-х гг.

Идеи программирования были заложены в середине прошлого века. В 1834 г. англичанин Чарльз Бэббидж после десятилетних поисков разработал так называемую аналитическую машину. Толчком для этого изобретения послужил успех француза Ж. Жаккара, впервые применившего перфокарты для автоматизации работы ткацкого станка. Позже принцип действия машины Жаккара применяли в механических пианино, телеграфных аппаратах. Но настоящая революционность этого изобретения связана все-таки с вычислительной техникой. На протяжении 150 лет перфокарты были основными носителями информации для различных вычислительных машин (только в 80-х гг. нашего века было объявлено о прекращении выпуска перфокарт). Но наиболее замечательным в этом изобретении было то, что впервые для управления каким-либо устройством использовались комбинации двух состояний: *отверстие* в карте или его *отсутствие*. Этот принцип управления был применен Бэббиджем в его машине, идеи которой стали фундаментом для дальнейшего развития ЭВМ. Поэтому иногда Ч. Бэббиджа называют «дедушкой» современных ЭВМ.

При участии Бэббиджа его ученица и помощница Ада Лавлейс — дочь поэта Байрона составила первые программы для решения систем двух уравнений и вычислений чисел Бернулли. «Аналитическая машина ткет алгебраические узоры» — так возвышенно воспринимала дочь поэта процесс вычислений. В 1843 г. она опубликовала конспект лекций своего учителя, где в виде комментария ввела основные принципы программирования аналитической машины, некоторые из которых используются и до сих пор. Но сам термин «программа» определился лишь 100 лет спустя.

Первую электронную машину *ENIAC (Electronic Numerial Integrator and Calculator),* созданную инженером Д. Эккертом и физиком Д. Моучли в электротехническом колледже при Пенсильванском университете, эффективно использовать не удалось из-за большого числа ручных операций по ее управлению.

В 1945 г. американец Джон фон Нейман предложил *команды управления* и *данные* хранить в *машинной памяти,* тем самым создав более благоприятные условия для работы с машиной. Эти хранимые в быстродействующей памяти команды и получили название *программы.* Уста-

новленные Нейманом принципы определили основы ЭВМ на несколько десятилетий вперед, и время их еще не прошло. За этот неоценимый вклад Д. фон Неймана называют «отцом современных ЭВМ».

2.4. ВНАЧАЛЕ – АЛГОРИТМ

2.4.1. Разрешимость задач

С давних времен стихийно складывались и применялись в математике простейшие *алгоритмы,* постепенно ставшие одним из основных ее понятий. Смысл этого слова понимается как точное предписание к выполнению в заданном порядке некоторой системы операций для решения задач определенного типа. Например, правила четырех арифметических действий представляют собой цепочку элементарных операций, носящих формальный характер. Рассмотрим сложение:

1. Вести сложение справа налево.
2. Записать сумму двух складываемых разрядов.
3. Пометить перенос (если он есть) над цифрой слева.
4. Записать сумму двух следующих разрядов и переноса из предыдущей операции (если он есть).
5. Далее выполнить пункты 3 и 4, пока не учтем все разряды складываемых чисел.

Аналогичный перечень элементарных операций можно привести и для других арифметических действий. (Известен алгоритм Евклида для нахождения наибольшего общего делителя двух натуральных чисел, состоящий из таких элементарных операций, как вычитание двух чисел, их сравнение и перестановка местами.) Использование указанных алгоритмов с различными исходными данными все равно приведет к определенному конечному результату. Это замечательное их свойство имеет место не только в математике, но и при выполнении производственных операций, приготовлении пищи, в процессе игры, т. е. везде, где требуются логические построения. Таким образом, сформировалось следующее определение: *алгоритм – это последовательность математических, логических или вместе взятых операций, отличающаяся детерминированностью, массовостью, направленностью и приводящая к решению всех задач данного класса за конечное число шагов.*

Детерминированность (от латинского *determinare* – определять), или *определенность алгоритма* означает: если метод вычисления сообщить другому лицу в виде указаний о действиях на отдельных этапах вычислений, то это лицо обязательно алгоритм выполнит. Когда мы говорим о каких-то машинальных действиях, то имеем в виду именно *определенность* тех правил, по которым выполняем эти действия.

Массовость выражается в том, что алгоритм как единое предписание, определяющее вычислительный процесс, может быть начат с множества различных исходных данных, но всегда приведет вычислителя к конечному результату, т. е. с помощью алгоритма можно решать не одну задачу, а серию однотипных задач, что называется *разрешимостью* таких задач.

Утверждение о том, что алгоритм всегда ведет к получению результата, определяет его *направленность* (результативность).

И еще одно свойство алгоритма — его *дискретность*. Это свойство отражается в приведенном выше определении в словах: *за конечное число шагов. Шаги* — это и есть те элементарные операции, из которых строится алгоритмическая последовательность. Именно благодаря этому свойству алгоритм может быть реализован на ЭВМ.

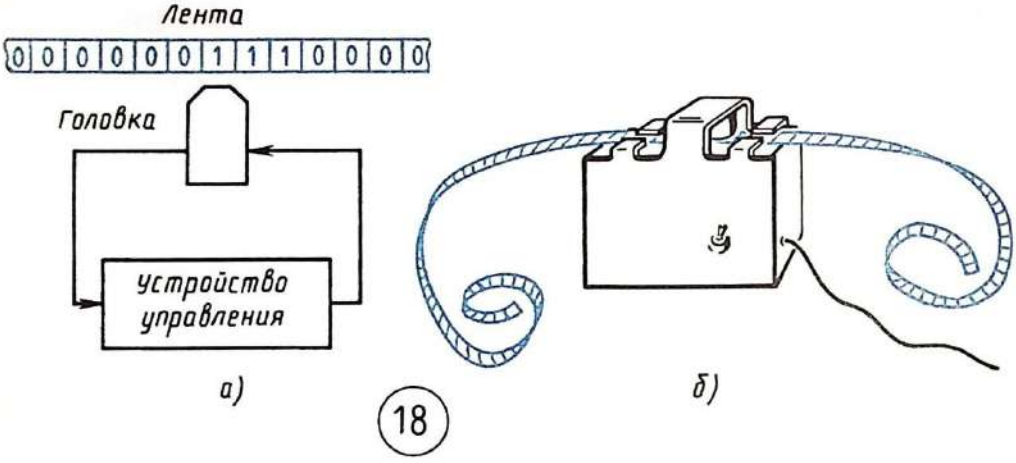
2.4.2. ... и их неразрешимость

Развитие науки и техники выдвигает все больше новых задач, образующих группы однотипных задач или целые серии. Перед математиками встала проблема оценки разрешимости этих серий задач. Важнейшее свойство результативности алгоритма привело к утверждению, что серия задач определенного типа считается решенной, если для ее решения есть алгоритм.

Однако в результате многочисленных попыток разработки всевозможных алгоритмов для решения различных математических и логических задач стало ясно, что далеко не для всякой серии задач возможно найти алгоритм решения.

Хорошо известны три древние неразрешимые геометрические задачи: о квадратуре круга, о трисекции угла и об удвоении куба с помощью циркуля и линейки. Здесь доказательство неразрешимости очевидно, так как мы точно знаем алгоритм построения с помощью циркуля и линейки. Но это алгоритм специального вида, а для общего понятия алгоритма его точного определения долгое время не имели. Поэтому выработка такого определения стала одновременно поиском точного доказательства неразрешимости, так как такое доказательство осуществимо лишь тогда, когда имеется точное определение понятия «алгоритм». Ведь в противном случае несуществование одного понятия доказывалось бы с помощью несуществующего другого, что заведомо абсурдно. Этот поиск был отнесен к числу важнейших задач математики. Трудность заключалась еще и в том, что искалось точное определение того понятия, которое фактически уже имелось с давних времен, хотя и в расплывчатом виде.

Наиболее интенсивные поиски в этой области начались в 30-х гг. XX в. Они основывались на различных технических и логических соображениях, что привело к выработке нескольких определений. Для нас особый интерес представляет то определение алгоритма, которое раскрывается через процессы, протекающие в вычислительной машине. Впервые максимально простую *схему* этих процессов, но вместе с тем настолько точную, что она стала служить предметом математических исследований, предложил английский математик Тьюринг в 1937 г. Эта схема в дальнейшем получила название *машина Тьюринга*. Таким образом, Тьюринг уточнил интуитивное понятие алгоритма. Часто машину Тьюринга пытаются изобразить просто в виде схемы, например как на рисунке 18,*а*, или в виде некоторого электромеханического аппарата (рис. 18,*б*), хотя машина Тьюринга — это математиче-



ская модель, а не машина в обычном смысле этого слова. Такие изображения применяются исключительно для обеспечения наглядности. В своем предложении Тьюринг исходил из общей идеи уподобления работы машины работе человека-вычислителя, оперирующего в соответствии с некоторым строгим предписанием.

2.4.3. Что же такое алгоритм?

В машине Тьюринга предполагаются три основные части:

1. Бумажная лента, поделенная на ячейки.
2. Устройство управления.
3. Головка.

Каждой такой машине соответствуют два конечных алфавита: алфавит *внешних символов* $A = \{a_0, a_1, ..., a_n\}$ и алфавит *внутренних состояний* $Q = \{q_0, q_1, ..., q_m\}$. В любой момент времени в каждой ячейке ленты бывает записана одна буква алфавита из A (считают, что a_0 — это пустая буква, т. е. отсутствие записи в ячейке интерпретируется как запись буквы a_0). Лента может двигаться вправо и влево, при этом одна ячейка точно находится под головкой, т. е. обозревается устройством. На устройстве управления имеется кнопка, при нажатии которой оно принимает исходное состояние q_0 и начинает работать.

Совокупность сведений о состоянии устройства управления и записи на ленте называется *конфигурацией машины Тьюринга*. Работа машины Тьюринга состоит из *тактов*, в каждом из которых производится *преобразование* конфигураций.

Такое *преобразование* называется *командой машины Тьюринга*, а совокупность команд — *программой*.

Работа машины Тьюринга по программе продолжается неограниченно, с какой бы конфигурации она ни начиналась. Однако можно ввести некоторые правила остановки этого процесса. Конфигурация, в которой машина останавливается, называется *заключительной*.

Кроме машины Тьюринга были предложены и другие способы уточнения интуитивного понятия алгоритма, например *нормальные алго-*

ритмы, введенные советским математиком Марковым. Доказано, что машина Тьюринга для функции существует только в том случае, если существует марковский алгоритм для вычисления этой функции. Таким образом выяснилось, что все эти определения равносильны между собой, т. е. определяют одно и то же понятие — понятие алгоритма.

2.4.4. Методы описания алгоритма

Алгоритмический язык. Классические алгоритмические системы, такие, как нормальные марковские алгоритмы и машина Тьюринга, оказались практически непригодными для описания разнообразных задач. Формулирование математической модели производственного процесса, экономического управления, научной или инженерной проблемы, не говоря уже об алгоритмизации творческих процессов искусства, представляет нелегкую работу. А запись сложного алгоритма с помощью фундаментальных систем теории является сама по себе непростым делом. В результате проблема автоматизации, неразрывно связанная с разработкой алгоритмов, привела к появлению множества *алгоритмических языков,* позволяющих, может быть, и не столь в лаконичной, сколько в точной, единообразной форме описать объект задачи.

Алгоритмический язык — это система обозначений формальной записи алгоритмов, предназначенных для анализа их исполнителем-человеком, а затем и для реализации на машине; для этого они переводятся на конкретный язык программирования. Поэтому можно сказать, что *система обозначений* присуща в той или иной форме языкам программирования. Фундаментальным понятием всех этих языков является понятие *оператор,* сформулированное советским математиком А. А. Ляпуновым как *описание однородных этапов изучаемого процесса.* В общем виде имеются операторы четырех типов:

1. Действующие; определяют изменения в объекте.

2. Логические; описывают условия, от которых зависит направление процесса.

3. Варьирующие; определяют изменения вспомогательных величин, называемых *параметрами.*

4. Операторы перехода; определяют порядок выполнения операторов первых трех типов. Кроме операторов, применяются знаки *начала* и *конца* процесса.

Фактически оператор предписывает исполнителю алгоритма какое-то указание, каждое из которых получило впоследствии название *команда,* так как в повелительном наклонении излагает определенное предложение. Для удобства *команды* стали обозначать одним полным или сокращенным словом, а также определенным значком. Эти обозначения команд получили название *служебные слова* и *служебные символы.*

Составим в качестве примера описание алгоритма для нахождения суммы пяти чисел в терминах алгоритмического языка. (Этот язык изучается в школе в курсе основ информатики.) Здесь же выделим применяемые операторы (команды) и знаки:

```
алг сумма (цел S, цел N, таб M[1:5])
    арг N
    рез S
        S:=0                    – действующий
        N:=5                    – действующий
нач цел i                       – знак начала процесса
        i:=0                    – действующий
    пока i≦N                    – логический
    нц                          – перехода
      i:=i+1                    – действующий
      S:=S+M[i]                 – действующий
    кц                          – перехода
кон                             – знак конца процесса
```

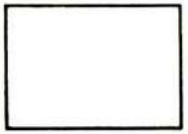
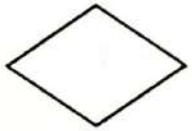
Действующий оператор содержит знак «:=», который является знаком присваивания. Он означает, что переменная величина, находящаяся слева от знака, в результате действия присваивания принимает значение числа или выражения, находящегося справа от знака. Знак «:=» соответствует знаку «=». Кроме этого, он несет алгоритмический смысл, т. е. указывает, что необходимо произвести некоторое действие, в то время как знак «=» лишь констатирует факт равенства и никаких действий не определяет.

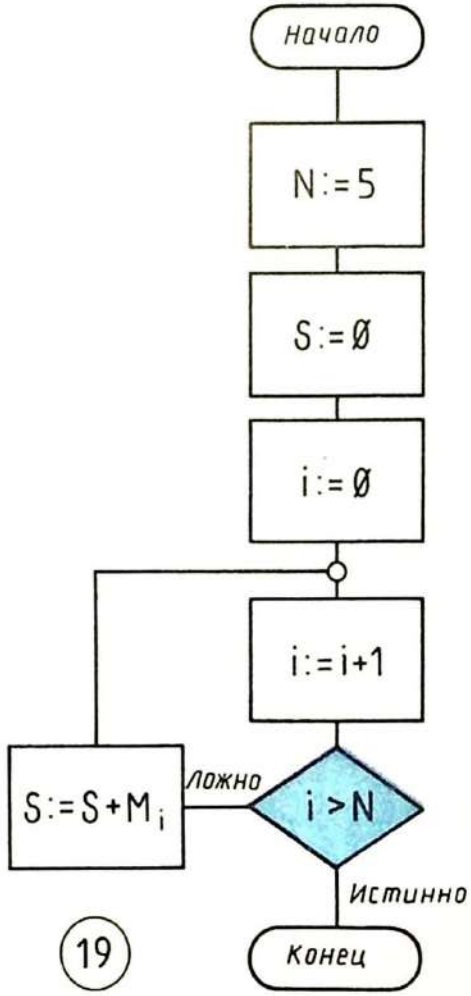
Графика. При всех своих достоинствах алгоритмические языки обладают существенным недостатком – они недостаточно наглядны. Действительно, описание такого простого процесса, как вычисление суммы пяти чисел, содержит более десятка строк, содержащих символы и слова. Подобные описания сложных математических задач или процессов управления занимают сотни страниц, сплошь заполненных этими строками. Вести изучение задачи, выявлять ошибки ее постановки и закономерности алгоритма решения, определять возможность оптимизации процесса обработки на таком материале весьма затруднительно. Сделать описание алгоритма более наглядным помогает его графическое изображение, которое называется *структурной схемой алгоритма*.

Элементами такой схемы являются графические символы: блоки и линии, соединяющие их. Существуют международные и отечественные стандарты (ГОСТ 19. 003-80), которые определяют форму и размеры блоков и линий, а также дополнительных графических символов. В таблице 22 приведены некоторые основные элементы, используемые в структурных схемах алгоритмов.

Структурная схема является ориентированным графом, у которого графические обозначения элементов соответствуют вершинам графа, а линии потока – его ребрам (см. рис. 16). Для упрощения структурной схемы договорились не ставить стрелочки на линиях потока, идущих сверху вниз и слева направо. Если же линия потока направлена снизу вверх или справа налево, то она обязательно должна завершаться стрелочкой.

Таблица 22

Графическое обозначение	Наименование элемента	Выполняемая функция
	Процесс	Определяет изменение в объекте (аналог действующего и варьирующего операторов)
	Решение	Описывает условие, от которого зависит направление процесса (аналог условного оператора и оператора перехода)
	Линия потока	Указывает последовательность выполнения процесса
	Соединитель	Указывает связь между линиями потока
	Пуск-останов	Начало или конец процесса



Используя приведенные пять элементов, алгоритм вычисления суммы пяти чисел изобразится так, как показано на рисунке 19.

Не забудь о структуре! Принципиальным отличием алгоритмических языков, в том числе и языка структурных схем, от естественного языка является наличие в первых жестких *структур*, логично, надежно и целесообразно построенных из нескольких блоков. Эти структуры делятся на три группы:

1. **Последовательные структуры:**
   — последовательность;
   — индексная последовательность.
2. **Разветвляющиеся структуры:**
   — ветвление двойное (*если — то — иначе*);
   — ветвление (*если — то*);
   — выбор.
3. **Циклические структуры:**
   — *пока — выполнить;*
   — *выполнить — до;*
   — *выполнить — пока — выполнить.*

При использовании *последовательных структур* операции записываются последовательно одна за другой, что представляет собой определенную *последовательность*. Фрагмент схемы алгоритма вычисления суммы пяти чисел, приведенный ниже на рисунке 20, является такой структурой. Конечно, последовательности могут состоять не только из трех операций, а и из большего их числа. Например, процесс нахождения суммы пяти чисел можно представить рисунком 21. Но гораздо эффективнее для таких случаев применять структуры типа с *индексной последовательностью,* которая показана на рисунке 22. Преимущество последней структуры состоит в том, что для использования этого алгоритма под вычисление суммы *десяти* чисел достаточно изменить один оператор: N := 10, а в остальном алгоритм не меняется. Здесь параметр *i* называется *индексом* последовательности суммируемых чисел и служит указателем числа, которое участвует в операции на данном шаге алгоритма.

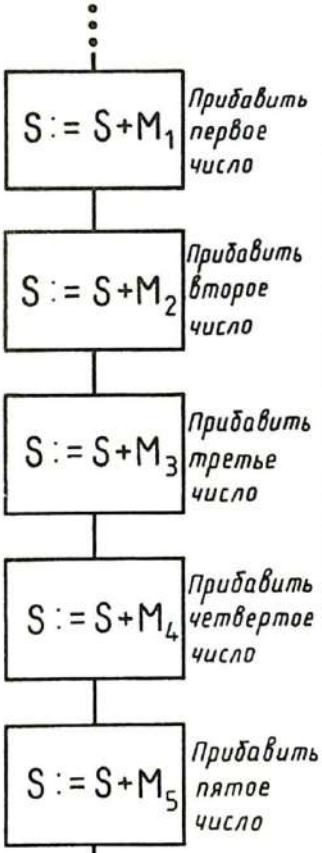
Разветвляющиеся структуры соответствуют реализации логических функций, неизбежно встречающихся в любом алгоритме, число которых определяет его сложность. Способность ЭВМ выполнять неограниченное число логических операций является замечательным ее свойством; ведь для человека решение логических задач является едва ли не самым трудным делом, а во многих случаях и невозможным.

Структура с *двойным ветвлением (если — то — иначе)* представлена на рисунке 23, где операторы *то — части* выполняются тогда, когда условие, следующее за *если,* принимает значение *истинно,* а операторы *иначе — части* выполняются в случае значения условия *ложно.*

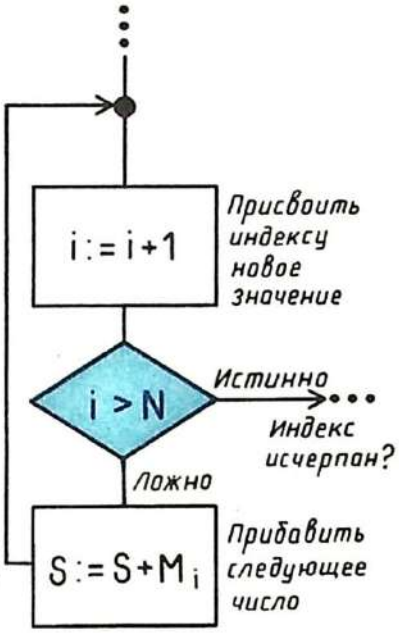
Частным случаем двойного ветвления является структура типа *ветвление (если — то),* в которой при значении *ложно* условия *если* никакая операция не выполняется (рис. 24).


N := 5
Первая операция
S := Ø
Вторая операция
i := Ø
Третья операция

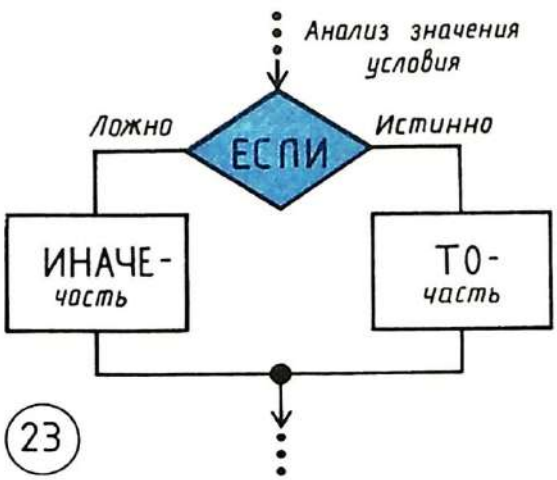
20


S := S+M1
Прибавить первое число
S := S+M2
Прибавить второе число
S := S+M3
Прибавить третье число
S := S+M4
Прибавить четвертое число
S := S+M5
Прибавить пятое число

21


i := i+1
Присвоить индексу новое значение
i > N
Истинно
Индекс исчерпан?
Ложно
S := S+Mi
Прибавить следующее число

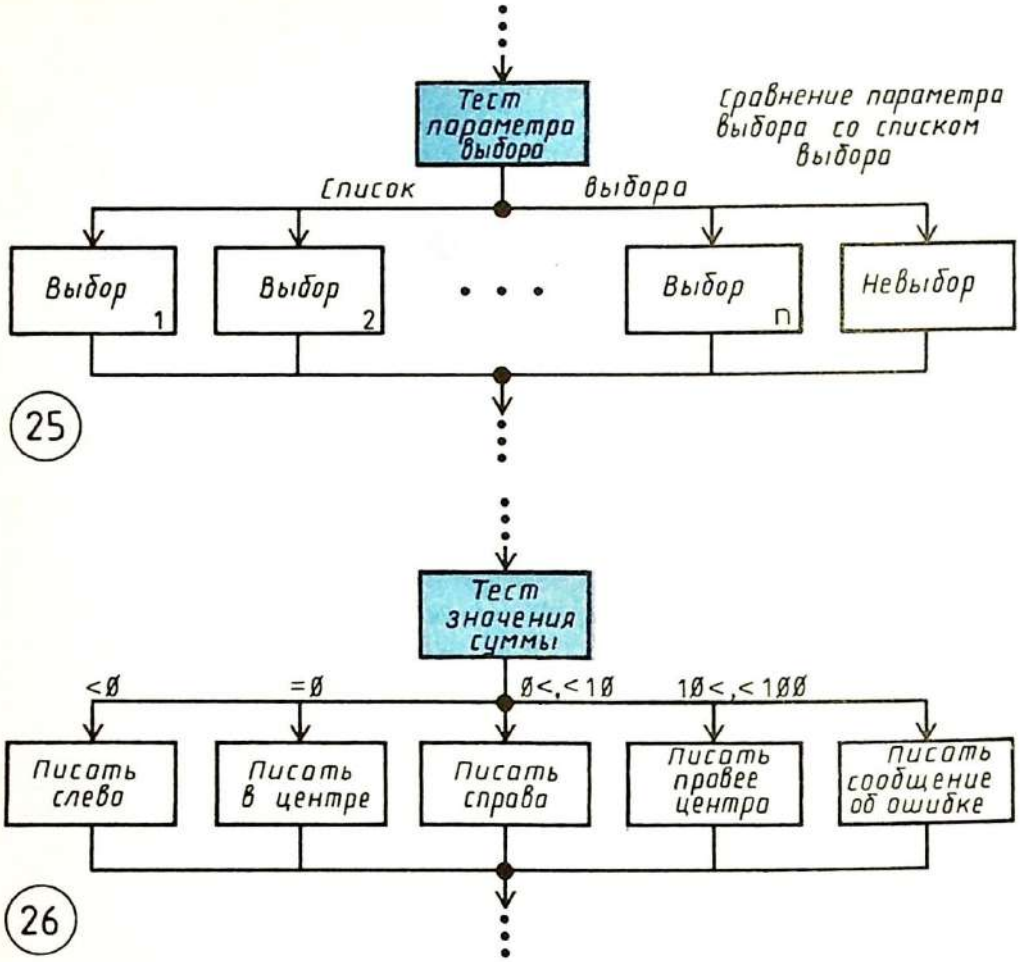
22


Анализ значения условия
Ложно
ЕСЛИ
Истинно
ИНАЧЕ-часть
ТО-часть

23


Анализ значения условия
ЕСЛИ
Истинно
ТО-часть

24



Аналогично строится структура *если — иначе*, в которой не выполняются действия, когда условие *если* принимает значение *истинно*. Подобную структуру мы применили в алгоритме вычисления суммы как составляющую индексной последовательности. В этом примере структура *если — иначе* выбрана лишь с целью некоторого упрощения. В практических задачах рекомендуется использовать структуры *если — то* как соглашение, облегчающее дальнейшее изучение алгоритма и его корректировку.

Разветвляющиеся структуры типа *выбор* показаны на рисунке 25. При выполнении фрагмента алгоритма, приведенного на этом рисунке, просматривается в определенной последовательности *список выбора* с целью сравнения *параметра выбора* с соответствующей частью списка. Если в результате этого сравнения определяется *часть списка*, соответствующая значению параметра, то она и выполняется далее. При отсутствии такой части выполняется *часть невыбор*. Для примера использования структуры типа *выбор* усложним наш алгоритм суммирования пяти чисел следующим требованием.

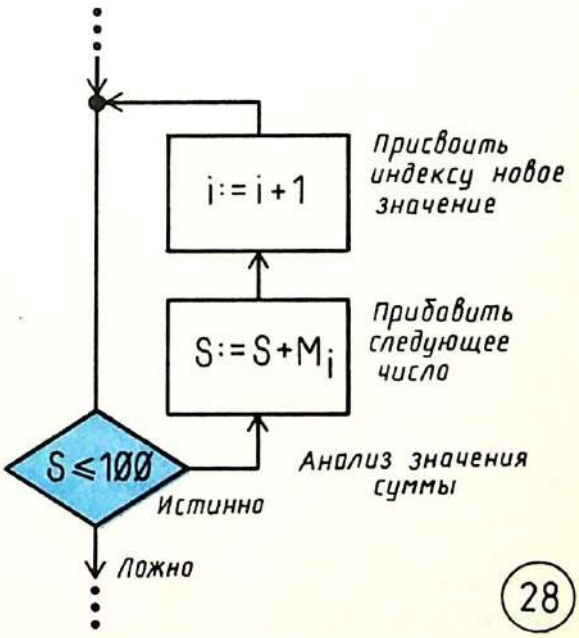
Если полученная сумма отрицательная, то необходимо написать ее значение на экране дисплея слева, если сумма равна нулю — в центре экрана. Если же сумма положительная, но меньше десяти, то ее значение пишут справа; если значение больше десяти, но меньше ста, то его записывают между центральной и правой позициями. Тогда этот фрагмент алгоритма будет иметь вид, показанный на рисунке 26. В этом случае предусматривается запись ошибки, если сумма принимает значение, не оговоренное условиями алгоритма.

Эту часть алгоритма можно реализовать и с помощью структур двойного ветвления *(если — то — иначе)*, но структура типа *выбор*, во-первых, более естественно описывает выбор одного из многих направлений логического ветвления алгоритма и, во-вторых, соответствует возможностям реальных вычислительных машин выполнять подобный выбор.

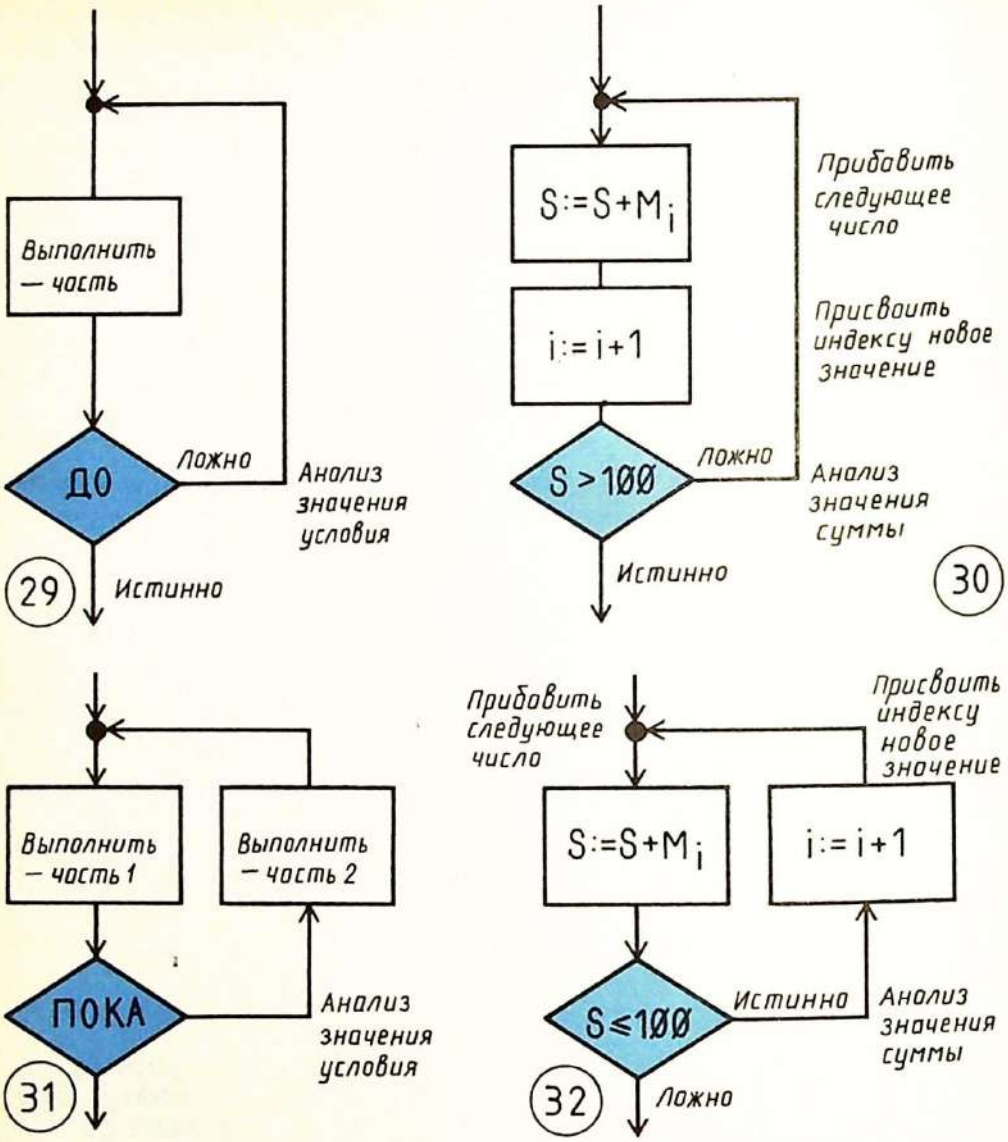
Циклические структуры едва ли не самые распространенные; они служат для того, чтобы заставить входящий в их состав действующий оператор выполняться несколько раз. Циклическая структура типа *пока — выполнить* имеет вид рисунка 27, где действующий оператор *выполнить — часть* выполняется нуль и более раз. Он выполняется тогда, когда условие *пока* имеет значение *истинно*. Разумеется, оператор *выполнить* на каждом шаге модифицирует условие *пока*. Например, если мы имеем последовательность не из пяти, а из большего количества чисел и хотим получить сумму больше ста, то этот фрагмент алгоритма примет вид, изображенный на рисунке 28.

Циклическая структура типа *выполнить — до*, приведенная на рисунке 29, позволяет выполнять действующий оператор до тех пор,

27



28



пока условие *до* не примет значение *истинно*. Здесь также на каждом шаге модифицируется значение *до*; в отличие от предыдущей структуры это действие выполняется до вычисления условия, т. е. эта структура применяется тогда, когда требуется по крайней мере одно выполнение действующего оператора, и в этом смысле она более предпочтительна для реализации предыдущего примера (рис. 30). Действительно, в циклической структуре вида *пока — выполнить* мы производим вначале лишнюю проверку ($S \leqq 100$), так как заведомо знаем, что на первом шаге $S = 0$.

Однако в примере алгоритма со структурой *выполнить — до* мы имеем тоже лишнее действие. Когда в результате выполнения оператора $S := S + M_i$ значение суммы превысит 100, выполнять опера-

тор $i = i + 1$ нет необходимости. Избавиться от этого недостатка поможет циклическая структура типа *выполнить — пока — выполнить* (рис. 31, 32).

Рассмотренные простые структуры позволяют записывать и читать алгоритмы решаемых задач; они полностью применимы для их переноса на языки программирования.

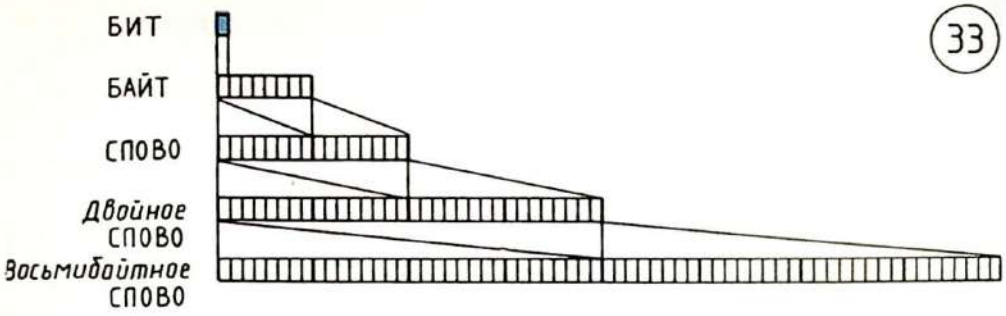
2. 5. ЧТО ЭВМ ПЕРЕРАБАТЫВАЕТ?

Всем знакома фраза: «ЭВМ перерабатывает информацию». Мы уже говорили, что информация — это определенные сведения о чем-либо. Как же машине удается перерабатывать эти сведения? Как мы закладываем в нее свои знания и идеи? Ответ на эти вопросы станет ясен, если ты вспомнишь, что информация имеет своих реальных *носителей*. Такими носителями являются *данные в формализованном виде,* содержащие те факты и идеи, которые мы хотим передать машине, например цифры и символы; они представляют собой *формализованный вид данных*. За фразой, с которой начался этот параграф, кроется скучная и монотонная работа, связанная с бесконечными арифметическими операциями над числами (буквами); тебе уже известно, что каждая буква — это число. И с этой работой ЭВМ справляется успешно, не уставая, не ошибаясь; ее операции выполняются с громадной скоростью. Как ни хороша вычислительная машина, но не будем забывать, что своими «успехами» она обязана программе, управляющей ее действиями. Эта программа пишется человеком. Ему необходимо предусмотреть то многообразие чисел и букв, которое будет «пропускать через себя» его программа. Для того чтобы облегчить эту работу, договорились в языках программирования применять некоторые *наборы типов данных*.

Каждый язык программирования может манипулировать с определенным таким набором. Забегая вперед, отметим, что языки бывают *низкого уровня* и *высокого*. С помощью языка *низкого уровня* программа взаимодействует непосредственно с аппаратурой ЭВМ и поэтому наиболее эффективно ее использует, но подготовка таких программ сопряжена с большими трудностями. Легче программировать на языках *высокого уровня,* составленных из близких к естественному языку фраз. Но такие языки удалены от аппаратных средств и менее эффективны. Соответственно и *данные* делятся на типы *низкого* и *высокого уровней представления* (см. рисунок цветной вклейки III).

2.5.1. Биты и байты

Данные низкого уровня представления можно назвать еще и машинными, так как с ними непосредственно оперирует аппаратура ЭВМ. С самым простым типом данных — *битом* — ты уже знаком. Это элементарная двоичная величина, которая может участвовать в машинных операциях. Чаще всего ЭВМ оперирует группой бит как одним целым. Одной из таких групп является байт (*byte*), содержащий восемь соседних бит. Байты используются для представления букв и других симво-



33

лов, а также двоичных чисел. Если для представления символьной информации байт является удобным типом, то для представления чисел возможности его ограничены из-за малой длины. Поэтому многие ЭВМ могут непосредственно оперировать с бóльшими группами — *словами (word)*, которые состоят из двух, четырех и восьми байт. Слова используются для представления чисел, и, естественно, слова бóльшей длины могут содержать большие или малые числа, но с бóльшей точностью. Давай посмотрим на рисунок 33. На нем изображены типы данных низкого уровня и взаимосвязь между ними.

Программист, пишущий на *языках высокого уровня*, не должен знать (чаще всего он действительно не знает) о существовании байтов и битов. Он с помощью своей программы собирается манипулировать некоторыми порциями данных — суммой пяти чисел, скоростью движения локомотива, диаметром трубы, именами своих товарищей, т. е. отдельными законченными по смыслу порциями данных. Обобщенным названием одной такой порции данных является *поле*. Оно характеризуется произвольной длиной и может быть представлено символами (код КОИ-7), двоичным числом или числом с плавающей точкой.

Однако обращение к каждому конкретному *полю* требует знания *типа данного*, находящегося в нем. Различают две группы *типов данных*. Элементарные типы называют *скалярами*, а наборы *агрегатами*. Со скалярами ты привык иметь дело в школе в математике, а примером агрегата может служить также таблица результатов лабораторной работы по физике. При алгоритмизации, а затем и программировании множества реальных задач применяются всевозможные типы данных. Но основой остаются те типы, с которыми мы продолжаем знакомиться.

2.5.2. Константы и переменные

Скаляры делятся на *константы* и *переменные*. Константа имеет известное перед началом операции значение и не меняет его после ее выполнения, т. е. остается постоянной. Переменная же принимает участие в выполнении операции со своим текущим значением, которое было определено в ходе предыдущей операции, и получает свое значение или меняет его в тех операциях, где она используется. В выражении, определяющем длину окружности

$$O = 2 \times 3{,}14 \times R,$$

величины 2 и 3,14 являются константами, а величина R — переменной. Так как значение переменной может быть различным в разные моменты времени, изображение переменной в выражении не может определяться ее значением. Поэтому изображениями переменных в выражениях служат буквы или совокупности букв. Такие изображения получили название *имена переменных*. С помощью имен могут изображаться и константы. Имена несут и естественный описательный смысл. Это выдерживается тогда, когда имя, придуманное программистом, соответствует математическому содержанию выражения. Без излишних комментариев ясно, что выражение

ДЛИНОКР = ДВА × ПИ × РАДИУС

служит для вычисления длины окружности, тогда как выражение

КАША = МАША × САША × ГЛАША

хотя и читается веселее, но не будет понято ни одним Мишей. Если в выражении используются константы, изображенные именами, их надо предварительно *определить* с помощью операции присвоения:

ДВА := 2, ПИ := 3,14.

Значение целочисленной переменной или константы представляет целое число в диапазоне $[-D, +D]$, где D является величиной, зависящей от разрядности *машинного слова*, содержащего эту величину.

Действительные или, как их еще называют, вещественные данные являются числами со знаком, которые могут содержать десятичную точку. Такие данные представляются в машине в виде чисел в формате с плавающей точкой.

Символьные, или литерные данные являются закодированными буквами, цифрами и другими знаками (+, −, =, ! и т. д.), используемыми для общения с ЭВМ.

Числовые и символьные типы данных тебе уже знакомы. Не будет для тебя открытием и суть *логического* типа данных. Такие данные принимают два значения — истина *(true)* и ложь *(false)*. Когда с помощью операций сравнения сопоставляют данные числового или символьного типов, результатом сравнения является величина логического типа. Например, если переменная DAT имеет целочисленный тип и DAT = 5, а также есть константа ONE = 1, то выражение DAT < ONE имеет значение *ложь*. При сравнении данных CHAR1 = A и CHAR2 = A выражение CHAR1 = CHAR2 имеет значение *истина*. Если есть переменные логического типа TRUE и FALSE, то эти выражения можно записать так:

(DAT меньше ONE) = FALSE

(CHAR1 равно CHAR2) = TRUE

2.5.3. Совокупность совокупностей

Реальные задачи, решаемые на ЭВМ, содержат в подавляющем большинстве случаев *упорядоченные совокупности данных*, для представления которых использовать скалярные типы очень неудобно. Например, когда обсуждается успеваемость в классе, то упоминается значение оценки каждого ученика; это значение будет скалярной величиной.

Когда же речь заходит об уровне успеваемости во всей школе, то обсуждение ведется на уровне классов, т. е. на *совокупности* оценок и их значений. Если пойти дальше и анализировать успеваемость учащихся в районе или городе, то рассматривать надо совокупность оценок успеваемости школ, т. е. *совокупность совокупностей оценок*. Так рождаются сложные структуры данных; с ними мы имеем дело повседневно. Давай попробуем разобраться с основными их типами.

Разложены по полочкам. Наиболее простым типом агрегатных данных являются *массивы* — упорядоченные наборы скаляров одного типа. Сами скаляры называются *элементами* массива. Элементы массива имеют *индексы*, представляющие собой множество целых чисел. Массивы могут быть *одномерными* и *многомерными*. Примером одномерного массива является перечень учеников в классе, показанный в таблице 23. Здесь индексами является множество { 1, 2, 3, 4, 5 }. Одномерным будет и массив учебных предметов (см. табл. 24), изучаемых в классе. Если же теперь представить успеваемость учеников по этим предметам, то массив оценок будет двумерным (см. табл. 25). Здесь индексами служат множества { 1,2, ..., 5 } и { 1, 2, 3 }. Обращение к элементу массива производится по имени элемента и по его индексу. Так, чтобы узнать оценку Воробьева по физике, надо отыскать элемент *О* [2,2], а оценка Соколовой по этому предмету находится в элементе *О* [4,2]. Для обращения ко всему массиву ему также присваивается имя. Наконец, для полной информации о массиве рядом с его именем указывается в круглых скобках размерность массива. Например, массив учеников можно определить так: МАСУЧ (5). Массив оценок — МАСОЦ (5,3).

Таблица 23

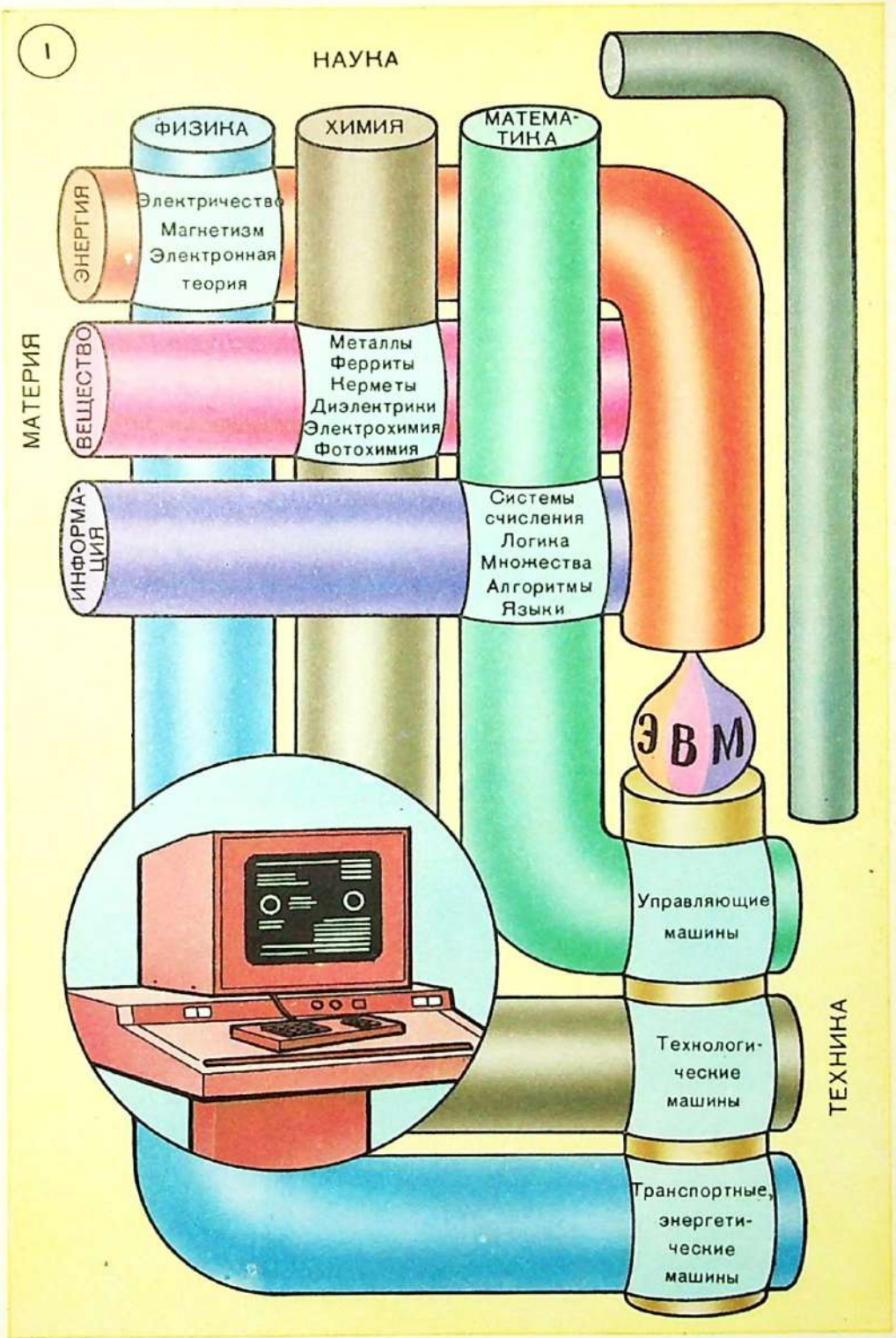
Элемент массива	Значение элемента
У[1]	Синицына
У[2]	Воробьев
У[3]	Орлов
У[4]	Соколова
У[5]	Грачев

Таблица 24

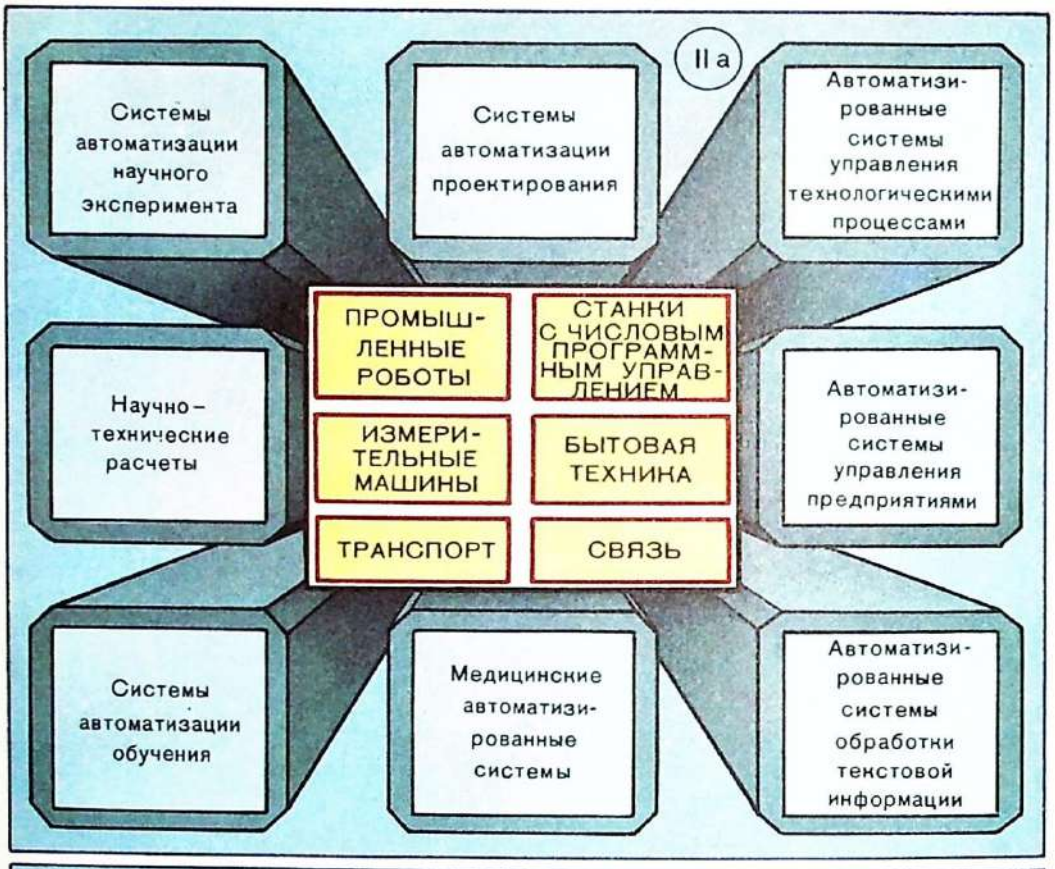
Элемент массива	Значение элемента
П[1]	Математика
П[2]	Физика
П[3]	Литература

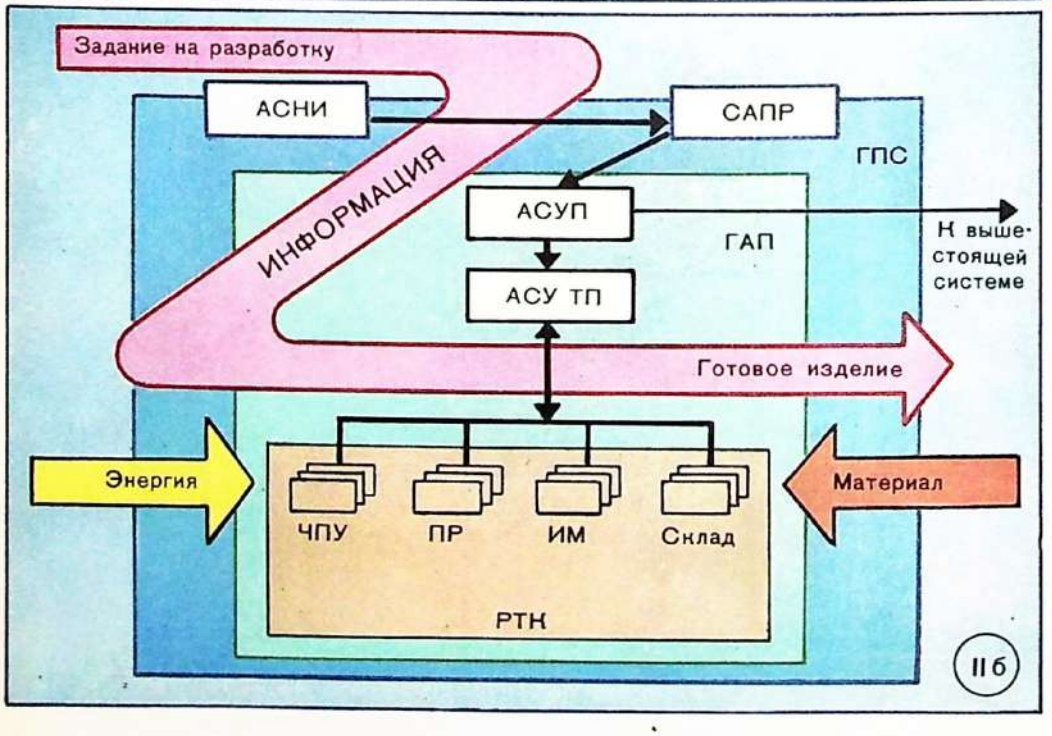
Таблица 25

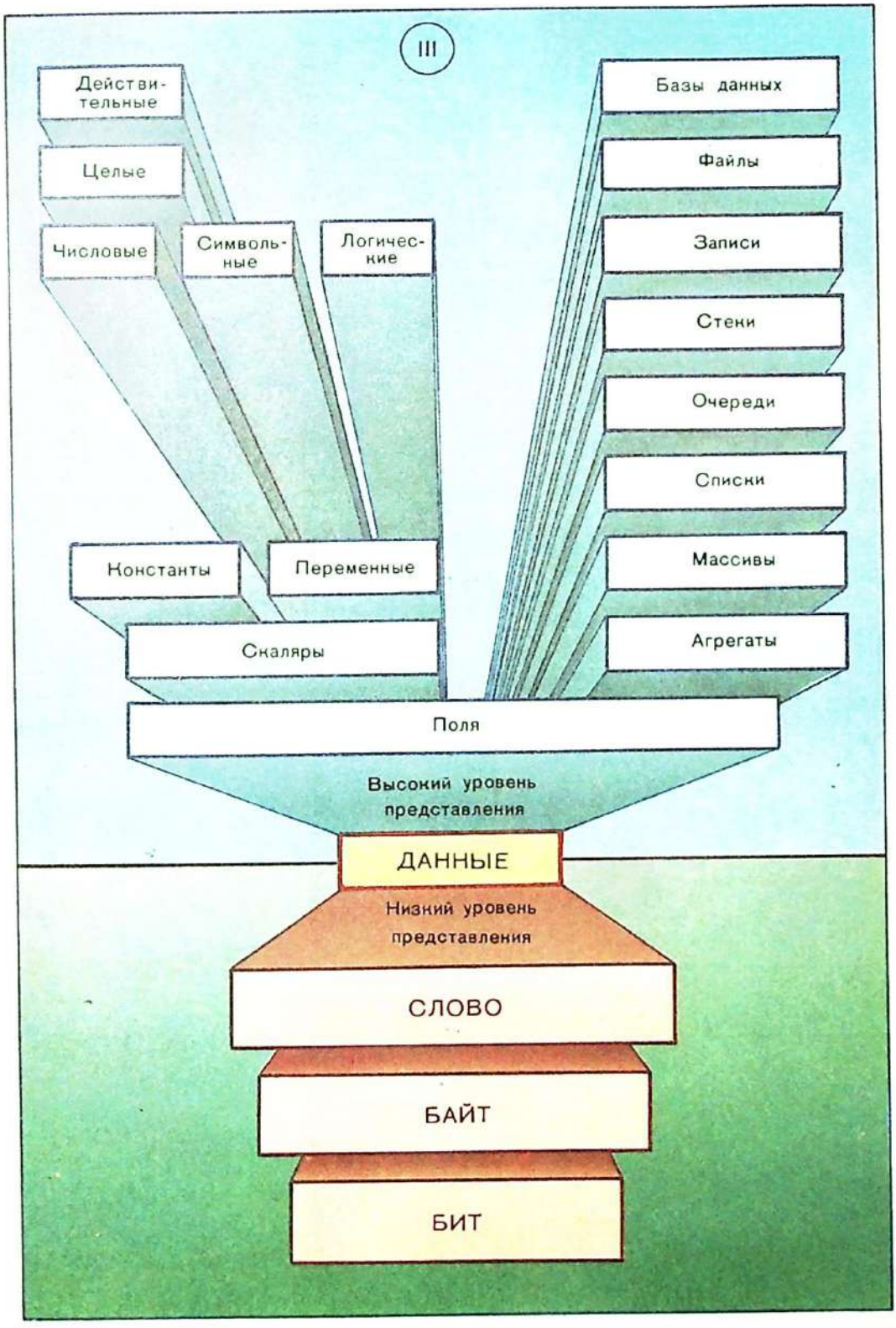
Элемент массива			Значение элемента		
О[1,1]	*О*[1,2]	*О*[1,3]	5	5	5
О[2,1]	*О*[2,2]	*О*[2,3]	4	3	5
О[3,1]	*О*[3,2]	*О*[3,3]	3	3	3
О[4,1]	*О*[4,2]	*О*[4,3]	5	5	4
О[5,1]	*О*[5,2]	*О*[5,3]	5	5	3



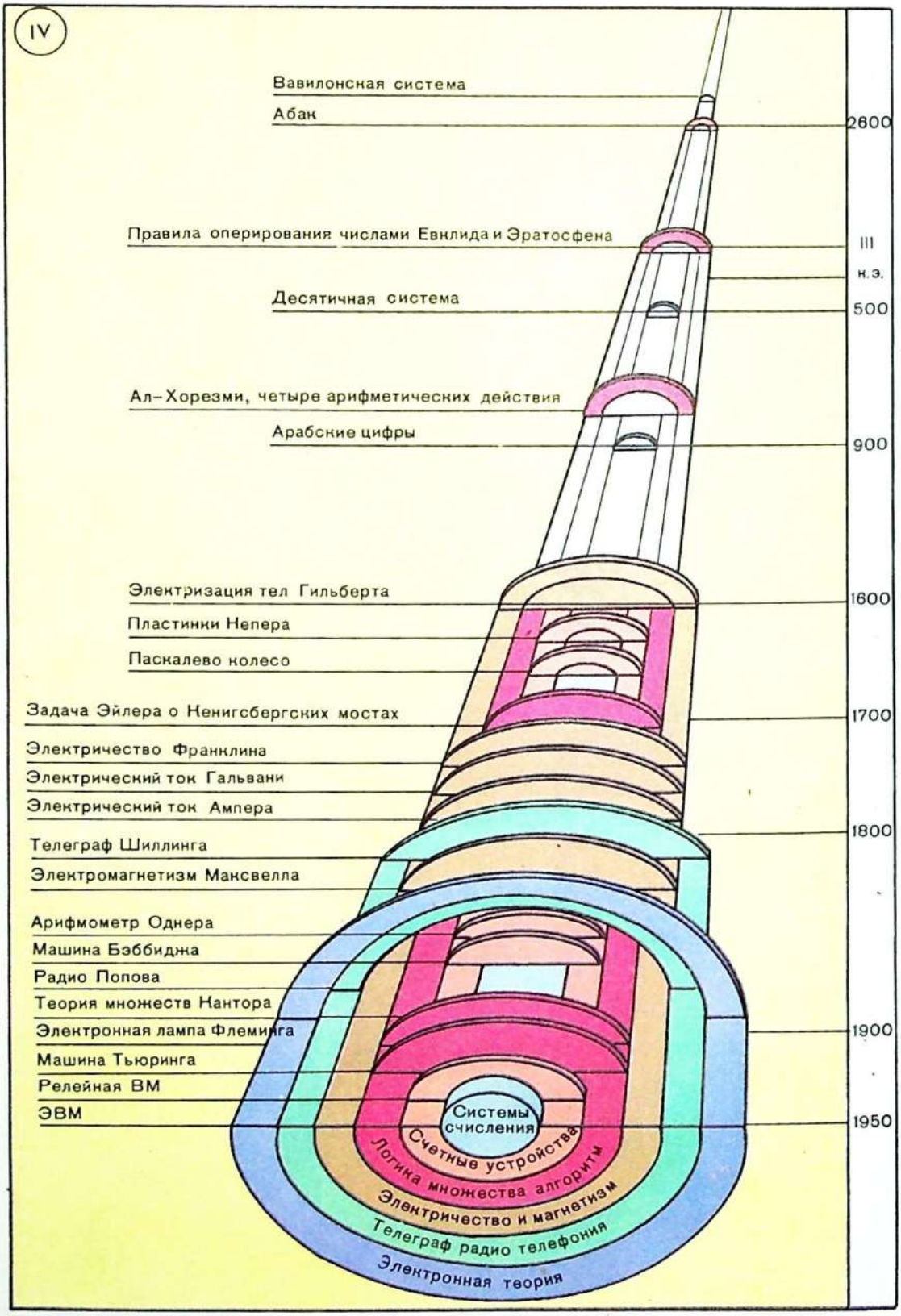
3 Зак. 2448 А. В. Нестеренко, вклейка

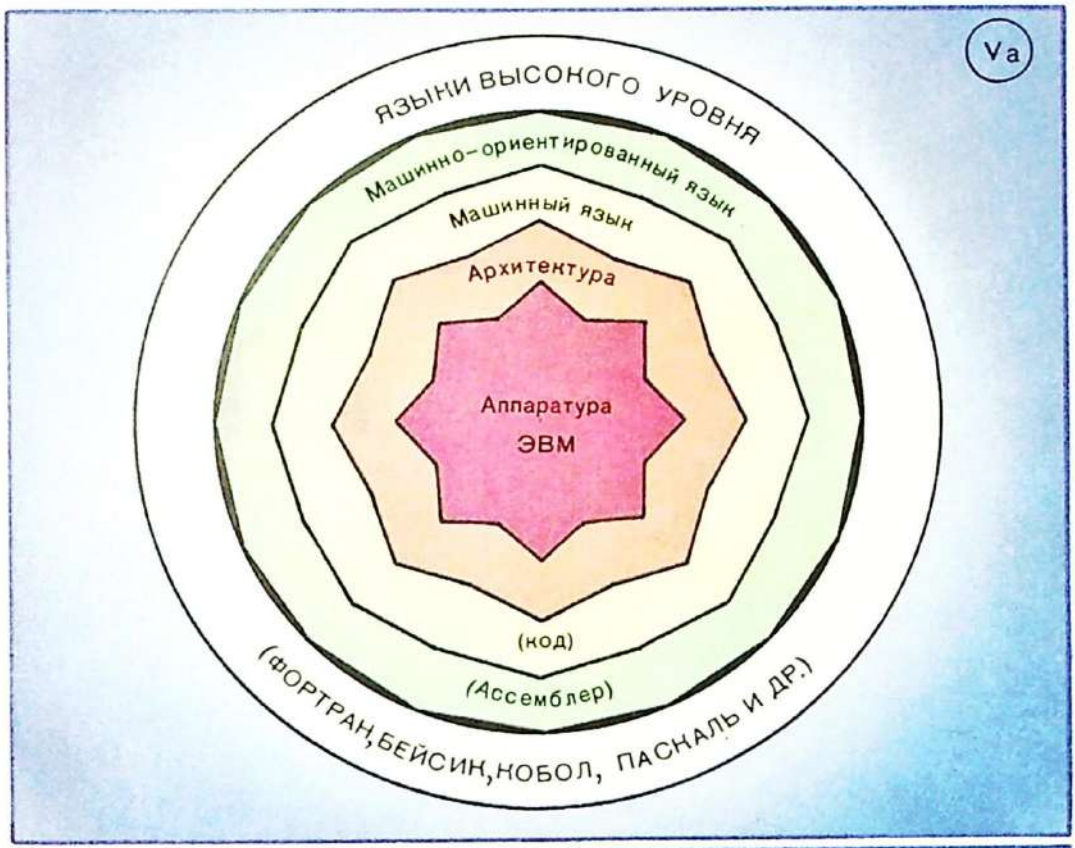

II а
Системы автоматизации научного эксперимента
Системы автоматизации проектирования
Автоматизированные системы управления технологическими процессами
Научно-технические расчеты
ПРОМЫШЛЕННЫЕ РОБОТЫ
СТАНКИ С ЧИСЛОВЫМ ПРОГРАММНЫМ УПРАВЛЕНИЕМ
ИЗМЕРИТЕЛЬНЫЕ МАШИНЫ
БЫТОВАЯ ТЕХНИКА
ТРАНСПОРТ
СВЯЗЬ
Автоматизированные системы управления предприятиями
Системы автоматизации обучения
Медицинские автоматизированные системы
Автоматизированные системы обработки текстовой информации

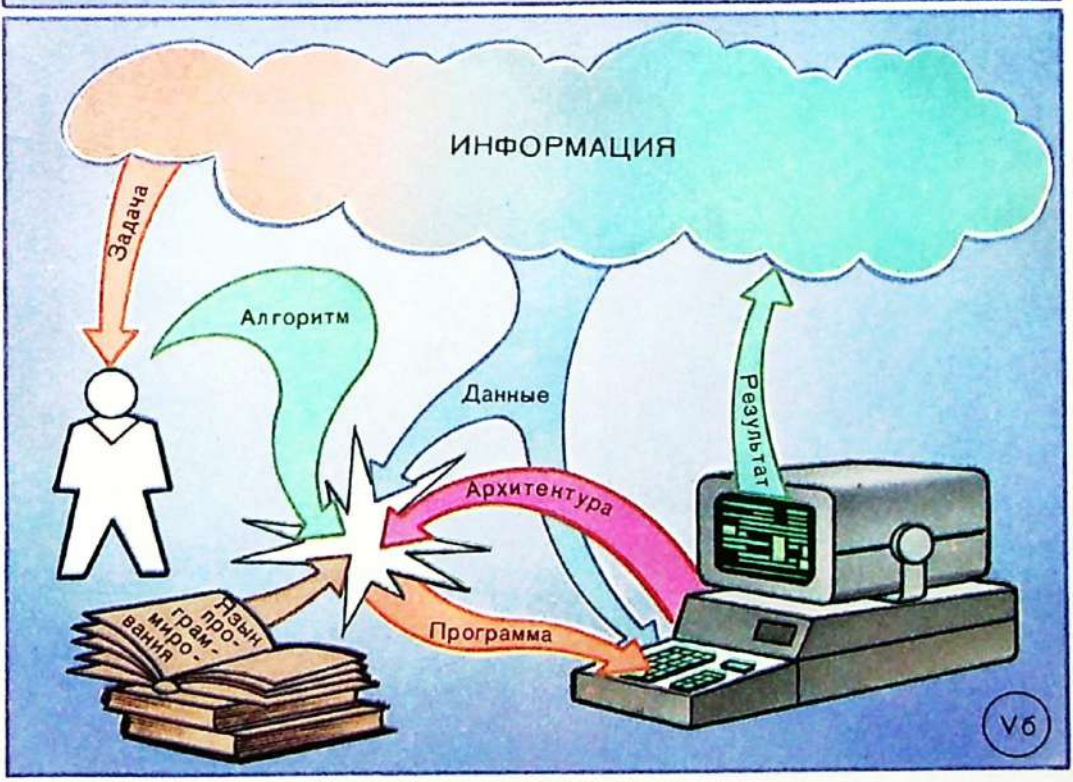

Задание на разработку
АСНИ
САПР
ГПС
ИНФОРМАЦИЯ
АСУП
ГАП
К вышестоящей системе
АСУ ТП
Готовое изделие
Энергия
Материал
ЧПУ
ПР
ИМ
Склад
РТК
II б

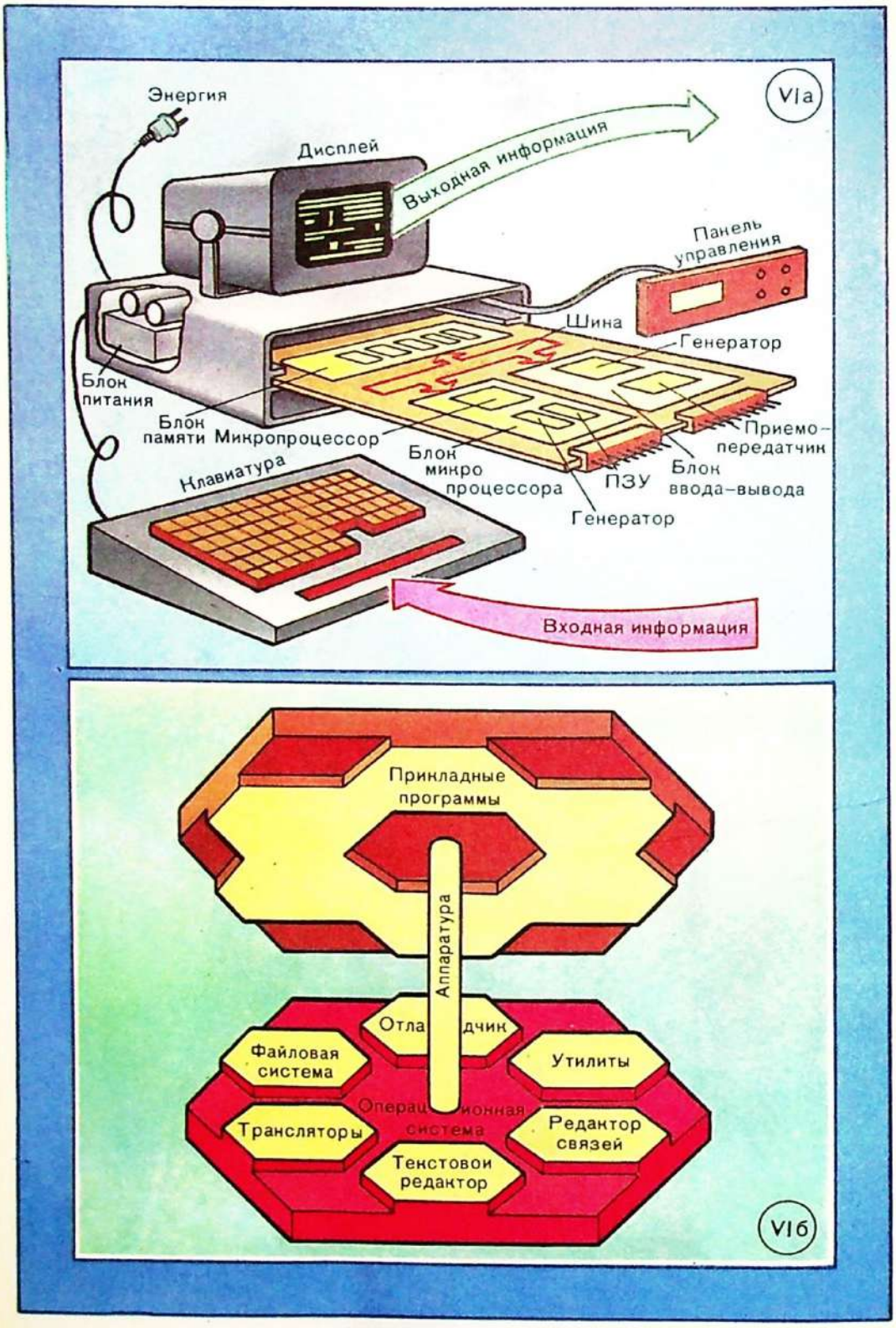

III
Действи-
тельные
Целые
Числовые
Символь-
ные
Логичес-
кие
Базы данных
Файлы
Записи
Стеки
Очереди
Списки
Массивы
Агрегаты
Константы
Переменные
Скаляры
Поля
Высокий уровень
представления
ДАННЫЕ
Низкий уровень
представления
СЛОВО
БАЙТ
БИТ

IV

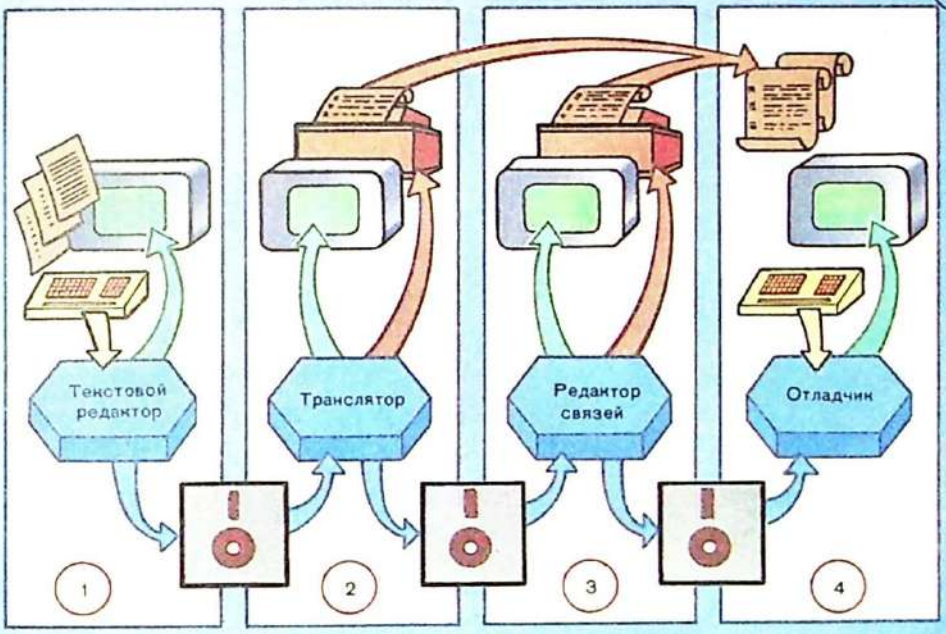

Вавилонская система
Абак
2600
Правила оперирования числами Евклида и Эратосфена
III
н.э.
Десятичная система
500
Ал-Хорезми, четыре арифметических действия
Арабские цифры
900
Электризация тел Гильберта
1600
Пластинки Непера
Паскалево колесо
Задача Эйлера о Кенигсбергских мостах
1700
Электричество Франклина
Электрический ток Гальвани
Электрический ток Ампера
1800
Телеграф Шиллинга
Электромагнетизм Максвелла
Арифмометр Однера
Машина Бэббиджа
Радио Попова
Теория множеств Кантора
Электронная лампа Флеминга
1900
Машина Тьюринга
Релейная ВМ
ЭВМ
1950
Системы счисления
Счетные устройства
Логика множества алгоритм
Электричество и магнетизм
Телеграф радио телефония
Электронная теория

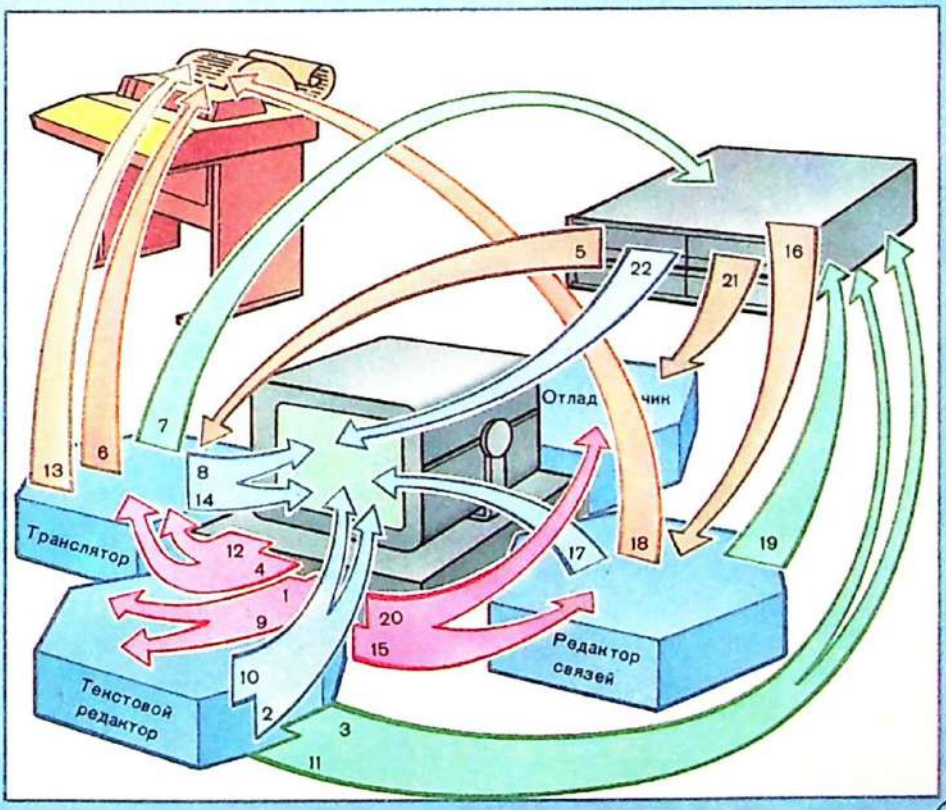

Va
ЯЗЫКИ ВЫСОКОГО УРОВНЯ
Машинно-ориентированный язык
Машинный язык
Архитектура
Аппаратура
ЭВМ
(код)
(Ассемблер)
(ФОРТРАН, БЕЙСИК, КОБОЛ, ПАСКАЛЬ И ДР.)


ИНФОРМАЦИЯ
Задача
Алгоритм
Данные
Результат
Архитектура
Программа
Язык программирования
Vб

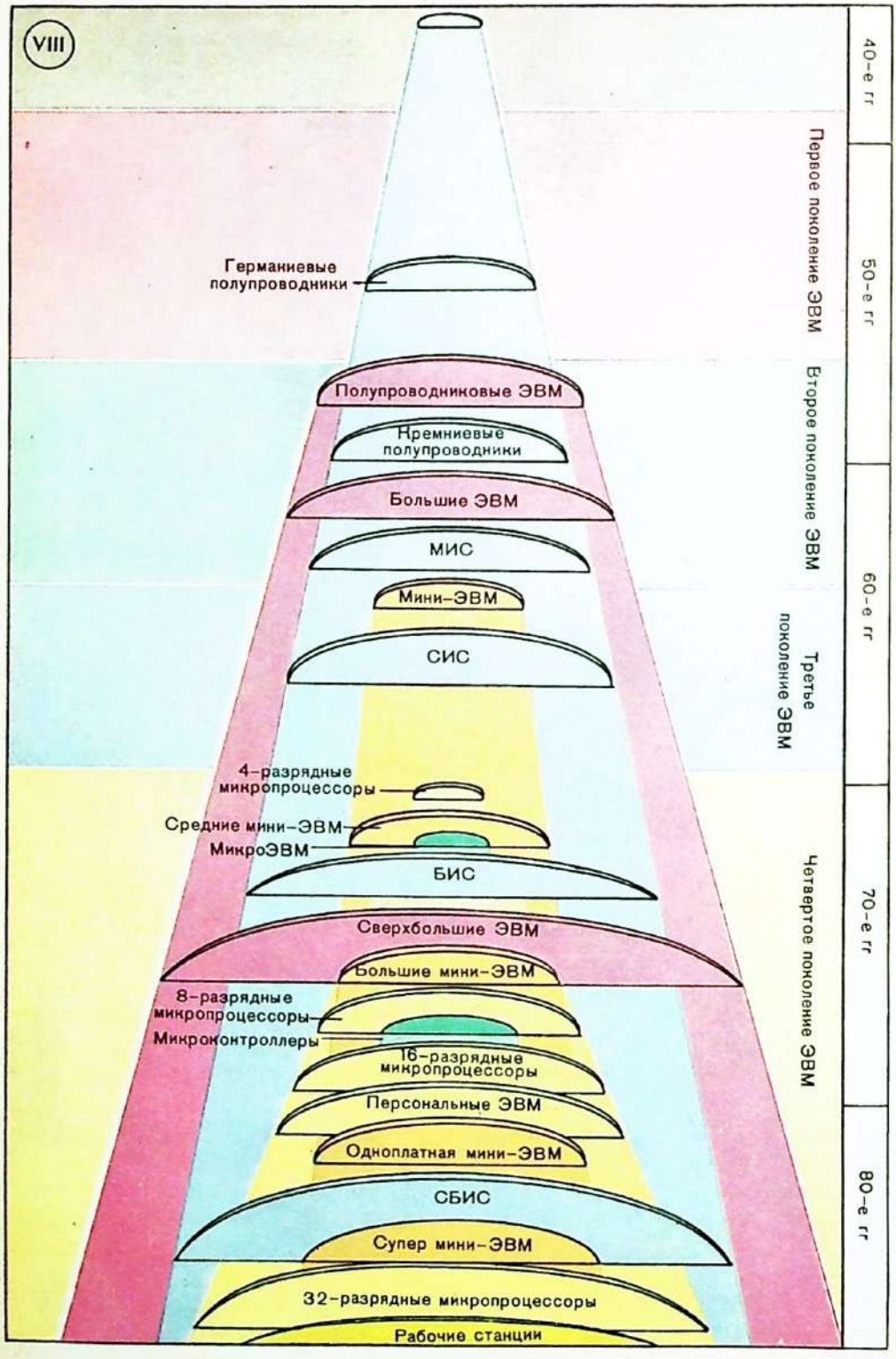

VIа
Энергия
Дисплей
Выходная информация
Панель управления
Шина
Генератор
Блок питания
Блок памяти
Микропроцессор
Блок микро процессора
ПЗУ
Генератор
Блок ввода–вывода
Приемо–передатчик
Клавиатура
Входная информация
Прикладные программы
Аппаратура
Отла дчик
Файловая система
Утилиты
Операц ионная система
Трансляторы
Редактор связей
Текстовои редактор
VIб

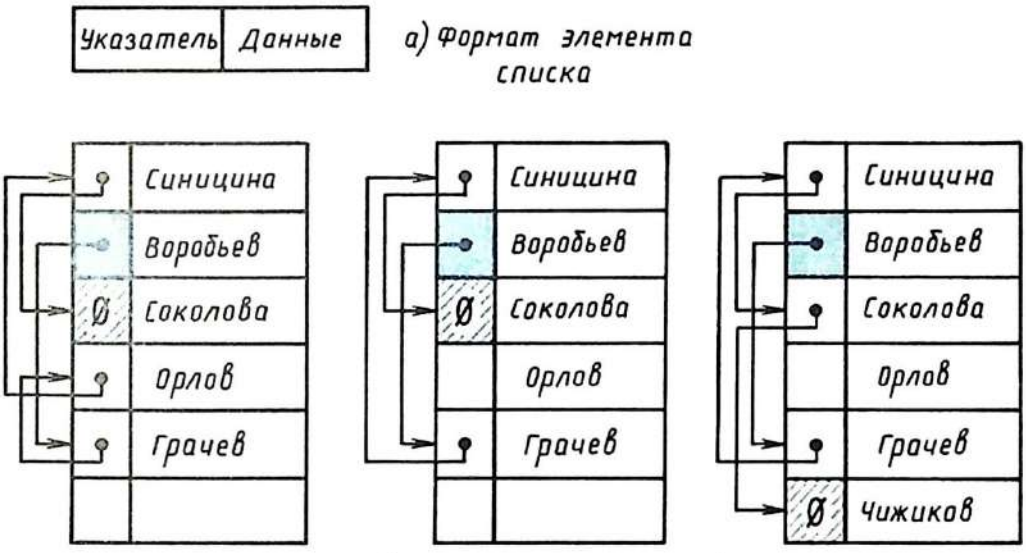
VIIа


Текстовой редактор
Транслятор
Редактор связей
Отладчик
1
2
3
4


Отладчик
Транслятор
Текстовой редактор
Редактор связей
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

VIIб


VIII
40-е гг
50-е гг
60-е гг
70-е гг
80-е гг
Первое поколение ЭВМ
Второе поколение ЭВМ
Третье поколение ЭВМ
Четвертое поколение ЭВМ
Германиевые полупроводники
Полупроводниковые ЭВМ
Кремниевые полупроводники
Большие ЭВМ
МИС
Мини-ЭВМ
СИС
4-разрядные микропроцессоры
Средние мини-ЭВМ
МикроЭВМ
БИС
Сверхбольшие ЭВМ
Большие мини-ЭВМ
8-разрядные микропроцессоры
Микроконтроллеры
16-разрядные микропроцессоры
Персональные ЭВМ
Одноплатная мини-ЭВМ
СБИС
Супер мини-ЭВМ
32-разрядные микропроцессоры
Рабочие станции

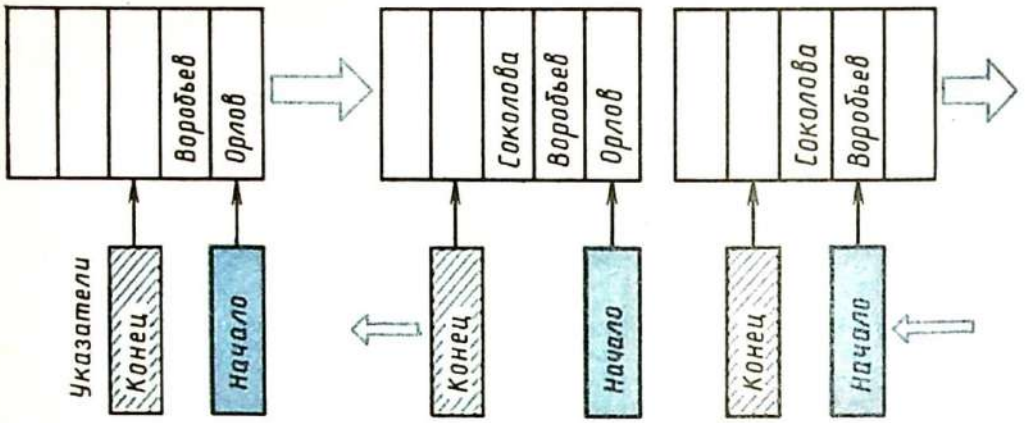


34

Размерность обрабатываемых на ЭВМ массивов, конечно, может быть и более двух. Так, например, массив средних баллов школ города по классам и по изучаемым в школе предметам будет *трехмерным*. Представить его уже затруднительно, а *четырехмерный* массив, в который превратится *предыдущий* массив, если включить еще и учеников, вообще необозрим. Но для ЭВМ не представляет труда не только хранить такие данные, но и мгновенно находить требуемый элемент. Как это происходит в машине, ты узнаешь из последующих глав. А пока рассмотрим следующий тип данных.

Составлены списки. *Списком* называют *одномерный упорядоченный набор переменных одного типа*. Список отличается от массива тем, что его размер является величиной переменной, т. е. элементы могут добавляться в список и удаляться из него, причем в любом месте, поэтому элемент списка состоит из двух частей — *данного* и *указателя* на следующий элемент. Последний элемент в списке имеет значение указателя, равное нулю. На рисунке 34 схематично показан формат элемента списка и правила операций со списком.

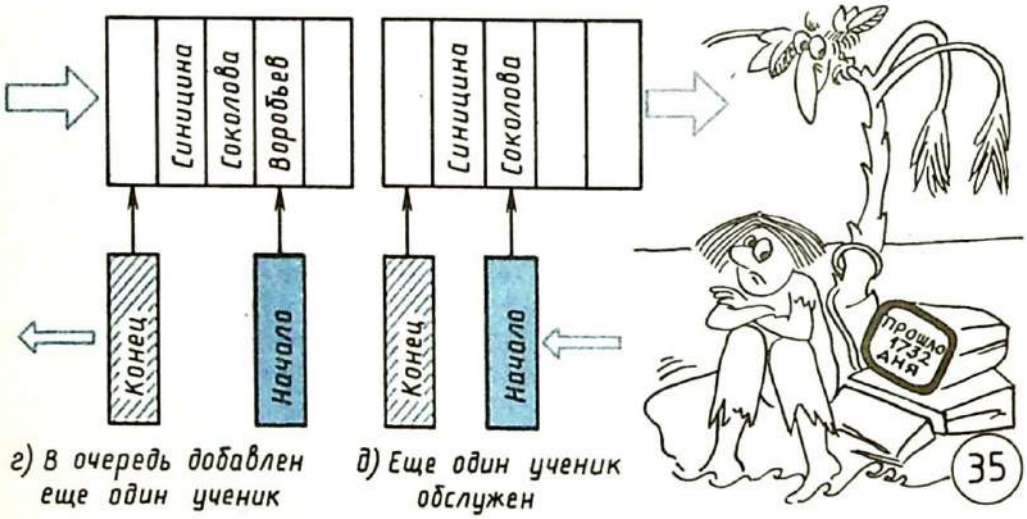
Поставлены в очередь. *Очередь* также является *одномерным упорядоченным набором переменных одного типа* и также может иметь переменную длину. Но в отличие от списка элементы добавляются в очередь с одного конца, а удаляются с другого. И никак иначе. Эти действия выполняются по принципу «первым пришел — первым обслужен» (*First In — First Out,* FIFO), который широко известен, и воспитанный школьник старается его придерживаться при покупке мороженого или билета



а) В очереди два ученика

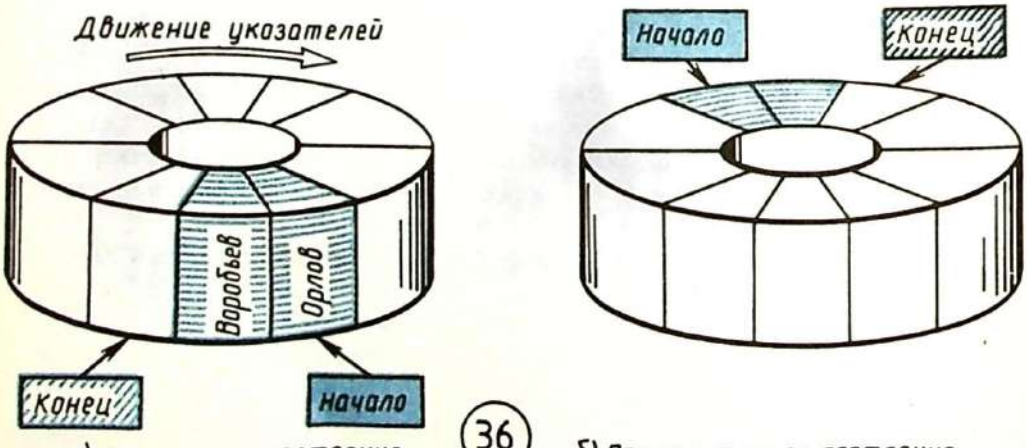
б) В очередь добавлен еще один ученик

в) Один ученик обслужен



г) В очередь добавлен еще один ученик

д) Еще один ученик обслужен



а) Начальное состояние

36

б) Промежуточное состояние

в кино. Для организации очереди используются *указатели,* определяющие текущее положение начала и конца очереди. На рисунке 35 показаны структура очереди и различные ее состояния при добавлении и удалении элементов.

Недостатком такой *организации* очереди является отсутствие возможности использовать элементы, освобождающиеся по мере ее обслуживания. Так, на рисунке 35 уже имеется два свободных элемента, которые никогда не будут использованы этой очередью, так как указатели при добавлении или удалении элементов двигаются все дальше от исходного состояния. Чтобы решить эту проблему, часто очередь организуют в виде кольца, так как при достижении последнего элемента в очереди ее заполнение опять начинается с первого элемента. На рисунке 36 изображена такая организация очереди.

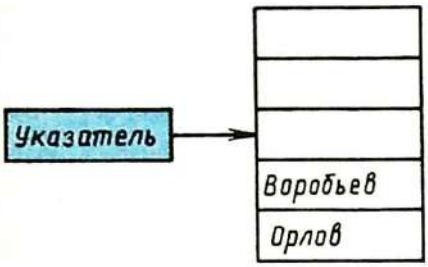
При включении в такую очередь большого числа элементов может наступить состояние ее *переполнения.* Тогда при первом методе организации указатель «конец» достигнет границы отведенного под очередь места, а при кольцевой организации этот же указатель, описав окружность, догонит указатель «начало». В первом случае ситуация сложится критическая, так как добавлять новые элементы больше некуда. Во втором случае достаточно подождать, пока произойдет обслуживание и удаление очередного элемента. Указатель «начало» сдвинется на один элемент, тем самым освободив место для включения в очередь нового элемента.

И в том и в другом случае при организации очереди необходимо постоянно следить за положениями указателей, чтобы предотвратить переполнение.

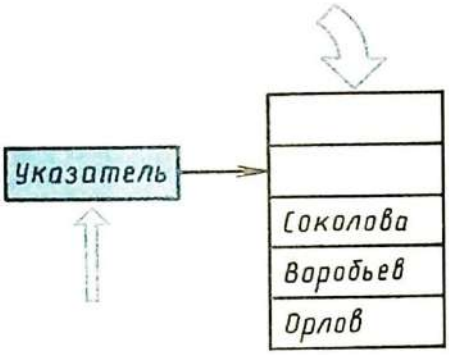
Но порядка нет! При организации данных по типу очереди существует привычный нам порядок. Но применяется и принцип невоспитанного школьника, стремящегося заполучить мороженое без очереди, т. е. по правилу «последним пришел — первым обслужен» (*Last In — First Out, LIFO).* Такая организация очереди получила название *стек.* Схема стека показана на рисунке 37.

Стек также является одномерным типом данных и имеет переменную длину, но у него есть только один вспомогательный элемент — *указатель стека,* который определяет первый свободный элемент. Данные при добавлении записываются на это место, а указатель сдвигается вверх, задавая тем самым следующий свободный элемент. Обслуживание и удаление данного производится сверху, и указатель сдвигается на один элемент ниже.

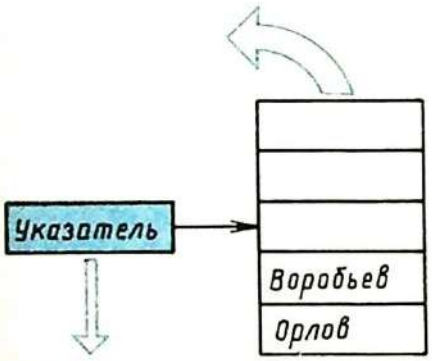
Для стека возможно не только состояние переполнения, когда указатель, двигаясь вверх, достигнет границы, т. е. места, отведенного под данные. Неприятная ситуация может возникнуть и тогда, когда указатель, двигаясь вниз, не остановится при пустом стеке, т. е. удаление данных будет продолжаться. Следовательно, при стековой организации также необходимо следить за положением указателя.



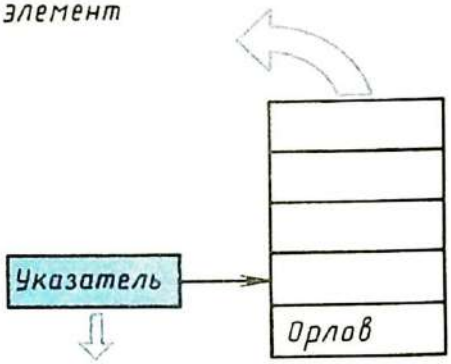
а) В стеке два занятых элемента



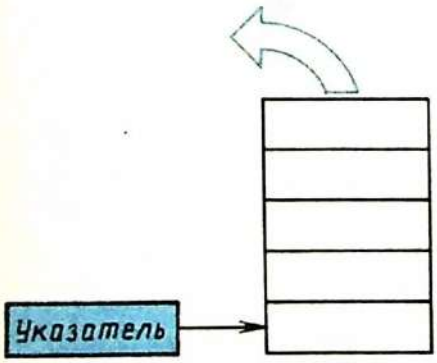
б) В стек помещен еще один элемент



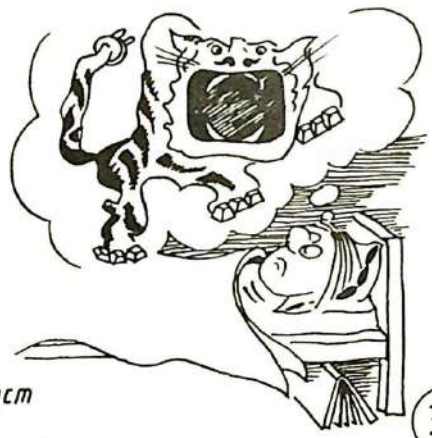
в) Один элемент удален



г) Еще один элемент удален

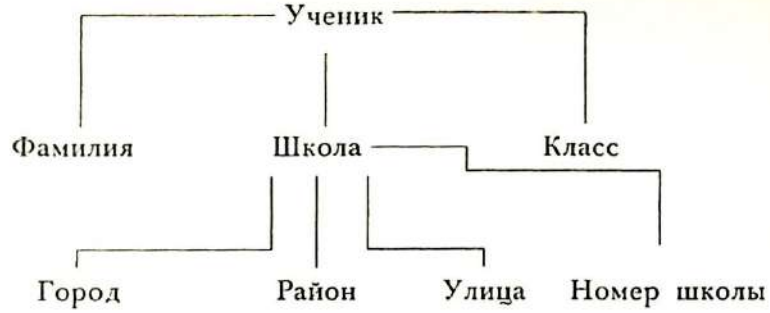


д) Удалены все элементы, стек пуст



37

Снова деревья, но уже на полях. Наиболее сложным типом данных являются *записи*, или *структуры*. Записи представляют собой совокупность данных разных типов и могут быть как фиксированной, так и переменной длины. Записи, как и массивы, являются именованными, в отличие от списковых типов данных. Совокупность этих данных, определяемая как запись, может быть представлена схемой в виде дерева:



Элементом записи является любая вершина такого дерева, но не все элементы равнозначны. Если из приведенного примера два элемента можно описать через производные вершины, как, например:

Ученик = (Фамилия, Школа, Класс)

Школа = (Город, Район, Улица, Номер школы), то остальные элементы не имеют производных и являются *полями*; запись, собственно, и состоит из полей. Присвоим *элементам* и *записи* имена. Тогда вышеприведенная структура будет иметь вид:

ЗАПУЧ = (ФАМ, ГОР, РАЙ, УЛЦ, НШК, КЛС)

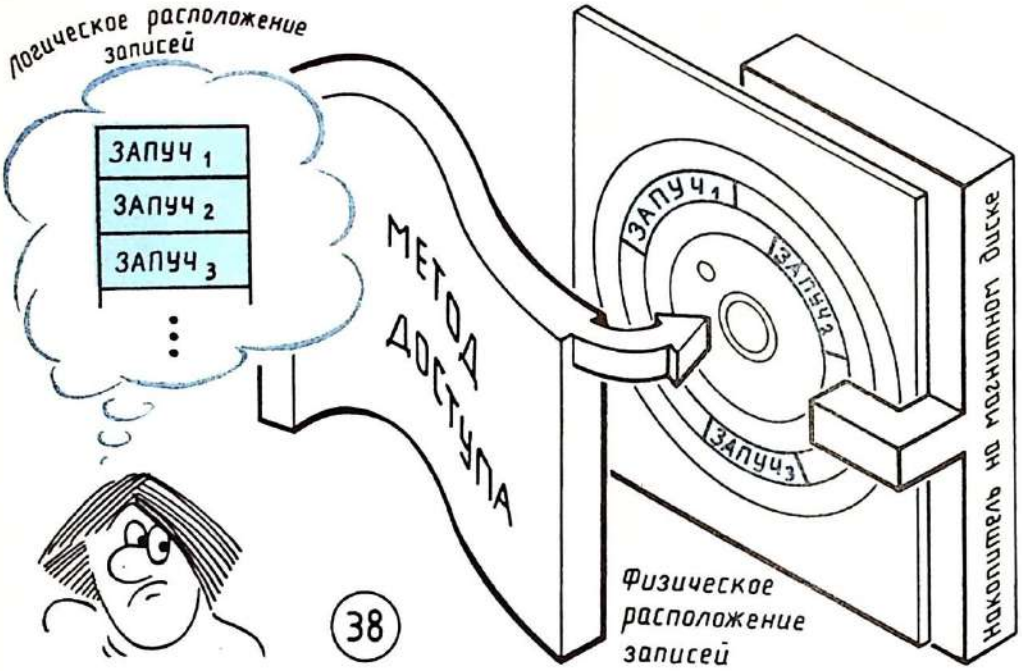
Для того чтобы разобраться, где же в этой записи расположены определенные совокупности полей, ее обычно логически описывают следующим образом:

```
ЗАПУЧ
        ФАМ
        ШКЛ
              ГОР
              РАЙ
              УЛЦ
              НШК
        КЛС
```

Если рассмотреть конкретные значения полей, то они могут иметь следующий вид:

ЗАПУЧ = (ОРЛОВ, КИЕВ, МОСКОВСКИЙ, ЛЯТОШИНСКОГО, 220, 9). Здесь мы видим, что поля записи содержат данные различных типов, в данном случае — символьные и целые. Кроме того, поля имеют и различную длину. Поэтому при логическом описании записи, кроме ступенчатого расположения совокупностей полей, обязательным является указание типа и длины данного, помещаемого в поле.

База данных. Типы данных, о которых ты узнал, практически используются при хранении и обработке сравнительно небольших объемов информации. Запоминание и преобразование таких данных в ЭВМ схоже с аналогичными процессами, протекающими в голове человека, где мозг, к сожалению, пока еще не может вместить все необходимые нам сведения. Основную часть этих сведений человек держит во *внешних* хранилищах — книгах, библиотеках, на магнитофонных лентах и т. д. «Голова» ЭВМ тоже, к сожалению, ограничена в возможностях



хранения информации, поэтому и ЭВМ имеет свои *внешние хранилища информации*. Об их типах и устройстве мы поговорим несколько позже. Сейчас же преследуя цель разобраться в многообразии типов данных, определим, какие же из них применяются для организации *внешних совокупностей информации*.

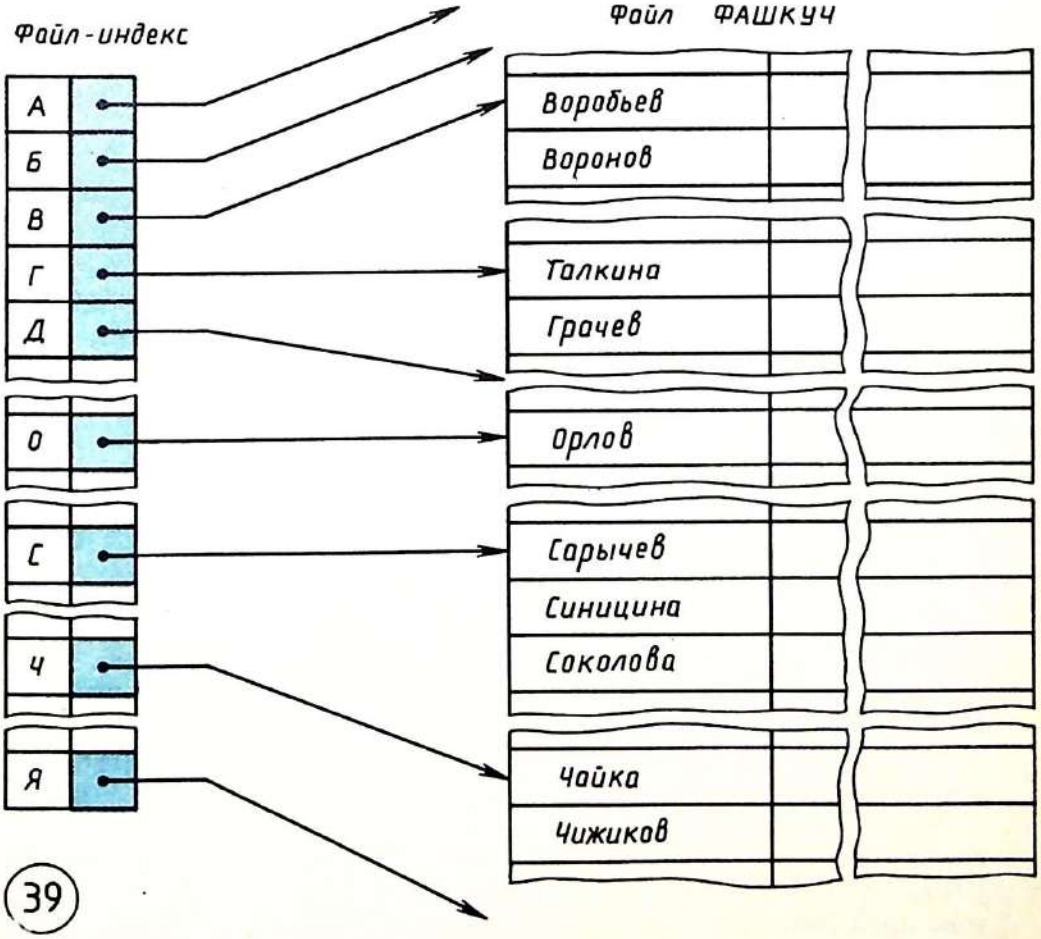
Основным таким типом, принятым к обработке данных на ЭВМ, является *файл* (*file* — папка для подшивки бумаг, скоросшиватель). Иногда подобные структуры называют *теками* (от греческого $\theta\eta\kappa\eta$ — хранилище).

Файл — это *набор записей*, связанных между собой и содержащих данные, имеющие конкретное назначение для определенной области деятельности. Записи ЗАПУЧ, собранные по всем ученикам школы, создадут файл, который можно назвать ФАШКУЧ, содержащий необходимые сведения о составе учащихся школы. Эти сведения необходимы, например, для повседневной деятельности директора школы.

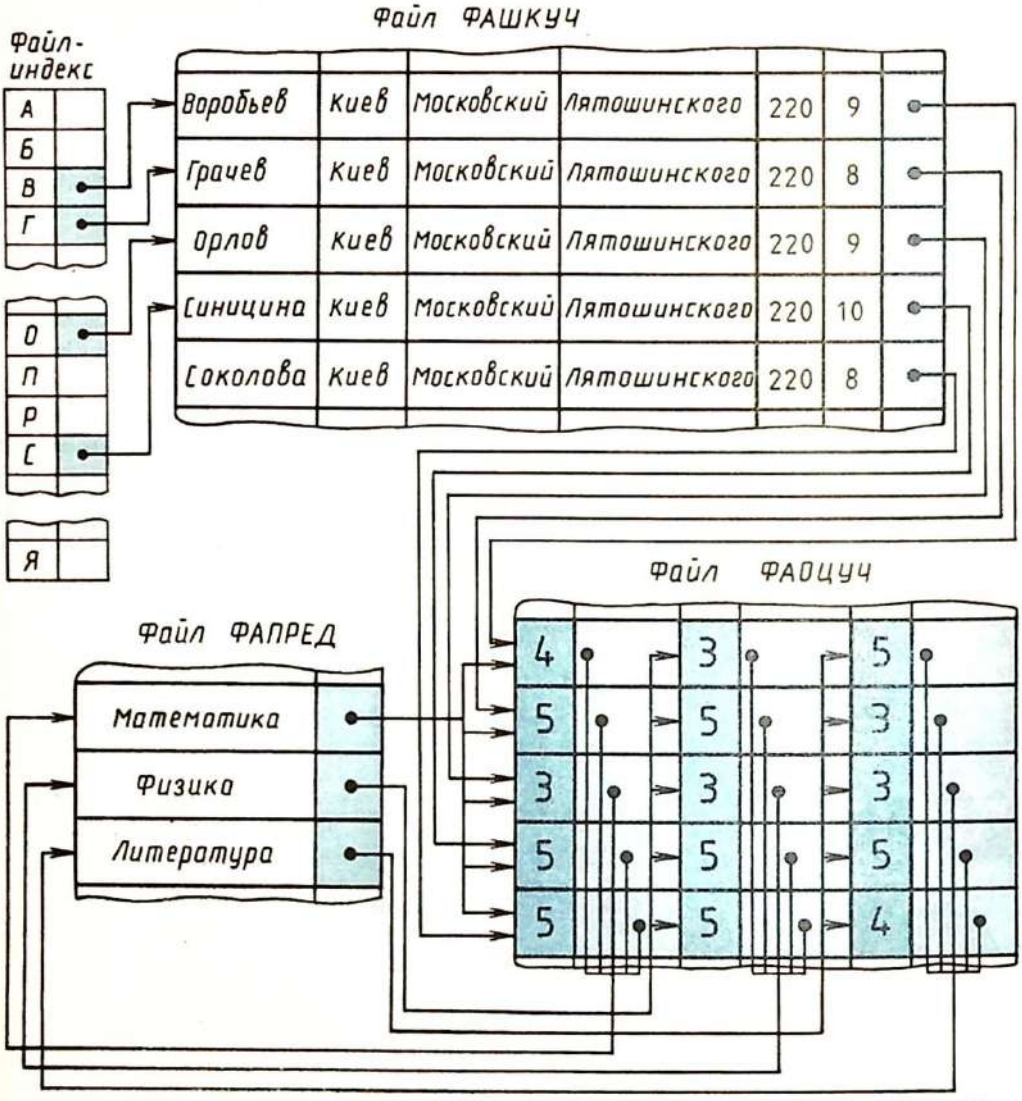
Как же файлы располагаются на внешнем носителе, например на магнитном диске? Учитывая, что на диске хранится много файлов, причем разной длины, и что они могут обновляться, расширяться, удаляться, оказалось, что удобнее каждый файл хранить не целиком в одном месте диска, а разбивать его на фрагменты — блоки, записывая их в произвольные свободные участки диска. Но такая организация усложняет поиск файла и его запись. Поэтому были созданы специальные программы, помогающие осуществлять эти процессы. Эти программы реализуют так называемый *метод доступа к файлам*. На рисунке 38 *метод доступа* как бы разделяет *реальное* (физическое) располо-

жение записей файла на диске и представляемое программистом — *логическое* расположение.

Существует несколько методов доступа к файлам. Можно просматривать файл с начала и до конца, чтобы найти требуемую запись. Такой просмотр называется *последовательным,* а файл, соответственно,— *с последовательной организацией записей.* Эта организация является самой простой, но требует много времени на обработку. Представь, что в реальном файле, содержащем тысячи записей, надо найти сведения об ученике Чижикове, находящемся где-то в конце файла, а начинать поиск надо с начала. Для ускорения этого процесса применяют другие методы доступа к файлам, например *индексно-последовательные.* При создании таких файлов их предварительно упорядочивают в определенной последовательности; например, если речь идет о файле с фамилиями учеников, то записи в нем упорядочивают в алфавитном порядке фамилий. Затем строится отдельный файл, называемый *индексом* и содержащий указатели на начало определенных областей основного файла, например в нашем случае на начало групп записей с фамилиями, начинающимися с одной буквы, как это показано на рисунке 39.



39



40

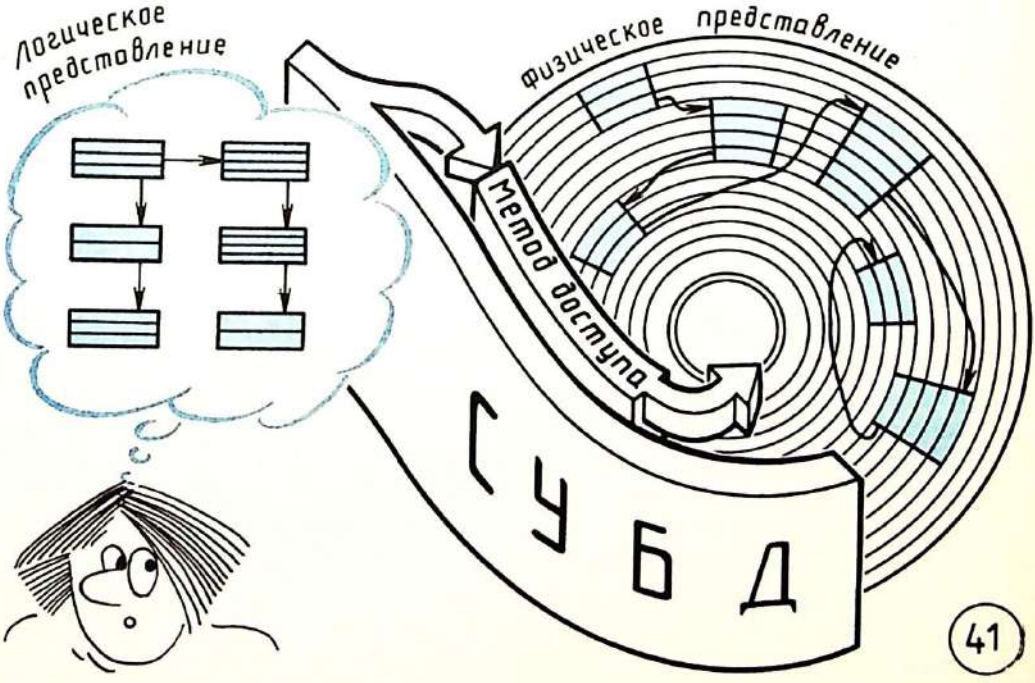
Теперь, для того чтобы найти Чижикова, надо просмотреть последовательно в индексе 23 записи и несколько записей в основном файле (в нашем примере – две). На это уйдет гораздо меньше времени, чем при просмотре полностью основного файла до необходимой записи. Процесс поиска можно еще ускорить, создав двойной индекс и введя дополнительный указатель, например на группу записей, содержащих фамилии, у которых одинакова не только первая буква, но и вторая.

Файл ФАШКУЧ содержит некоторые сведения, необходимые для анализа деятельности школы. Однако этих сведений, конечно, недостаточно, поэтому можно создать файл оценок каждого ученика по изучаемым предметам. Назовем его ФАОЦУЧ. Школьных предметов набе-

рется тоже немало, поэтому можно организовать файл предметов ФАПРЕД. Теперь, чтобы обеспечить директора школы быстро и в любой момент необходимыми данными о любом ученике или группе учеников, эти файлы необходимо как-то связать между собой, чтобы облегчить и ускорить поиск необходимых сведений. На рисунке 40 показана совокупность файлов, увязанная с помощью указателей. Подобные совокупности экземпляров различных записей и связей между ними называются базами данных (БД).

Поиск необходимой записи среди множества подобных, да еще рассредоточенных между различными файлами,— задача непростая. Как всегда, программиста выручают программы. Такие специальные программы, обеспечивающие простую и надежную выборку необходимых записей из БД, назвали *системой управления базы данных* (СУБД). Сложное физическое представление данных с помощью СУБД скрывается от программиста, а ему дается ясное и простое логическое представление (рис. 41).

Для того чтобы получить сведения об ученике Орлове с помощью базы данных ШКОЛА, приведенной выше на рисунке 40, необходимо в файле ФАШКУЧ найти запись с полем ФАМ, содержащим значение «Орлов». Из этой записи узнаем, где Орлов учится. Затем по указателю попадаем на запись файла оценок ФАОЦУЧ, из которой узнаем первую оценку, например 3. По указателю переходим на файл предметов ФАПРЕД, вернее, на запись этого файла, содержащую в единственном своем поле значение «математика». Далее анализируем следующее поле записи файла ФАОЦУЧ; выбираем оценку и переходим на файл ФАПРЕД, и т. д.



41

С помощью базы данных ШКОЛА можно, например, быстро определить оценки всех учеников по любому предмету. Обратившись к файлу ФАПРЕД и найдя необходимую запись, по указателю выходим на поля записей файла ФАОЦУЧ, содержащие требуемые сведения.

Завершая эту главу, мы оставляем понятие *указателя (last but not least* — последнее по упоминанию, но не по значению) не раскрытым до конца. Действительно, как физически реализуются те стрелочки, которыми мы изображали связи данных на рисунках? Ответ на этот вопрос ты найдешь в следующей главе, а пока взгляни на рисунок цветной вклейки IV, где приведена ретроспектива развития основных направлений науки и техники, приведших к появлению в середине нашего века электронной вычислительной машины. К таким направлениям прежде всего следует отнести развитие систем счисления, логики, теории множеств и алгоритмов, электричества и магнетизма, электронной теории, а также технических решений в области счетных устройств и устройств связи, например телеграфа, радио и телефонии. Почти пять тысяч лет потребовалось человечеству для создания ЭВМ.

Раньше назначение программ заключалось в управлении нашими вычислительными машинами, теперь назначение вычислительных машин состоит в исполнении наших программ.

Э. Дейкстра

ГЛАВА 3. ЭВМ: АРХИТЕКТУРА

3.1. АРХИТЕКТУРА. ПЕРВОЕ ЗНАКОМСТВО

Согласно идее Ч. Беббиджа, основу вычислительной машины составляет «мельница» — устройство для выполнения арифметических операций. Потребляемым сырьем «мельницы» служат числа, хранящиеся на «складе». Измененные числа, представляющие продукцию нашей «мельницы», выводятся с помощью *устройства вывода данных*.

Какой бы ни была ЭВМ — большой, малой или очень малой, приведенная аналогия обобщенно дает понятие о работе каждой из них. И для эффективного использования вычислительной машины, а тем более для успешного программирования необходимо иметь определенные знания об ее внутреннем устройстве и работе. Чем больше ты будешь об этом знать, тем большую пользу ты можешь иметь от ее применения.

Каким должно быть изучение, чтобы можно было приступить к программированию? Это зависит от класса решаемой задачи. Но в любом случае считается, что сначала необходимо определить возможности машины для выполнения самого важного процесса, связанного с профессией программиста,— *обработки данных*. Совокупность знаний о ресурсах ЭВМ, доступных пользователю и помогающих наиболее эффективно решать поставленную задачу, и составляет понятие *архитектура ЭВМ*.

С точки зрения программиста имеются две машины. С одной стороны — физическая машина, т. е. устройство, наполненное электронными приборами («железками»), опутанное проводами, которое шумит и выделяет тепло. С другой стороны, в воображении программиста существует некоторая абстрактная машина, для которой он пишет программы. Таким образом, архитектура и представляет эту абстрактную модель реальной физической машины. Для широких масс пользователей, решающих на своих «персоналках» сравнительно простые вычислительные задачи, достаточно элементарных представлений об архитектуре машины. Возможность работы с ЭВМ как многоопытным профессионалам, так и начинающим школьникам предоставляется за счет использования различных языков программирования. Степень различия этих языков как раз и заключается в степени учета возможностей машины, т. е. в глубине проникновения в ее архитектуру. Чем дальше

язык конкретной ЭВМ от использования ее архитектуры, тем *выше* его уровень и тем проще его осваивать и применять. Языки, наиболее полно использующие возможности машины, называются *машинно-ориентированными* и применяются опытными программистами для решения специальных задач. На рисунке цветной вклейки V, *а* приведена схема, иллюстрирующая взаимное расположение языков программирования и аппаратуры ЭВМ.

Как ты уже знаешь, машина понимает только один язык: последовательности единиц и нулей. Поэтому, естественно, программы, которые программисты пишут на различных языках, должны быть сведены к единому виду, т. е. к единицам и нулям. Это достигается тоже с помощью программ, написанных ранее, более опытными профессионалами. Такие программы называются *трансляторами*. Сколько языков программирования, столько и трансляторов. Поэтому перед тем как выполнить твою программу, ее необходимо обязательно протранслировать, т. е. обработать с помощью *программы-транслятора*. Но здесь мы уже забежали немного вперед. А пока давай-ка поговорим еще о том, что же такое программирование.

3.2. ПРОГРАММИРОВАНИЕ. С ЧЕГО НАЧАТЬ?

Для успешного решения любой задачи на ЭВМ программист должен проделать определенные действия, состоящие из 7 этапов:

1. Поставить задачу.
2. Выбрать и составить алгоритм.
3. Определить типы входных и выходных данных.
4. Проанализировать архитектуру конкретной ЭВМ.
5. Выбрать язык программирования.
6. Закодировать алгоритм на языке программирования.
7. Проверить программу.

На рисунке цветной вклейки V, *б* ты видишь изображение связей этих этапов.

Во многих случаях *постановка задачи* (первый этап) не является существенным этапом, например постановка элементарной задачи суммирования пяти чисел очень проста, а вот чтобы решить задачу по созданию системы управления учебным процессом в школе, для этого необходимо уделить большое внимание ее постановке. На этом этапе надо ответить на ряд вопросов о том, какой должна быть будущая программа. Пренебрежение этапом постановки задачи может привести в дальнейшем к потере сил и средств. Рассмотрим те вопросы, которые решаются при постановке задачи суммирования пяти чисел. Они могут быть следующими:

— какими являются числа — целыми или дробными?
— какова значность чисел?
— как вводятся числа в ЭВМ (группой или по одному)?
— в каком виде выводится результат суммирования?

Четкое, продуманное решение по каждому вопросу поможет работе на последующих этапах.

Выбор и составление алгоритма (второй этап) должны происходить с использованием тех структур, которые мы рассмотрели в предыдущих главах.

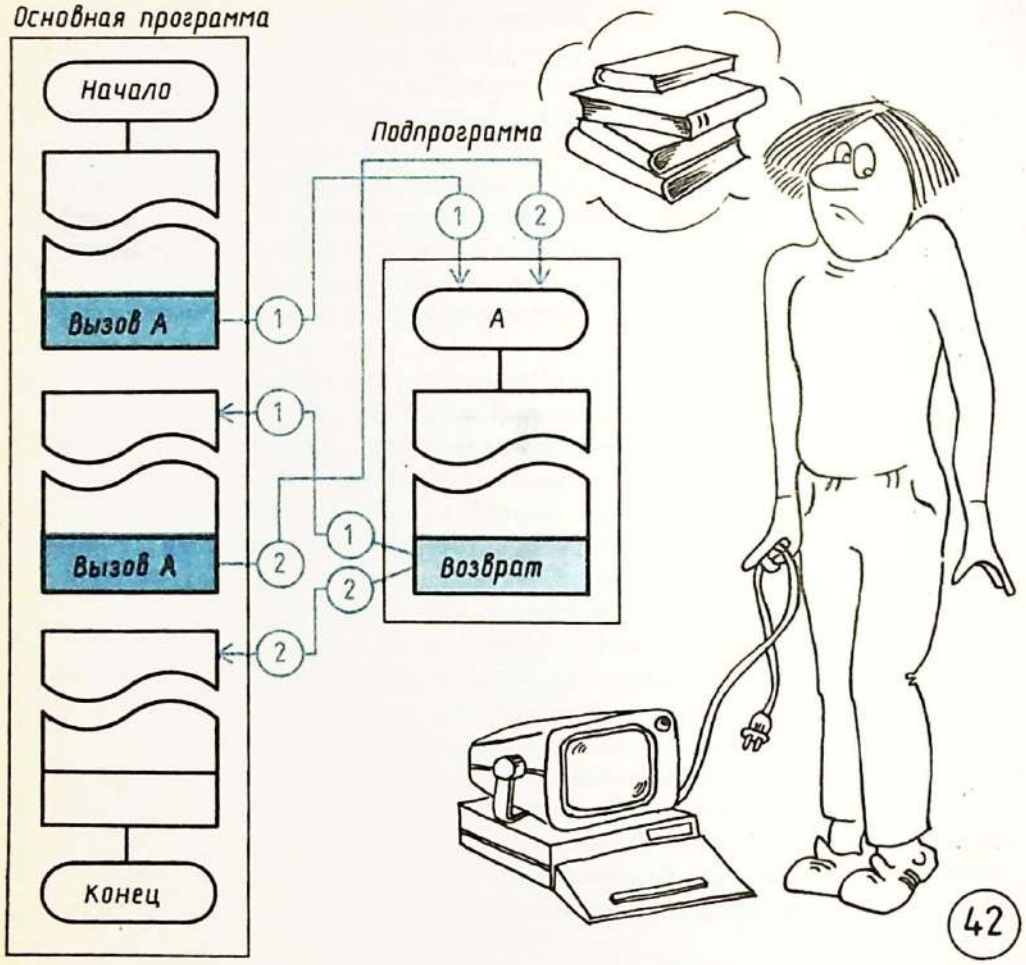
При *определении типов входных и выходных данных* (третий этап) необходимо использовать известные сведения о типах данных.

Следующие три этапа решаются, как правило, одновременно, потому что они тесно связаны между собой. С их содержанием мы ознакомимся позже, как только выясним, что же входит в понятие *архитектура ЭВМ.*

Наконец программа готова. Возможно, что, введенная в память машины, она сможет решить задачу. Но такое случается редко. Даже у опытных программистов весьма небольшая часть программ при первом вводе в ЭВМ работает так, как надо. Поэтому программист вынужден перейти к следующему этапу — *проверке и отладке программы* (седьмой этап). Дело в том, что ЭВМ — устройство, не наделенное какими-либо умственными способностями. Когда говорят, что ЭВМ — «машина умная», то это относится к программе, заложенной в машину, и только к ней. А сама ЭВМ лишь точно исполняет то, что от нее затребовано программой, будь то даже полнейшей бессмыслицей. Малейшая ошибка в программе, скажем, отсутствие одной-единственной точки, может привести к весьма неожиданным результатам, самым распространенным из которых является *останов* программы в неизвестном месте и отсутствие результатов на выходе. Поиск ошибок, который и называется *отладкой программы,* может занять достаточно много времени, даже может быть больше, чем само составление программы. Для облегчения этого процесса придуман ряд методов, которые будут описаны ниже.

Перед тем как перейти к анализу архитектуры ЭВМ, после которого мы можем, наконец, составить программу, вернемся к этапу составления алгоритма. Предложенные структуры — *последовательные, разветвляющиеся, циклические* — вполне достаточны для построения алгоритма практически любой задачи и, соответственно, любой программы. Очень часто случается, что при разработке алгоритма в различных его местах обнаруживаются одинаковые фрагменты, например вычисление по одной и той же формуле, ввод числа в ЭВМ или вывод результата. Возникает вопрос: нельзя ли сократить длину программы, написав такой фрагмент один раз, а выполнять его в необходимых местах программы? Оказывается, можно. Выделенная часть программы, используемая несколько раз в процессе выполнения последней, называется *подпрограммой.* Здесь мы подробно останавливаемся на определении подпрограммы потому, что она, с одной стороны, является фундаментальным понятием программирования, а с другой — знакомство с ней необходимо для последующего изучения архитектуры ЭВМ.

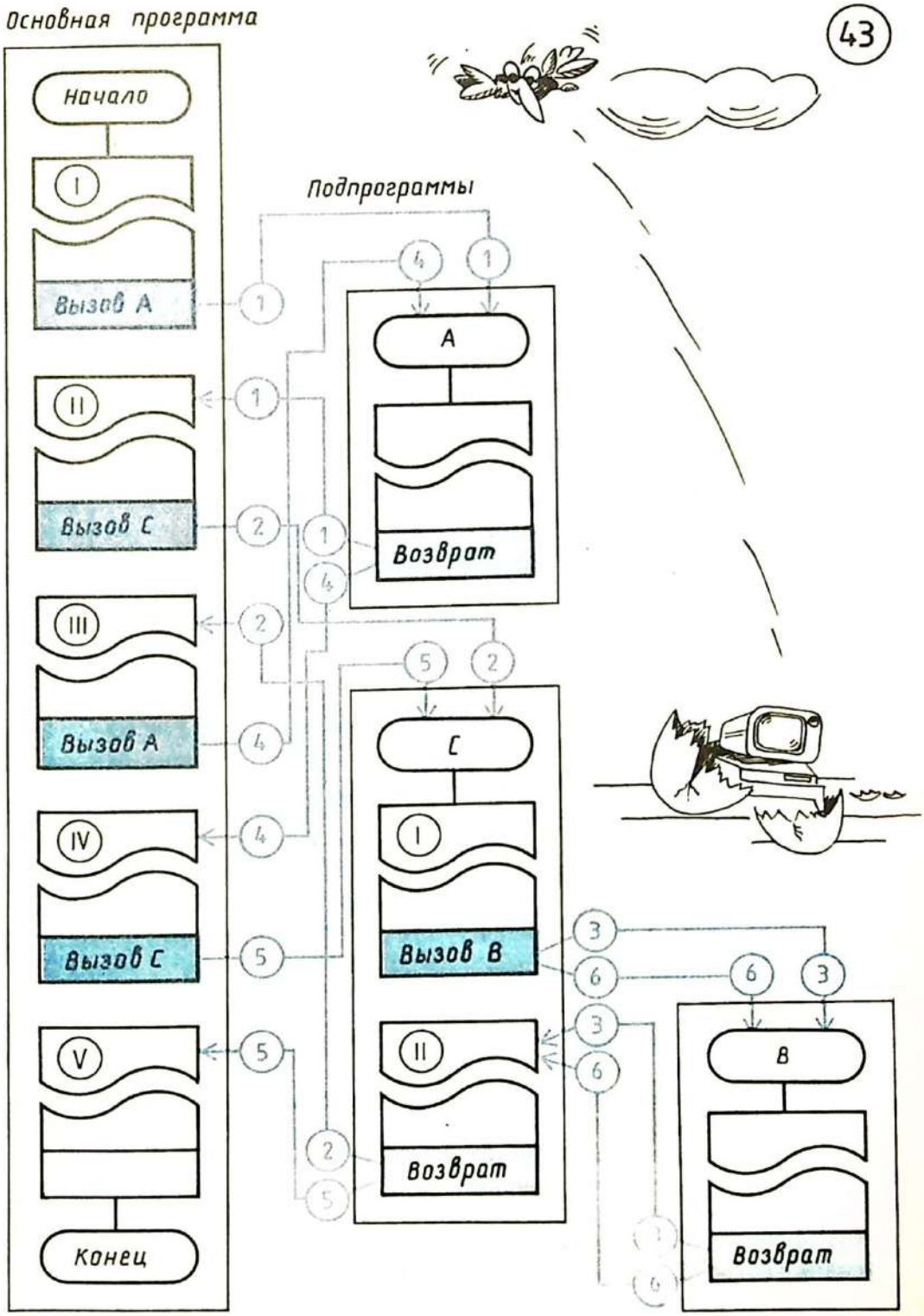
Итак, для того чтобы воспользоваться подпрограммой, достаточно в требуемом месте основного алгоритма ее *вызвать.* В результате вызова



начинает выполняться указанная подпрограмма, и в конце обязательно осуществляется *возврат* в основную программу, после чего ее выполнение продолжается с того места, которое следует за вызовом подпрограммы. На рисунке 42 показана схема двукратного обращения к подпрограмме. Естественно, что ее вызов может осуществляться несколько раз. Да и количество подпрограмм тоже не ограничивается.

Одна подпрограмма может обращаться за помощью к другой. Такое построение называется *вложением подпрограмм*. Двухуровневое вложение с несколькими подпрограммами показано на рисунке 43. Здесь последовательность выполнения отдельных этапов программы имеет следующий вид:

— основная программа (часть I);
— подпрограмма А;
— основная программа (часть II);
— подпрограмма С (часть I);
— подпрограмма В;


Основная программа
Начало
I
Вызов A
II
Вызов C
III
Вызов A
IV
Вызов C
V
Конец
Подпрограммы
A
Возврат
C
I
Вызов B
II
Возврат
B
Возврат

— подпрограмма C (часть II);
— основная программа (часть III);
— подпрограмма A;
— основная программа (часть IV);
— подпрограмма C (часть I);
— подпрограмма B;
— подпрограмма C (часть II);
— основная программа (часть V).

Осуществление вызовов подпрограмм и возврат из них в прерванную программу берет на себя микропроцессор, освобождая программиста от дополнительной работы при написании программы. Это достигается за счет архитектурных особенностей микропроцессора, к рассмотрению которых мы и переходим.

3.3. ЭВМ. ЧТО ВНУТРИ?

В школьном кабинете, студенческой лаборатории или на рабочем месте тебе в первую очередь придется осваивать персональную ЭВМ, построенную на основе микропроцессора. Отечественная промышленность и страны социализма выпускают целую серию таких машин. Каждая из них обладает определенными преимуществами. Не отдавая сейчас предпочтения конкретной ЭВМ, для знакомства с архитектурой мы будем использовать воображаемую — *гипотетическую машину,* которая обладает типичными характеристиками машин класса микроЭВМ, так как все они строятся на единой основе.

На рисунке цветной вклейки VI, *а* приведена схема такой машины. Она состоит из двух частей — *вычислительной* и *внешней. Внешним устройством* микроЭВМ является алфавитно-цифровой *дисплей* (*display* — отображение). Такое устройство еще называется *терминалом* (*terminal* — оконечное устройство). Экран дисплея служит для отображения выходной информации и может быть либо бытовым телевизором, либо специальным устройством. Клавиатура, которой снабжен дисплей, используется для ввода входной информации — программ, данных и команд управления машиной.

Для связи ЭВМ с пользователем в качестве устройства вывода применяются *принтеры* (*printer* — печатающее устройство). Они позволяют получать на бумажной ленте строки текста, а также графические изображения.

Расширение возможностей ЭВМ по накоплению и хранению данных обеспечивают *устройства внешней памяти,* представляющие собой накопители на магнитной ленте и магнитном диске.

Вычислительная часть машины состоит из набора блоков, помещенных в отдельный корпус. Центральным является *микропроцессор* — устройство, непосредственно осуществляющее преобразования информации, которая поступает от дисплея (или от других внешних устройств) через блок *ввода-вывода.* Накопление информации происходит в *блоке памяти.* Некоторые операции, такие, как включение

машины, производятся через *внешнюю панель* управления. Энергоснабжение системы осуществляется через *блок питания*. Все блоки машины соединяются между собой с помощью *шины*. Английское название шины *Bus* происходит от латинского *omnibus*, что означает «ко всем».

Для понимания того, как машина решает поставленные задачи, достаточно проанализировать устройство и работу трех блоков: блока микропроцессора, памяти и блока ввода-вывода.

Блок микропроцессора состоит непосредственно из самого микропроцессора, генератора тактовых сигналов, синхронизирующих работу микропроцессора, и вспомогательного постоянного запоминающего устройства (ПЗУ), с которого данные только считываются (ROM, *Read Only Memory*). Здесь хранится программа, предназначенная для организации начала работы машины при ее включении.

Блок памяти — устройство, состоящее из схем памяти произвольного доступа (*Random Acces Memory* — RAM); называется *оперативным запоминающим устройством* (ОЗУ). В этом устройстве хранятся данные и программы, которые могут не только читаться, но и снова записываться в эту память.

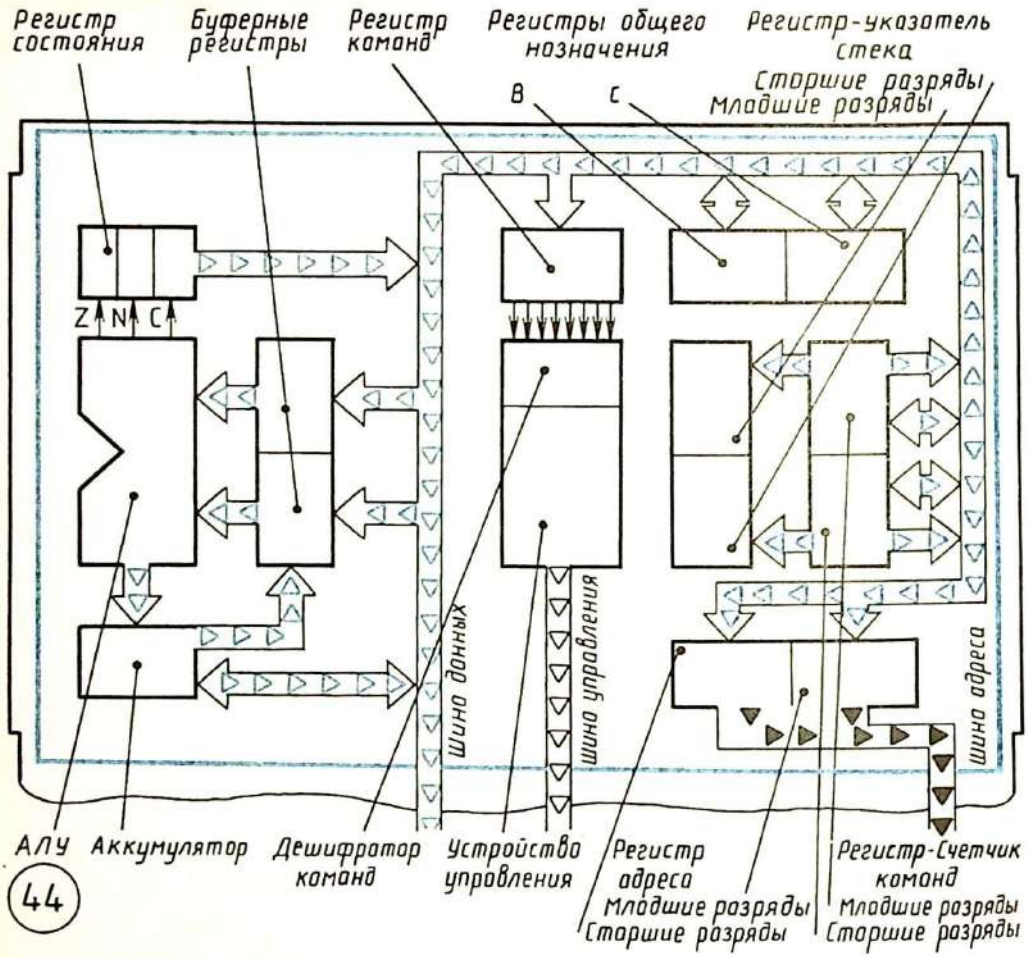
Блок ввода-вывода содержит универсальный приемопередатчик (УПП) *(Universal Receiver-Transmitter—URT)* и свой генератор. УПП служит для преобразования данных, закодированных специальным кодом и передаваемых между терминалом и блоками машины.

3.4. МИКРОПРОЦЕССОР. ДВЕ ФУНКЦИИ И ТРИ УЗЛА

Часто применение терминов «микропроцессор» и «микроЭВМ» не определяет однозначно то устройство, которое подразумевается. Из предыдущего раздела ты понял, что микроЭВМ — это целая система устройств, а микропроцессор — лишь составная часть машины.

Слово «микропроцессор» произошло от английского *processor*, что значит «выполняет действия». Процессор — устройство, выполняющее программу. Приставка *микро-* говорит о том, что процессор выполнен на микросхеме, в противоположность процессору больших ЭВМ, представляющему достаточно громоздкое устройство.

На микропроцессор (как, впрочем, и на любой процессор) возложены две основные функции: *обработка данных* и *управление машиной*. Для выполнения этих важных функций в структурной схеме микропроцессора имеются три основных устройства: арифметико-логическое (АЛУ) *(Arithmetic Logic Unit)*, устройство управления — УУ *(Control Unit)* и набор регистров *(register)*. Схема микропроцессора приведена на рисунке 44. Для передачи данных между узлами предусмотрена внутренняя шина.



3.4.1. АЛУ: сложение и сдвиг

АЛУ выполняет первую функцию микропроцессора — обработку данных, их вычисление и перемещение. Через два входа в АЛУ поступают данные, результаты которых передаются дальше через единственный выход. В АЛУ может быть преобразовано одно или два слова данных. Так, для сложения поступает на оба входа по одному слову данных. Для инвертирования используется один вход, так как операция производится над одним словом данных.

Кроме данных, на АЛУ поступают сигналы — преобразованные коды операторов программы (команды), указывающие, что необходимо делать с поступившими данными. Перечень команд, принимаемых АЛУ, фактически определяет возможности микропроцессора. Таких команд может быть много, но основными из них являются: СЛОЖИТЬ, ВЫЧЕСТЬ, ИНВЕРТИРОВАТЬ, И, ИЛИ, СДВИНУТЬ, УВЕЛИЧИТЬ НА ЕДИНИЦУ, УМЕНЬШИТЬ НА ЕДИНИЦУ. Но все эти команды реализуются двумя операциями АЛУ — сложением и сдвигом; даже команду ВЫЧЕСТЬ АЛУ выполняет с помощью операции сложения.

3.4.2. УУ: выборка и выполнение

Вторая функция микропроцессора — управление машиной, или выполнение программы, обеспечивается УУ с помощью дешифратора команд — кодов операторов программы, преобразуемых в сигналы, поступающие на все узлы микропроцессора по линиям управления.

Кроме команд программы на УУ поступают от генератора тактовые импульсы, синхронизирующие работу микропроцессора. Эта синхронизация заключается в выполнении микропроцессором двух непрерывно повторяющихся шагов: считывания очередной команды из памяти (цикл выборки — *Fetch*) и выполнения операций, предписываемых этой командой (*Execute* — цикл выполнения). Совокупность этих двух шагов образует *машинный цикл*, который соответствует почти всем процессорам.

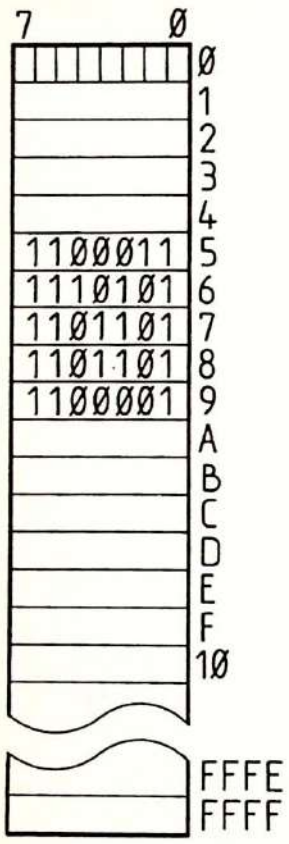
Кроме организации этого цикла и дешифрации команд УУ выполняет некоторые другие функции, в частности управляет внутренней шиной, т. е. определяет, какое устройство ЭВМ в данный момент времени может ею пользоваться, так как к ней подключено множество устройств и, естественно, необходим определенный порядок приема и передачи данных.

Изучая схему, приведенную на рисунке 44, ты, конечно, заметил, что внутренняя шина состоит из двух частей — *шины данных* и *шины адреса*. Понятие *адрес* является фундаментальным в вычислительной технике и связано с организацией памяти ЭВМ. Поэтому оставим на некоторое время изучение блока микропроцессора и обратимся к блоку памяти, чтобы затем выяснить, что такое адрес.

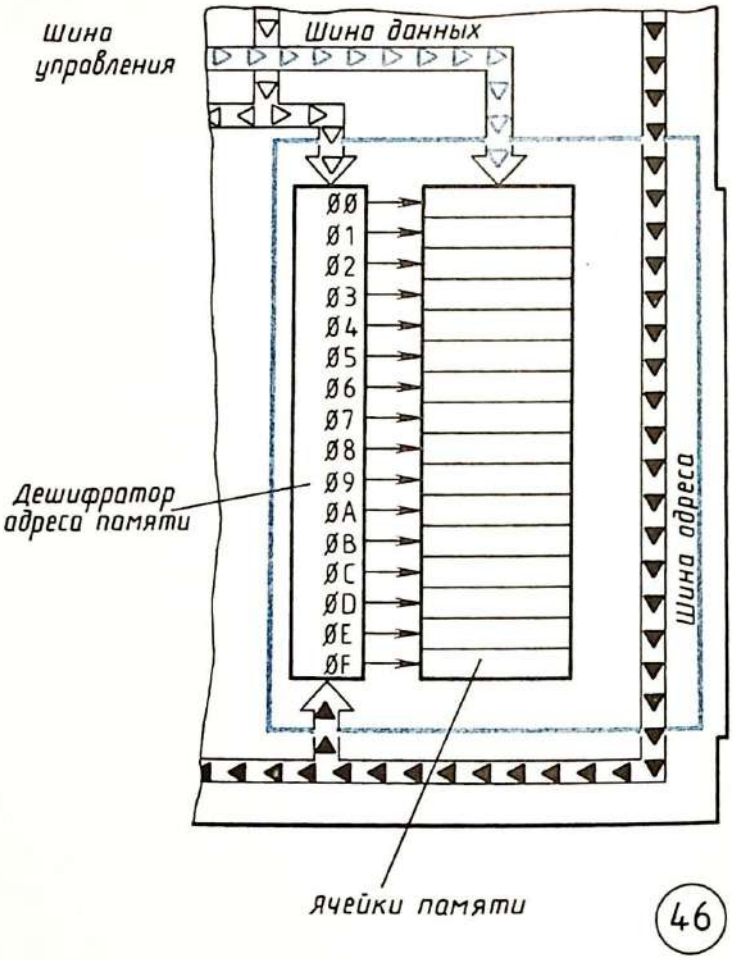
3.5. ПАМЯТЬ

Память, являясь хранилищем данных и программ, состоит из ячеек, в каждой из которых может находиться одна единица информации — бит, байт или слово. Ячейка памяти имеет две характеристики: *адрес* — числовое значение, индивидуально определяющее местонахождение ячейки в памяти, и *содержимое* — число, хранимое в данной ячейке. Очень важно не путать адрес с содержимым, хотя и то и другое является числом. Для понимания различия между адресом и содержимым можно провести аналогии: адрес памяти — адрес дома, а содержимое ячейки — жильцы дома. *Адрес ячейки* — это номер книги по каталогу библиотеки, а *содержимое ячейки* — сама книга, стоящая на одной из полок книгохранилища.

В зависимости от длины информационной единицы различают *битовые, байтовые* и *словные машины*. Для рассматриваемой нами воображаемой микроЭВМ примем размер ячейки в 8 бит, т. е. в один байт. Если для адресации ячеек принять тоже байт, то память будет содержать всего $2^8 + 1 = 257$ ячеек. Поэтому обычно для адресации определяют 16-разрядное слово, тогда емкость памяти увеличивается до $2^{16} + 1 = 65537$ ячеек. Следовательно, для шины данных достаточно



45



46

8 разрядов, а для шины адреса необходимо 16 разрядов. На рисунке 45 показана принятая нами организация памяти. Обычно память изображают так, что адресация идет от младших адресов к старшим сверху вниз, а разряды в байте нумеруются справо налево, начиная с нуля. На этом рисунке в ячейках, начиная с адреса 5, записано слово СУММА в КОИ-7. Для дальнейших примеров упростим изображение памяти нашей машины до F (16) ячеек. На рисунке 46 приведено такое изображение, где показано расположение шин и дешифраторов адреса памяти. На дешифратор по шине поступают числовые 16-разрядные значения адреса, а по линиям управления сигналы от УУ. В результате определяется конкретное местонахождение ячейки, и в зависимости от сигналов управления содержимое ячейки выдается на шину данных (операция чтения памяти), либо в ячейку помещается данное, находящееся на шине (запись в память).

А теперь снова вернемся к блоку микропроцессора и познакомимся с третьей его частью — набором регистров.

3.5.1. Регистры

Регистры — это устройства, представляющие собой отдельные ячейки внутренней быстродействующей памяти микропроцессора. Они используются для временного хранения единицы информации (в нашем случае — байта) при прохождении данных через блок микропроцессора. Количество и назначение регистров в реальных микропроцессорах различно, но восемь типов регистров практически встречаются всегда. Это регистры состояния, команд, адреса, счетчика команд, указателя стека, буферные регистры, аккумулятор и *регистры общего назначения.* Первые семь регистров называются регистрами *специального назначения.* Основной особенностью регистров является то, что большинство из них может управляться программой, в отличие от других устройств микропроцессора, например АЛУ, УУ, шины, которые недоступны программисту. Размещение регистров в блоке микропроцессора приведено на рисунке 44.

Аккумулятор. Три типа регистров окружают АЛУ, и главным среди них является *регистр-аккумулятор.* Количество разрядов аккумулятора соответствует длине адресуемой ячейки, т. е. для нашей машины это 8 бит. Рассмотрим назначение аккумулятора.

1. Он является промежуточной памятью при выполнении арифметических и логических операций в АЛУ. Любая из этих операций над двумя байтами предполагает размещение одного из них в аккумуляторе. Результат операции тоже обычно помещается в аккумулятор. При этом предыдущее содержимое его теряется.

2. Является промежуточной памятью при пересылке данных из одной части микропроцессора в другую. В этом случае сначала пересылаются данные из источника в аккумулятор, а затем из него — в приемник.

3. Изменяет свои данные непосредственно в аккумуляторе. Так, в байте, помещенном в аккумуляторе, могут быть изменены разряды на противоположные, т. е. инвертированы, либо переведены все в нуль; сдвинуты вправо или влево.

Как видно из рисунка 44, данные поступают в аккумулятор с шины или из АЛУ. Из аккумулятора они могут возвращаться обратно, но при этом используются промежуточные буферные регистры.

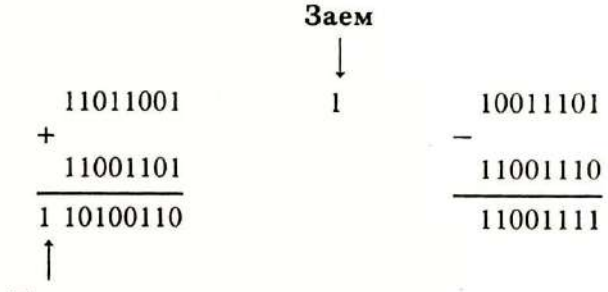
Буферы. Другой тип регистров, примыкающих к АЛУ,— буферные. АЛУ построено таким образом, что в нем отсутствует своя память. Поэтому при поступлении с шины исходных данных они сначала накапливаются в *буферных регистрах,* тем самым освобождая шину, а затем передаются в АЛУ на обработку. Таких регистров два, потому что, как мы уже знаем, в АЛУ может производиться операция над двумя байтами, например при сложении чисел. Один из буферных регистров может получать данные не только с шины, но и из аккумулятора, поэтому он еще называется *буфером аккумулятора.* Оба регистра недоступны программисту для использования.

Регистр состояния. Наиболеее интересным регистром из окруже-

ния АЛУ является регистр *состояния*. Вычислительная машина немыслима без этого регистра, что связано с известным тебе фактом наличия в любом алгоритме, а значит и в программе, логических переходов, вызванных определенными проверками результатов операций. Вспомним алгоритм суммирования пяти чисел (проверка параметра *i*). Результаты таких проверок запоминаются в регистре состояния и используются программой для осуществления перехода.

Для функции контроля результата операции в регистре состояния устанавливаются в единицу определенные разряды. Стандартными для всех ЭВМ являются три разряда:

1. Разряд *переноса* или *заем;* обозначается *C* (от английского *carry*). Этот разряд указывает, что выполненная операция завершилась переносом из 8-го разряда результата при сложении двух чисел или заемом единицы при вычитании большего числа из меньшего. Например:



2. *Нулевой разряд;* обозначается *Z (zero)*. Этот разряд показывает, что после операции во всех разрядах результата обнаружены нули, и помогает сравнивать два числа, когда надо определить, равны они или нет. Для этого вычитают одно число из другого и проверяют значение разряда *Z*. Если оно равно 1, то сравниваемые числа равны.

3. *Отрицательный разряд;* обозначается *N (negative)*. Разряд *N* указывает, что старший разряд результата по окончании операции принял значение 1 и при выполнении арифметических действий над числами в дополнительном коде результатом является отрицательное число. Разряд удобно применять тоже в случае сравнения двух чисел, когда нас интересует, какое число больше.

Для многих микропроцессоров применение регистра состояния не ограничено тремя разрядами. Оставшиеся разряды используются как индикаторы состояния некоторых дополнительных программно-аппаратных средств и различны для различных ЭВМ.

Еще одной особенностью регистра состояния является то, что он единственный из всех регистров не имеет входа со стороны шины, но отдельные разряды его, в том числе и три стандартных, могут не только считываться, но и изменяться (устанавливаться или сбрасываться) программой.

Счетчик команд. Следующие 4 регистра специального назначения (счетчик команд, регистр команд, указатель стека и регистр адреса)

используются для организации прохождения команд программы через микропроцессор. Как известно, *программа* — это последовательность команд, предназначенных для управления устройствами микропроцессора при решении задач. Чтобы соблюдался алгоритм решения, необходимо команды подавать в строгом порядке. Ответственность за сохранение этого порядка лежит на регистре *счетчика команд* — СК *(programm counter — PC)*. Относительно этого регистра надо знать следующее: его содержимым является адрес ячейки памяти, где находится команда; именно он всегда указывает на следующую команду программы, а не на ту, которая выполняется в данный момент.

Счетчик команд содержит столько разрядов, сколько адресов памяти. В нашей воображаемой микроЭВМ 16 разрядов. Поэтому СК состоит из двух частей: регистра младших разрядов и регистра старших разрядов.

Перед выполнением программы в СК необходимо поместить адрес ячейки памяти, содержащей первую команду. После извлечения команды из ячейки и передачи ее по шине данных устройство управления автоматически *увеличивает значение СК*, т. е. изменяет адрес. Новый адрес и указывает на следующую команду. Этот адрес будет храниться в СК на протяжении всего времени выполнения текущей команды. По окончании выполнения команды микропроцессор выбирает из памяти следующую команду по адресу, находящемуся в СК, и тут же вновь *увеличивает значение СК*. Этот цикл повторяется на протяжении выполнения программы.

Указанные действия микропроцессора над СК означают то, что программа хранится в последовательно расположенных друг за другом ячейках памяти. Так бывает не всегда. Часто возникает необходимость выполнить только часть программы, находящейся в другом месте памяти. Речь идет о выполнении подпрограмм.

Для организации перехода к подпрограмме текущая команда, выполняемая непосредственно перед ней, должна занести в СК адрес первой команды этой подпрограммы. Затем СК получает приращения адресов в соответствии с последовательностью команд подпрограммы. Для возврата в основную программу последняя команда подпрограммы должна модифицировать СК на соответствующий адрес основной программы.

Таким образом, для организации выполнения программ регистр СК используется как устройствами микропроцессора, так и самой программой.

Регистр команд. Выбранная из памяти команда поступает по шине данных в *регистр команд* — РК *(instruction register — IR)* после чего начинается цикл выполнения команды, первым действием которого является ее дешифрация, обеспечиваемая УУ.

В отличие от других регистров, РК только принимает данные, а посылать их на шину он не может. Число разрядов РК зависит от состава команд микропроцессора; оно может быть три и более.

Регистр адреса. Для того чтобы выбрать очередную команду из

памяти, содержимое счетчика команд передается по шине в *регистр адреса памяти* — РА *(memory adress register — MAR)*. Выход этого регистра образует шину адреса, по которой числовое значение последнего поступает в блок памяти.

В течение цикла выборки команды регистры РА и СК имеют одинаковое значение. После декодирования команды СК получает приращение, но содержимое РА не меняется.

При выполнении команды может появиться необходимость обратиться к памяти для получения данного. В этом случае выполняемая команда должна поместить в РА адрес ячейки, хранящей требуемое данное.

В некоторых случаях для организации прохождения программы значение РА формируется после вычислений, основанных на изменении значения регистра СК. После вычисления новое значение помещается в РА для выборки по этому адресу.

Указатель стека. Когда несколько страниц назад мы вели разговор о подпрограммах, то закончили его упоминанием о том, что микропроцессор берет на себя осуществление вызовов подпрограмм и возврата в прерванную программу. Эти функции микропроцессор выполняет с помощью стека и специального регистра, называемого *указатель стека* — УС *(stek pointer — SP)*. Под стек отводится область памяти, в которой временно сохраняется информация, необходимая микропроцессору для осуществления возврата из подпрограмм. Эту информацию, которая представляет из себя текущее значение счетчика команд, микропроцессор загружает в стек по команде вызова подпрограммы. Так как текущим значением счетчика команд всегда является адрес следующей команды, то сохраненный в стеке адрес указывает на команду, следующую непосредственно за командой вызова подпрограммы. Поэтому когда при возврате из подпрограммы микропроцессор восстанавливает из стека значение счетчика команд, появляется возможность продолжить прерванную основную программу с требуемой команды, т. е. с команды, следующей за вызовом подпрограммы.

Почему именно стек выбран для организации вызовов подпрограмм? Причина этого становится очевидной, если учесть, что при вызове вложенных подпрограмм, т. е. вызове подпрограммы из подпрограммы, также необходимо сохранять информацию о возврате в вызвавшую подпрограмму. Таких подпрограмм может быть достаточно много, и при последовательном возврате в основную программу, необходимо соблюдать очередность извлечения сохраненных адресов. Стек является наиболее удобным типом данных для организации такой очередности. Поддержание порядка здесь осуществляется с помощью указателя стека — регистра, в котором хранится адрес стека, определяющий первый свободный элемент в стеке. Значение этого адреса увеличивается или уменьшается на единицу при записи или извлечении из стека.

Регистр УС имеет длину 16 бит, так как хранит адреса памяти и фактически может содержать любой из 65 537 адресов нашей ЭВМ.

Поэтому когда мы говорили о неограниченности вложения подпрограмм, то имели в виду эту большую цифру. Конечно, так много вложенных подпрограмм никто не использует. Обычно их число ограничивается единицами, и область памяти, отводимая под стек, также невелика. Учитывая, что программист может увлечься и нарушить границы стека, подвергнув разрушению данные, хранящиеся в других важных областях памяти, в микропроцессоре предусмотрена возможность слежения за содержимым регистра УС. Как только адрес указателя стека превысит допустимые величины нижней и верхней границ, микропроцессор вправе остановить выполнение программы. Поэтому при пользовании стеком программист должен заранее рассчитать его максимальную длину во избежание критических ситуаций.

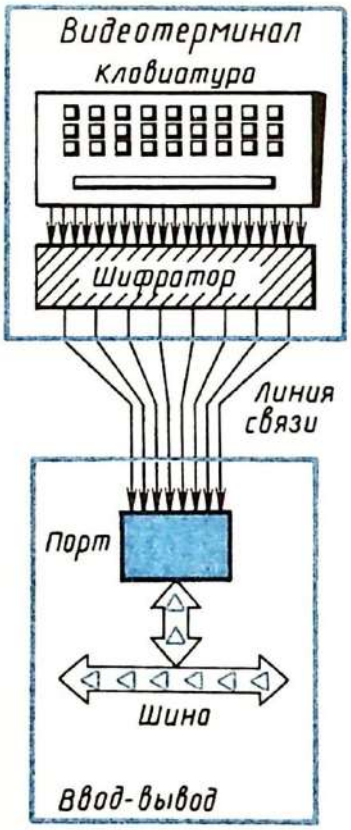
Регистры общего назначения. Для расширения возможностей по обработке данных программисту предоставляются *регистры общего назначения* — РОН *(general-purpose register)*. На рисунке 44 они обозначены как регистр B и регистр C. Число этих регистров в разных микропроцессорах различно; мы же для своей микроЭВМ ограничимся лишь двумя, которые служат для временного промежуточного запоминания данных, участвующих в обработке. Выбор конкретного регистра определяется лишь тем, какой из них наиболее удобен программисту.

Регистры B и C совместно могут выполнять функции 16-разрядного регистра, содержимое которого может, например, помещаться в РА. Таким образом, регистры B и C можно использовать отдельно или совместно.

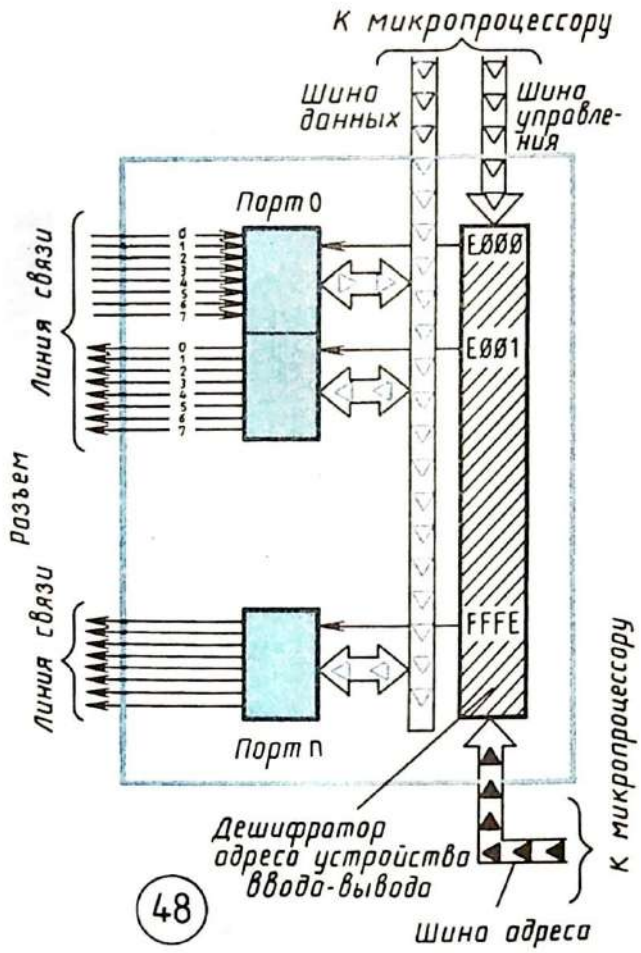
3.6. ВВОД-ВЫВОД

3.6.1. Регистр-порт

Внешние устройства ЭВМ, как правило, размещаются отдельно от блоков процессора и памяти. Для передачи данных от внешнего устройства на внутреннюю шину микропроцессора используется *интерфейс* (*interface* — средства сопряжения). Он представляет собой программные средства и электронную схему, размещаемую в блоке ввода-вывода данных. В частности, для связи с терминалом, как мы уже говорили, применяется устройство, получившее название *универсального приемо-передатчика* (УПП), соединенное с внешним устройством кабельной линией связи, где число проводов различно для разных интерфейсов. На рисунке 47 изображена схема подключения клавиатуры терминала к блоку ввода-вывода данных. Каждая клавиша такой клавиатуры соединена проводом с *шифратором* — устройством, которое преобразует сигнал, поступающий от нажатой клавиши в соответствующий ей код КОИ-7. В связи с тем, что этот код 7-битовый, необходимо минимум семь проводов для передачи кода в микропроцессор. Правда, существуют способы передачи кода символа и по меньшему числу проводов, но это сейчас для нас несущественно. Итак, код символа поступает на УПП, который запоминает его в специальном регистре, получившем название *порт* ввода-вывода (*port* — многораз-



47

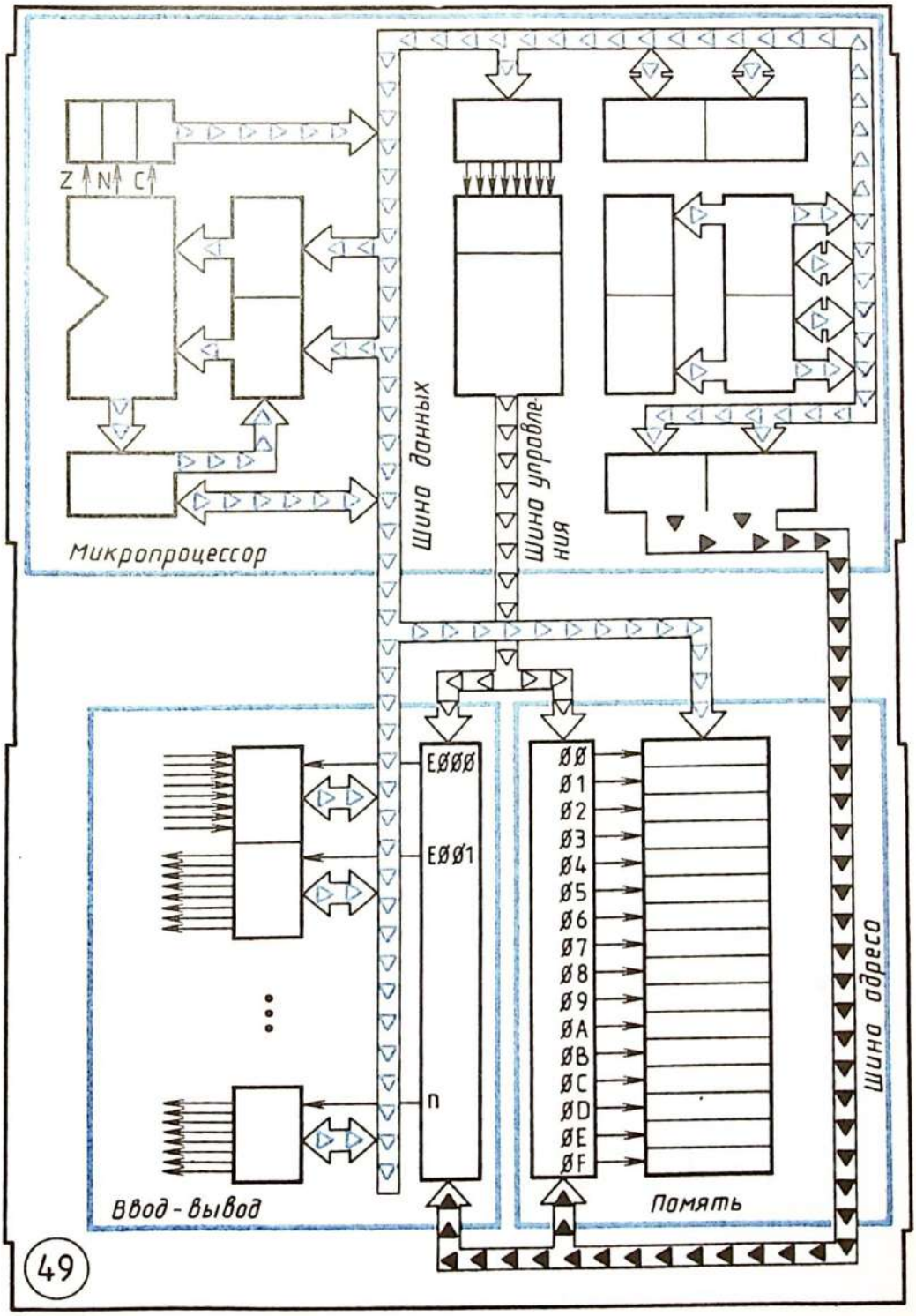


48

рядный вход или выход). Микропроцессор во время операции ввода считывает содержимое *порта* на шину и передает его, например, в *аккумулятор* для обработки. Порт ввода данных с клавиатуры — 8-разрядный. Семь бит его содержат поступающий код символа; старший бит, как правило, имеет значение 0.

Аналогично организован порт вывода данных на экран. В отличие от порта ввода, код символа поступает с внутренней шины микропроцессора и затем передается по линии связи на внешнее устройство.

Количество портов ввода-вывода в различных ЭВМ колеблется от единицы до нескольких десятков, хотя в принципе их количество может быть 2^n, где n — разрядность адресации портов. К этому моменту ты уже, вероятно, задумался над тем, как процессор отличает один порт от другого. Так вот, слово «адресация», только что упомянутое, подсказывает ответ на этот вопрос. Распространенным решением является присвоение каждому порту адреса из пространства адресов основной памяти процессора. Поэтому теоретически порт может располагаться в любом месте памяти, и даже все адресное пространство


Z
N
C
Микропроцессор
Шина данных
Шина управления
Шина адреса
EØØØ
EØØ1
n
ØØ
Ø1
Ø2
Ø3
Ø4
Ø5
Ø6
Ø7
Ø8
Ø9
ØA
ØB
ØC
ØD
ØE
ØF
Ввод - вывод
Память

(2^{16}) может быть занято портами. Практически под порты ввода-вывода отводится часть адресов, например верхние 4 килобайта — от шестнадцатеричного адреса E000 до адреса FFFE. Как и основная память, порты связаны с микропроцессором через внутреннюю шину. На рисунке 48 показана схема блока ввода-вывода, состоящего из нескольких портов. В состав блока входит дешифратор адреса порта ввода-вывода, на который заводятся шина адреса и линии управления. Дешифратор по поступившему адресу выбирает требуемый порт и в соответствии с сигналами линий управления задает ему необходимую операцию: чтение или запись на шину. Так как различаются порты ввода и порты вывода, то попытка, например, записать данные в порт ввода не даст никакого результата, поэтому надо использовать порты по назначению.

Кроме портов ввода-вывода, каждому внешнему устройству соответствуют порты управления и синхронизации. Эти порты, представляющие также 8-разрядные регистры, используются для программного управления внешними устройствами. Отдельные биты этих портов определяют, например, момент освобождения порта ввода-вывода от предыдущего данного или поступление нового данного от внешнего устройства. Таким образом, для связи с видеотерминалом необходимо четыре порта: два ввода-вывода и два порта управления, используя которые программист может в определенном месте своей программы организовать выдачу символов на экран или прием их с клавиатуры.

Рассмотренные нами три блока (микропроцессор, память и блок ввода-вывода) объединяются в единую систему (рис. 49) с помощью внутренней шины. Она состоит из 8 линий данных, по которым эти данные можно передавать в двух направлениях: к микропроцессору и от него; 16-разрядная адресная часть шины соединена с дешифраторами памяти и ввода-вывода. Линии управления предназначены для определения типа операции, которую необходимо выполнить памяти или порту ввода-вывода. Кроме этого, по внутренней шине передаются некоторые другие сигналы управления, а также питание и заземление. Однако для организации программирования они несущественны.

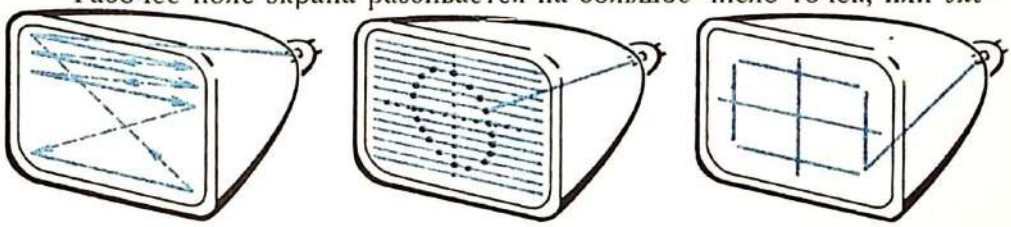
3.6.2. Терминалы

Создавая различные машины, человек стремится обеспечить в их конструкции средства, позволяющие ему общаться с этими машинами, следить за «жизнедеятельностью» их устройств. Для этого применяются всевозможные циферблаты, стрелочные индикаторы и мерцающие разными цветами лампочки. Чем сложнее машина и процессы, протекающие в ней, тем больше требуется подобных элементов индикации, так как информативность каждого из них пока еще предельно мала. Например, мигание красной лампочки ни о чем больше не говорит, как только о факте аварии. Вычислительные машины, обеспечивающие уникальную технологию переработки информации, притом с колоссальной скоростью и громадным числом важных событий, потре-

бовали и соответствующих средств общения с человеком. Нет средства более насыщенного информацией, чем слово или изображение. Поэтому уже первые ЭВМ стали снабжаться оперативными печатающими устройствами — *терминалами,* на которые естественным языком слов и цифр выводилась текущая информация. Но такие устройства представляли собой механические машинки с движущимися частями; они очень часто выходили из строя. Да и бумаги расходовалось много, так как выводимая информация важна в данный момент, а после того как человек принимает какое-то решение, она устаревает и рулоны бумаги идут в макулатуру. Наконец, скорость работы машины явно не соответствует высокому быстродействию электронных схем ЭВМ. И лишь стремительное развитие и совершенствование устройств с электронно-лучевыми трубками спасло положение; не требуя бумаги, не имея движущихся частей, такие устройства практически мгновенно отображают не только отдельные строки команд или сообщений, но и целые тексты, чертежи деталей, архитектурные эскизы, поворачивающиеся в разных плоскостях и окрашенные различными цветами.

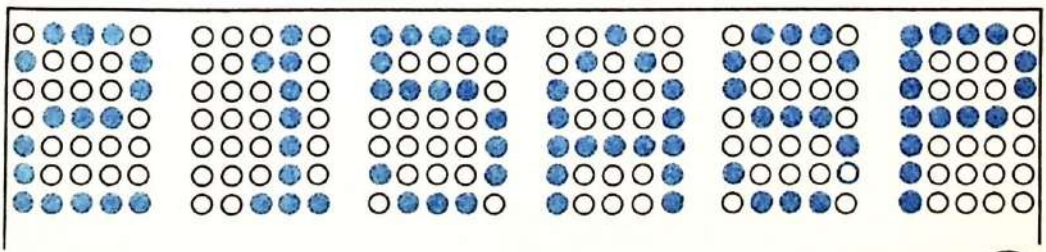
Существует много способов формирования изображений символов на экране электронно-лучевой трубки. Сейчас наиболее распространено *растровое сканирование* с формированием *точечных изображений* символов, когда электронный луч перемещается на экране по регулярной траектории, состоящей из отдельных строк (линий) растра. Обычно луч движется из верхнего левого угла экрана слева направо и сверху вниз, как показано на рисунке 50, *а*. Во время обратного хода (к началу строки) луч выключается, и его не видно.

Рабочее поле экрана разбивается на большое число точек, или *эле-*



а) Ход луча на экране *б) Растровая графика* *в) Векторная графика*

50



51

b7 → b6 → b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0		1 0 1		1 1 0	1 1 1
Биты	b4	b3	b2	b1										
	0	0	0	0					@	Ю	P	П		
	0	0	0	1					A	A	Q	Я		
	0	0	1	0					B	Б	R	P		
	0	0	1	1					C	Ц	S	C		
	0	1	0	0					D	Д	T	T		
	0	1	0	1					E	E	U	У		
	0	1	1	0					F	Ф	V	Ж		
	0	1	1	1					G	Г	W	B		
	1	0	0	0					H	X	X	Ь		
	1	0	0	1					I	И	Y	Ы		
	1	0	1	0					J	Й	Z	З		
	1	0	1	1					K	K	[	Ш		
	1	1	0	0					L	Л	\	Э		
	1	1	0	1					M	M	]	Щ		
	1	1	1	0					N	H	↑	Ч		
	1	1	1	1					0	0	—	Ъ		

ментов изображения. Путем подсвета любой комбинации элементов формируются различные изображения. В черно-белых дисплеях каждому элементу соответствует один бит в памяти; если он равен единице — элемент подсвечивается, если равен нулю — элемент темный. Символы формируются на матрице, имеющей чаще всего 5 × 7 элементов (рис. 51). Число символов в строке экрана обычно равно 80, но для некоторых дисплеев — 132. Число строк, как правило, равно 24, но может меняться от 12 до 40. Таким образом, емкость одной страницы экрана для типичного дисплея 80 × 24 = 1920 символов.

Чтобы обеспечить формирование символа, в дисплее предусмотрен *знакогенератор*. Он представляет собой полупроводниковое постоянное запоминающее устройство (ПЗУ), на которое подается код символа (например, в коде КОИ-7). По этому коду из ПЗУ считываются соответствующие сигналы, подсвечивающие требуемые элементы символа.

В графических дисплеях применяются три способа формирования изображений: *графический, векторное сканирование* и *растровое сканирование.*

Графический способ напоминает формирование символов и заключается в том, что определяются графические элементы, имеющие однозначное кодирование (рис. 52). Эти элементы хранятся в ПЗУ, называемом *графическим генератором.* Изображение строится из простых графических элементов по кодам, подаваемым на графический генератор. Именно так построено изображение шахматной партии, показанной на втором форзаце. Разрешающая способность таких дисплеев невысока.

При *векторном сканировании* луч движется только по отрезкам прямых или кривых линий, а не по строкам растра (см. рис. 50, *в*). Изображение строится из таких высвечиваемых отрезков. Предпочтительное применение векторное сканирование находит в системах проектирования деталей и строительных конструкций.

Наиболее качественное изображение получают на *растровом дисплее,* где в его построении может участвовать любая точка растра (см. рис. 50, *б*). Трудность заключается в том, что для получения изображения необходим большой объем преобразований битов, соответствующих каждой точке растра и насчитывающих десятки тысяч на одно изображение.

3.6.3. Принтеры

Как ни хорош дисплей, помогающий быстро, надежно и без лишнего шума общаться с ЭВМ, он не позволяет зафиксировать для длительного изучения и хранения наиболее важную информацию, возникающую при таком общении. Картинка на экране недолговечна: она быстро сдвигается следующим изображением, не оставляя никаких следов. Поэтому необходимость вывода на бумагу не исчезла, а, наоборот, более обострилась, особенно после того как ЭВМ «научилась» обрабатывать тексты и изображения: научилась не только считать, но и писать и рисовать, что делает с помощью устройств, называемых *принтеры.*

Современные принтеры различаются по способу формирования строки и отдельного символа. В *построчных* (параллельных) принтерах строка, предварительно набранная в памяти, печатается вся сразу. В *последовательных* принтерах строка текста образуется последовательным выводом символа за символом. *Параллельные* принтеры работают с большой скоростью (до 1200 строк в минуту), однако они имеют высокую стоимость и конструкции их громоздки.

Скорость печати последовательных принтеров невелика — до 180 символов в секунду (приблизительно 85 строк в минуту), так как необходимо перемещать на каждую позицию строки инерционную каретку и возвращать ее в исходное состояние после печати строки. И тем не менее такие устройства получают все большее распространение благодаря компактности и дешевизне.

Собственно изображение символа может быть *слитным*, как у пишущей машинки, и точечным. Последнее изображение сходно с экранным; оно также формируется на матрице точек 5 × 7. Код выводимого символа подается в знакогенератор, который формирует сигналы, управляющие иглами печатающей головки принтера, которые приходят в движение и ударяют по красящей ленте; на бумаге появляется символ.

Традиционный метод печати (удар печатающей головки по красящей ленте) используется в *ударных* принтерах, а в *безударных* принтерах формирование изображений символов не связано с механическим движением такой головки. В основе безударных принтеров, например в термографических, лежит локальный нагрев специальной теплочувствительной бумаги, приводящий к изменению ее цвета. Печатающая головка имеет небольшие элементы, которые нагреваются до 200 °С. Прижимаясь к бумаге, они формируют точечное изображение символа. Достоинством таких устройств является бесшумность в работе, но зато скорость достигает всего 30 символов в секунду.

Совершенствование принтеров основано на применении встроенных микропроцессоров, которые позволяют организовать печать строки в двух направлениях и при этом исключить потери времени на возврат печатающего механизма к исходному состоянию, а также позволяют изменять размеры и наклон символов.

3.6.4. «Записные книжки» ЭВМ

Как часто бывает, значительное техническое достижение позволяет попутно решать и другие задачи, о которых изобретатели раньше и не подозревали. Так получилось и с вычислительной техникой. Главная цель — получить быстродействующий вычислитель — была достигнута достаточно быстро. Благодаря прогрессу электроники современные процессоры способны выполнять миллионы операций в секунду. Если принять, что в среднем в одной операции участвует два байта информации, то только для обеспечения минуты работы такого процессора необходимо подать ему более 10^8 байт данных. Это громадное число,

измеряемое еще как 100 мегабайт, позволило использовать ЭВМ при работах, связанных с большими объемами информации — переработка и хранение текстов и изображений. В связи с возможностью ЭВМ поглотить за 1 с большой объем информации возникла проблема с хранением этих объемов. Оперативная память 16-разрядной микроЭВМ, как ты знаешь, ограничена 64 кбайтами. Используя дополнительные шины, можно довести этот объем до мегабайта. Дальнейший рост оперативной памяти сегодня ограничен техническими и экономическими возможностями. Поиск удобной и емкой среды для хранения информации привел к *магнитной записи*. Постепенно в качестве основных носителей в ЭВМ стали применяться магнитные ленты и диски, основными достоинствами которых являются сравнительно небольшие габариты и стоимость, большая емкость, а также большие скорости чтения и записи. Существенным преимуществом магнитных носителей является их энергонезависимость, что позволяет длительное время на ленточных катушках, пакетах дисков и дискетах без затрат энергии сохранять записанную на них информацию.

Магнитофонные записи. Магнитная лента имеет синтетическую основу, покрытую тонким слоем специального магнитного материала — ферролака. Запись на ленту основана на принципе насыщения магнитного материала в том или ином направлении, что позволяет отличать единицу от нуля. Один байт формируют поперек ленты 8 головок, соединенных в блок. При этом используется и дополнительная девятая головка для контрольного бита. Байты, записанные на ленту, группируются в блоки данных длиной до 2048 байт. Блоки объединяются в записи, а совокупность записей образует файл, снабженный заголовком.

Конструктивно *накопители на магнитной ленте* (НМЛ) больших машин имеют две съемные катушки, у которых емкость каждой ленты до 180 Мбайт, а также высокие скорости прохождения этих лент, что обеспечивает и высокую скорость передачи данных между ЭВМ и НМЛ. Для того чтобы лента быстро разгонялась и находилась в постоянном натяжении, требуются дополнительные конструктивные решения, а это делает НМЛ громоздкими и дорогими.

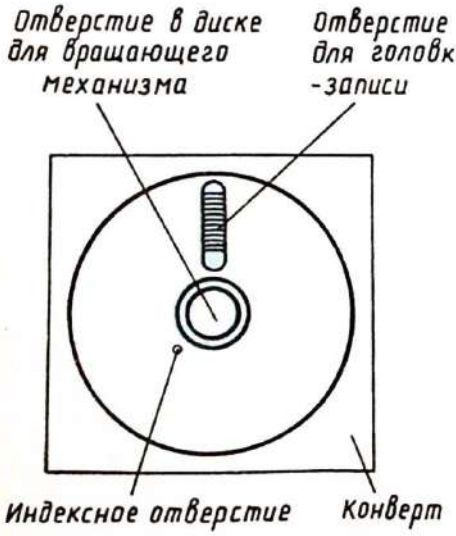
Для микроЭВМ нашли применение кассетные НМЛ. Они имеют значительно меньшую емкость и скорость обмена, но зато компактны и дешевы. Миниатюрные кассеты находятся в пластмассовом корпусе, защищающем ленту от пыли и отпечатков пальцев, а также значительно упрощающем процесс установки ленты в НМЛ. На рисунке 53 приведены некоторые характеристики типичного кассетного НМЛ. В качестве дешевых НМЛ могут быть использованы бытовые кассетные магнитофоны, подключаемые к ЭВМ.

Дисковая память. Магнитные ленты хороши для длительного хранения больших объемов информации, но сам принцип, основанный на перематывании ленты для поиска нужных данных, ограничивает их применение там, где требуется очень высокая скорость доступа к данным. В качестве устройств с малым временем *обращения* широко используются накопители на магнитных дисках (НМД). За несколько

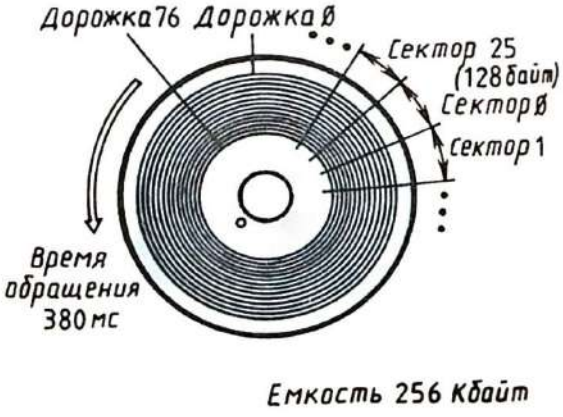
53

десятков миллисекунд удается считать с диска блок данных, имеющий в среднем объем 512 байт. А емкость некоторых дисков, собранных в пакет, достигает 100 и более Мбайт. Быстрая работа с НМД достигается за счет того, что диск, в отличие от ленты, постоянно вращается с очень высокой скоростью, а процесс записи или считывания осуществляется подвижными головками.

Существуют различные конструкции НМД, но они определяются двумя разновидностями самих дисков. *Накопители с металлическими дисками* (алюминий с ферритовым покрытием) имеют наиболее высокие характеристики, но зато более дорогие и громоздкие, требуют специальных условий для эксплуатации. Для микроЭВМ используются *накопители с гибкими дисками* из синтетического материала (майлара) с ферролаковым покрытием. Они уступают по скорости и емкости НМД с металлическими дисками, но зато имеют небольшие габариты, экономичны и дешевы, не требуют особых условий эксплуатации и при этом обладают достаточной надежностью.



а) Диск в конверте



б) Формат диска

54

Гибкий диск, называемый еще *дискет,* постоянно находится в защитном конверте (рис. 54), внутренняя поверхность которого поглощает инородные частицы. В конверте имеется прорезь для доступа к магнитной головке и индексное отверстие, отмечающее начало дорожки. Дорожек обычно насчитывается 77, одна из которых — нулевая; она служит для разметки диска, а еще две — запасные; другие 74 дорожки могут использоваться как рабочие для записи информации пользователя. Каждая дорожка разбивается на секторы, число которых равно 26. Емкость одного сектора 128 байт; этими порциями и происходит обмен информацией между НМД и ЭВМ. Обычно в НМД имеется два механизма привода гибких дисков, так что общая емкость НМД достигает 512 кбайт, но это еще не граница, так как возможности повышения емкости гибких дисков есть, и они в ближайшем будущем будут реализованы.

Программирование — зеркало разума
Дж. Вейнберг

ГЛАВА 4. ЭВМ: ПРОГРАММИРОВАНИЕ

4.1. АРХИТЕКТУРА. ПРОДОЛЖЕНИЕ ЗНАКОМСТВА

Наконец пришла пора перейти от общего определения понятия «архитектура» к выявлению его конкретных составляющих. С некоторой частью этих составляющих мы уже знакомы. Речь идет о программно-доступных регистрах, таких, как аккумулятор, регистр состояния, счетчик команд, регистр адреса, регистры общего назначения. Кроме них, понятие архитектуры включает *систему команд* и *режимы адресации*. Сюда же относятся организация памяти и ввода-вывода данных, о которых мы тоже упоминали.

4.1.1. Команды. Что понимает микропроцессор?

Когда ты говоришь своему товарищу: «Положи книгу на стол», то подаешь ему *команду*, в какой бы вежливой форме эта фраза ни прозвучала. Каковы же составные части команды?

Во-первых, это указание на то, какое необходимо произвести *действие*, во-вторых, над чем его произвести и, в-третьих, куда это действие направить. Чтобы микропроцессор мог что-то сделать, его обязательно надо попросить, передав свою просьбу в виде команды. Только есть у него одна особенность — он несмышленый. Поэтому команды, подаваемые ему, должны быть покороче и поконкретнее, а уж обращение к нему с волшебным словом «пожалуйста» произведет обратное действие, т. е. он тебя не поймет. Такой у него «ограниченный интеллект», а говоря серьезнее — набор команд, «понимаемых» микропроцессором, что и определяет все его возможности. К тому же, раз уж речь зашла о недостатках микропроцессора, единственный язык, которым он владеет, это *язык двоичных цифр*. Следовательно, команды микропроцессора должны представлять данные, которые он может распознавать. Это в первую очередь биты. Естественно, мы хотим, чтобы микропроцессор выполнял побольше команд, поэтому для их представления необходимы группы бит. Реальные микропроцессоры выполняют до 50—75 команд, а некоторые и несколько сотен, поэтому количество бит, отводимых под *код команды*, колеблется от 4 до 16.

То, что мы назвали *кодом команды*, правильнее назвать *кодом операции*, так как команда состоит из двух частей: операции и операндов. Объекты действия, или операнды, задаются адресами, или регистрами.

В общем виде команда выглядит так: в старших байтах располагается код операции, в младших — операнды. Количество операндов в команде может быть два и более или совсем ни одного, поэтому и длина команды является величиной переменной. Так как основной величиной информации является байт, то команды могут быть одно-, двух-, трехбайтовые и т. д. Конечно, для конкретного микропроцессора длина команд заранее определена и вычисляется им исходя из кода операции. Код операции в дальнейшем будем сокращенно обозначать как КОП. В международном обращении он обозначается OPC (*operation code*).

Операнды делятся на *исходные,* указывающие *данное,* над которым производится действие, и на операнды *назначения,* определяющие место размещения результата операции. Назовем их соответственно *операнд-источник* (ИСТ) и *операнд-получатель* (ПЛЧ). Таким образом, обязательной частью команды является КОП, размещаемый всегда в первом байте.

Операнды могут использоваться в различных сочетаниях, но существуют команды и без операндов. В случае же двух операндов их расположение после КОП в различных микропроцессорах отличается. Во многих системах действие команды направлено справа налево, т. е. команда записывается так:

КОП ПЛЧ, ИСТ.

Распространено и более естественное представление — слева направо, а именно:

КОП ИСТ, ПЛЧ.

В командах нашего микропроцессора мы будем использовать последнюю форму.

4.1.2. Как обратиться к данным?

Различный набор операндов определяет тот факт, что команд больше, чем существует операций. Так, например, можно переместить данные из регистра в регистр, из памяти в регистр и наоборот, из одной ячейки памяти в другую. Для этих действий применима одна операция ПЕРЕСЛАТЬ ДАННЫЕ, но для приведенного примера команд получается как минимум четыре. Тип обращения к данным в команде называют *режимом адресации.* Практически реализуются многие режимы адресации, но мы для своей системы рассмотрим лишь некоторые из них, являющиеся типичными для многих микропроцессоров.

Через регистры. Наиболее простым режимом адресации является *регистровый,* когда данные, участвующие в операции, находятся в регистрах или пересылаются между ними.

Но чтобы обратиться к регистру, его надо как-то определить. Принято нумеровать регистры, например аккумулятор — как 0, регистр B — 1, регистр C — 2 и т. д. Для нумерации семи регистров достаточно трех бит, поэтому в одном байте команды можно разместить КОП и номера двух регистров (см. табл. 26). Чтобы закодировать операцию

Таблица 26

Содержание	КОП		Регистр ИСТ			Регистр ПЛЧ		
Биты	7	6	5	4	3	2	1	0

Таблица 27

Содержание	КОП	Адрес	операнда
Номер байта	1-й байт	2-й байт	3-й байт

Таблица 28

Содержание	КОП	Регистр	Данное
Номер байта	1-й байт		2-й байт

ПЕРЕСЛАТЬ ДАННЫЕ из регистра В в регистр С, достаточно записать такой байт: 01001010, где левые биты 01 представляют КОП, следующие три бита 001 — номер регистра В, наконец 3 последних бита 010 являются номером регистра С. Нумерация, конечно, приводится в двоичном коде. Таким образом, команда разместилась всего в одном байте.

Такое построение команды имеет не только минимальную длину, но и выполняется значительно быстрее, и для ее реализации микропроцессору достаточно сделать два действия — осуществить цикл выборки команды из памяти и произвести ее выполнение за один шаг. Если цикл выборки присущ всем командам, то цикл выполнения для разных команд может быть длительностью в один или несколько шагов, поэтому минимальное время выполнения имеют команды с *регистровой адресацией*.

Прямо. Если в предыдущем режиме операнды были скрыты в регистрах, номера которых, в свою очередь, встроены в байт вместе с кодом операции, то при *прямом,* или *абсолютном,* режиме адресации операнд находится в памяти и адрес памяти указывается во втором и третьем байтах команды (см. табл. 27).

Такой режим применим для вызова подпрограмм. Последовательность 00000100 00111010 00010101 означает, что надо выполнить операцию ВЫЗОВ ПОДПРОГРАММЫ: ее код 04, а сама подпрограмма находится по адресу 3A15 (в шестнадцатеричной системе).

Непосредственно. Особенностью последнего режима адресации является возможность обращения к данным, расположенным в памяти отдельно от программы. Эти данные могут не только считываться, но на их место могут быть записаны новые. Таким образом, здесь мы имеем дело с переменными. Что же касается констант, то их удобнее хранить непосредственно в программе. При этом требуется меньше времени на выполнение команды, оперирующей с такими данными. Это происходит потому, что данное следует в программе сразу во втором байте команды (или во втором и третьем, если оно велико). В первом, как всегда, расположен КОП (см. табл. 28).

Такой *режим* получил название *непосредственной адресации*. Например, команда ЗАГРУЗКА РЕГИСТРА НЕПОСРЕДСТВЕННАЯ, имеющая код 18, может выглядеть так: 11000001 00000101. По этой команде в регистр В (три правых бита 001 первого байта) загружается число 5, непосредственно находящееся в команде (во втором байте). Длительность команды состоит из цикла выборки команды и одного шага цикла выполнения.

Прежде всего Ассемблер. Если двоичные числа являются языком машины, то командами процессора являются слова; составленные в предложения, они образуют программу — руководство к действию для ЭВМ. Человеку же очень трудно воспринимать этот набор нулей и единиц, трудно запомнить коды команд и применять их для написания программы. Первые ЭВМ программировались медленно, с большими затратами человеческого труда, и это, конечно, быстро надоело пользователям машин. Тогда они и нашли выход в применении мнемонического обозначения, т. е. сокращенной записи команд с помощью символов естественного языка. В этой более удобной форме стали представлять операции, операнды и адреса.

Для обозначения операции используют обычно три-четыре буквы из ее названия. Например, мнемоническое обозначение команды ПЕРЕСЛАТЬ ДАННЫЕ может иметь вид: ПСД. В международном выражении эта команда обозначится *MOV* (*move* — переслать). Команда СЛОЖИТЬ будет выглядеть как СЛЖ (*addition* — *ADD*). Такое обозначение команды легче усвоить, чем ее двоичный код. Символами обозначают и регистры, участвующие в операции. Чтобы сложить два числа, находящиеся в аккумуляторе и в регистре общего назначения, достаточно записать:

СЛЖ В, А.

Так же можно облегчить работу с адресами, заменив их символическими *именами*. Так, чтобы переслать данные из массива, находящегося в памяти, в аккумулятор, можно записать:

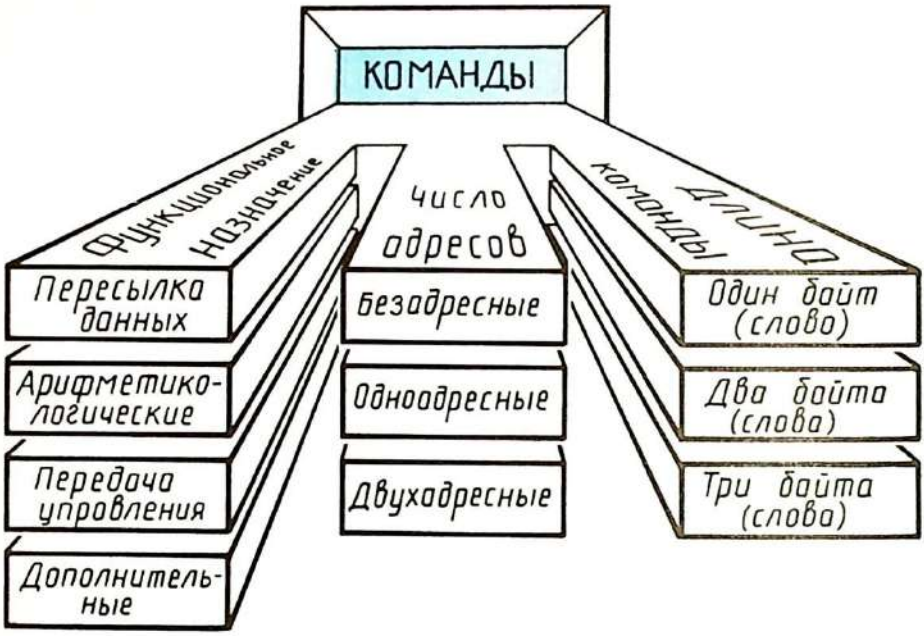
ПСД 001Е, А,

где 001Е — адрес массива. Но проще присвоить массиву имя, например МАСС, и в дальнейшем в программе использовать такую запись:

ПСД МАСС, А.

Таким образом, строка команды состоит из букв и цифр. К сожалению, за всю историю развития письменности люди не придумали ничего лучшего, чем записывать букву *О* и нуль одним и тем же символом. Теперь перед программистами возникла проблема, как отличать нуль от буквы-близнеца. В связи с этим договорились нуль обозначать символом Ø, а букву *О* оставить в первозданном виде. Этой уловке научили машину, и теперь на распечатках, выдаваемых ЭВМ, а также на экранах дисплеев применяется для обозначения нуля символ Ø.

Переход на мнемоническое представление команд образовал новый



55

язык программирования, который по существу эквивалентен машинному языку двоичных цифр, но позволяет разрабатывать программы быстрее и с меньшими затратами. Этот язык получил название *Ассемблер*, потому что программа, которая переводит тексты с него на машинный язык, была названа ассемблирующей (*assemble*), что значит «собирать, компоновать». Ассемблер (см. цветную вклейку V, *а*) относится к машинно-ориентированным языкам. Поскольку такие языки отражают структуру машины и выполняемые ею операции, то для каждого типа машин создан свой *Ассемблер*, т. е. имеются свои мнемоника команд, режимы адресации и, конечно, свой транслятор.

Основные правила программирования на Ассемблере мы рассмотрим после знакомства с командами нашей воображаемой ЭВМ.

Команды. Какие они? Всякое изучение полезно начать с классификации предмета изучения. Последуем этому совету и мы. На рисунке 55 приведена схема, позволяющая рассматривать команды по трем признакам: функциональному назначению, числу адресов и длине команды. Эти признаки определяют важнейшие свойства ЭВМ — ее возможности, объем требуемой памяти подпрограммы и скорость их выполнения. Интересно, что указанные признаки находятся в постоянном противоречии между собой. Так, увеличение числа команд расширяет возможности машины: ограничений на количество команд нет, т. е. каждый дополнительный бит в коде операции удваивает число команд. Но при этом растет длина команды и, как следствие, требуется больше памяти для размещения программы. Чем длиннее команда, тем меньше обращений к памяти и тем быстрее выполняется программа. Самой быстродействующей является безадресная однобайтовая команда, но для реализации определенной функции подобных команд

надо написать гораздо больше, чем сложных. Поэтому программисту, пишущему на *Ассемблере*, необходимо учитывать приведенные противоречия для построения эффективных программ.

Классификация команд по функциональному назначению строится исходя из основных алгоритмических признаков решения задачи. Кроме выполнения арифметических и логических операций в любом алгоритме существуют переходы от одной части процесса вычислений к другой (в зависимости от полученного на отдельном этапе результата), т. е. происходит передача *управления* от этапа к этапу. Данные, участвующие в вычислениях, постоянно извлекаются из мест хранения и передаются от этапа к этапу, от формулы к формуле, а затем вновь помещаются на хранение. Поскольку алгоритм реализуется на машине, набор команд включает средства, помогающие реализовать алгоритм в «машинной среде». К таким относятся команды, отмеченные на рисунке 55 как дополнительные.

Классификация команд по числу адресов имеет фактически две группы — безадресные и адресные команды. Последние используются при работе с памятью. При этом адрес указывается в команде явно или косвенно, путем адресации относительно регистра. Безадресные команды в качестве операндов используют регистры, содержимым которых могут быть и адреса, но здесь они расцениваются как данные, т. е. над ними могут быть произведены арифметико-логические операции.

В зависимости от числа адресов в команде ее длина меняется от одного байта для безадресных команд до трех слов для двух адресных.

Команды. Сегодня их лишь одиннадцать. Как уже говорилось, реальные микропроцессоры содержат десятки и сотни команд. В нашей небольшой микроЭВМ мы ограничимся набором всего из 11 команд, которые, однако, позволят написать нам законченную программу. Но сначала мы рассмотрим каждую команду в отдельности по следующей схеме: название команды на русском языке; название команды на английском языке; мнемоника команды; расположение команды в памяти (по байтам); возможные значения разрядов регистра состояния и КОП.

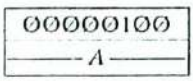
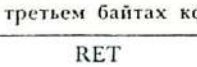
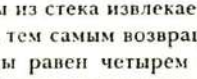
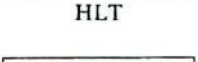
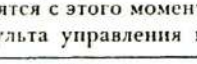
В связи с тем, что практически все *Ассемблеры* реальных микропроцессоров имеют *мнемонику команд*, составленную из букв английского названия, этот же подход мы применим и для нашей машины. При указании *значений разрядов*, принимаемых после выполнения команды в регистре состояния, будем использовать следующие условные знаки: «—», если значение разряда не меняется, «+», если значение разряда может измениться, а также 0 и 1, если значение разряда будет именно таким. Эти сведения о командах представим в виде таблицы 29.

Конечно, приведенные здесь команды не позволят реализовать даже минимальные требования, которые могут предъявляться к реальному микропроцессору. Необходимы еще команды, позволяющие выполнять вычитание, сдвиг, инвертирование разрядов, переходы по условиям отрицательного результата и переноса, безусловные переходы, операции со стеком и др. Но принцип действия этих команд во многом оди-

Таблица 29

Наименование команды (русское и английское)	Мнемоника команды и ее расположение в памяти	КОП	Значения разрядов *C*, *Z*, *N*
Команды пересылки данных			
ПЕРЕСЛАТЬ ДАННЫЕ MOV*e data*	MOV P1, P2 Ø1 P1 P2	Ø1	– + +
По этой команде данные, содержащиеся в регистре P1, копируются в регистр P2. Действие производится за один цикл выполнения.			
ЗАГРУЗИТЬ РЕГИСТР НЕПОСРЕДСТВЕННО Loa*D immediate Register*	LDR Д, P 11ØØØ P Д	11ØØØ	– + +
В регистр P загружаются (пересылаются) данные Д, находящиеся во втором байте команды. Цикл выполнения состоит из двух шагов.			
Арифметико-логические команды			
ОЧИСТИТЬ РЕГИСТР CL*ear Register*	CLR P 1ØØØØ P	1ØØØØ	ØIØ
В результате выполнения этой однобайтовой команды все разряды регистра P сбрасываются в нуль. Разряд *Z* регистра состояния устанавливается в единицу, а остальные разряды сбрасываются в нуль. Цикл выполнения состоит из одного шага.			
УМЕНЬШИТЬ НА ЕДИНИЦУ DEC*rement*	DEC P 1ØØ1Ø P	1ØØ1Ø	– + +
По этой команде из содержимого регистра P вычитается 1. Операция выполняется за один шаг. Основное назначение команды – организация счетчика выполнения некоторых действий.			
СЛОЖИТЬ С РЕГИСТРОМ ADD*ition register*	ADD P 1Ø1ØØ P	1Ø1ØØ	+ + +
В результате выполнения команды содержимое аккумулятора становится равным сумме предыдущего значения аккумулятора и содержимого регистра P. Цикл выполнения равен одному шагу.			
Команды передачи управления			
ПЕРЕЙТИ, ЕСЛИ НЕ НУЛЬ B*ranch if Not Zero*	BNZ A ØØØØØØ1Ø *A*	ØØØØØØ1Ø	– – –
По этой команде осуществляется переход на некоторую часть программы. Если результат предыдущей операции не равен нулю, т. е. значение разряда Z = Ø, то переход осуществляется к той части программы, на которую указывает адрес A, содержащийся во втором и третьем байтах команды. Если же разряд Z стал равным единице, то переход производится на команду, непосредственно следующую за последним байтом команды BNZ. Команда выполняется в цикле из двух шагов.			

Продолжение

ВЫЗВАТЬ ПОДПРОГРАММУ CALL *subroutine*	CALL A 	ØØØØØ1ØØ	– – –
Эта трехбайтовая команда выполняется в цикле из четырех шагов. Содержимое регистра счетчика команд запоминается в стеке и управление передается по адресу A, содержащемуся во втором и третьем байтах команды.			
ВОЗВРАТ ИЗ ПОДПРОГРАММЫ RET*urn*	RET 	ØØØØØ1Ø1	– – –
При выполнении этой команды из стека извлекается текущее содержимое и помещается в регистр счетчика команд, тем самым возвращая управление прерванной программе. Цикл выполнения команды равен четырем шагам.			
Дополнительные команды			
ОСТАНОВИТЬ ПРОЦЕСС HaLT *process*	HLT 	ØØØØØØØØ	– – –
Эта команда вызывает *останов* микропроцессора, т. е. машинные циклы выборки и выполнения команд не производятся с этого момента. Вывести микропроцессор из этого состояния можно только с пульта управления машины.			
ВВЕСТИ ДАННЫЕ INP*ut data*	INP П 	ØØ11ØØØØ	– – –
По этой команде из порта ввода П, имеющего адрес A, содержащийся во втором и третьем байтах команды, данное поступает в аккумулятор. Действие команды осуществляется за три шага цикла выполнения.			
ВЫВЕСТИ ДАННЫЕ OUT*put data*	OUT П 	ØØ11ØØØ1	– – –
Во время трехшагового цикла выполнения этой команды данное из аккумулятора поступает на порт вывода, имеющий адрес A, содержащийся во втором и третьем байтах команды.			

наков с выполнением приведенных команд. После знакомства с ними нетрудно представить и действие других команд. А его (знакомство) мы продолжим на примере программы, которую с помощью рисунков прогоним через нашу машину.

4.2. ПРОГРАММИРОВАНИЕ. ЗНАКОМСТВО ПРОДОЛЖАЕТСЯ

В этой главе мы уже можем начинать писать программу; осталось лишь уточнить порядок ее записи. В языке ассемблера принято следующее расположение мнемонических обозначений в строке программы:

Метка Операция Операнды Комментарий

Операция и *операнды* составляют команду, с мнемоническими обозначениями которой мы уже знакомы. *Метка* — это имя строки программы. Если операция является обязательным элементом строки, а операнды имеются в большинстве команд, то меткой снабжаются лишь те команды, местоположение которых в памяти необходимо точно знать. Например, в *командах перехода* или в *командах вызова* подпрограмм указывается адрес команды, которой требуется передать управление. Мнемоническое обозначение этого адреса и является меткой. В некоторых ассемблерах метка обязательно должна заканчиваться двоеточием «:». В нашей машине, как и во многих распространенных микропроцессорах, этот символ не требуется.

Комментарий — это краткая запись, поясняющая то действие команды, которое производится ею в данном месте программы. Комментарий присутствует только в записи на бумаге и в двоичный код не переводится, т. е. место в памяти не занимает. Поэтому писать комментарии можно в неограниченном количестве. Хотя комментарий и не является обязательным элементом команды, но он очень полезен при отладке программы и ее дальнейшем сопровождении. Комментарий может предваряться знаком «;», но наш ассемблер обходится без него.

Элементы команды, за исключением операндов, должны отделяться друг от друга пробелами. Число этих пробелов не имеет значения, однако тут существуют некоторые ограничения. Во-первых, число символов, отображаемых на экране терминала, равно 80. Для того чтобы текст программы удобно читался на экране, каждая команда должна занимать одну строку, поэтому число пробелов надо выбирать исходя из этого положения. Во-вторых, элементы команд в строках надо писать один под другим, выравнивая их левый символ по вертикали.

Разработанная на бумаге программа должна быть введена в машину. В этом нам поможет специальная обслуживающая программа — *редактор,* которая воспринимает вводимые с клавиатуры символы, выводит их на экран для контроля и одновременно формирует программу в памяти ЭВМ. С помощью редактора можно аккуратно разместить команды по строкам, а также внести исправления в программу.

А теперь приступим к написанию программы.

4.2.1. И все-таки — с чего начать?

Вспомним алгоритм суммирования пяти чисел, структурная схема которого приведена на рисунке 19. Давай попытаемся его перевести на язык ассемблера. Наши действия должны проводиться по этапам, приведенным в начале этой главы.

Первый этап — постановка задачи. Будем считать, что здесь все ясно, так как мы уже не первый раз обращаемся к задаче суммирования пяти чисел. Договоримся лишь о том, что числа будут поступать с клавиатуры терминала, а сумму необходимо будет выводить на его экран.

Второй этап — выбор и составление алгоритма. С учетом уточненной

Таблица 30

Адрес	Код	Исходный текст программы			
0000	82		CLR	C	Очистить регистр С
0001	C1		LDR	5; B	Установить счетчик
0002	05				
0003	30	INPUT	INP	00	Ввести число в аккумулятор
0004	00				
0005	E0				
0006	A2		ADD	C	Сложить с регистром С
0007	42		MOV	A, C	Переслать сумму в С
0008	91		DEC	B	Уменьшить счетчик
0009	02		BNZ	INPUT	Перейти, если не 0
000A	03				
000B	00				
000C	31		OUT	01	Вывод суммы
000D	01				
000E	E0				
000F	00		HLT		Останов

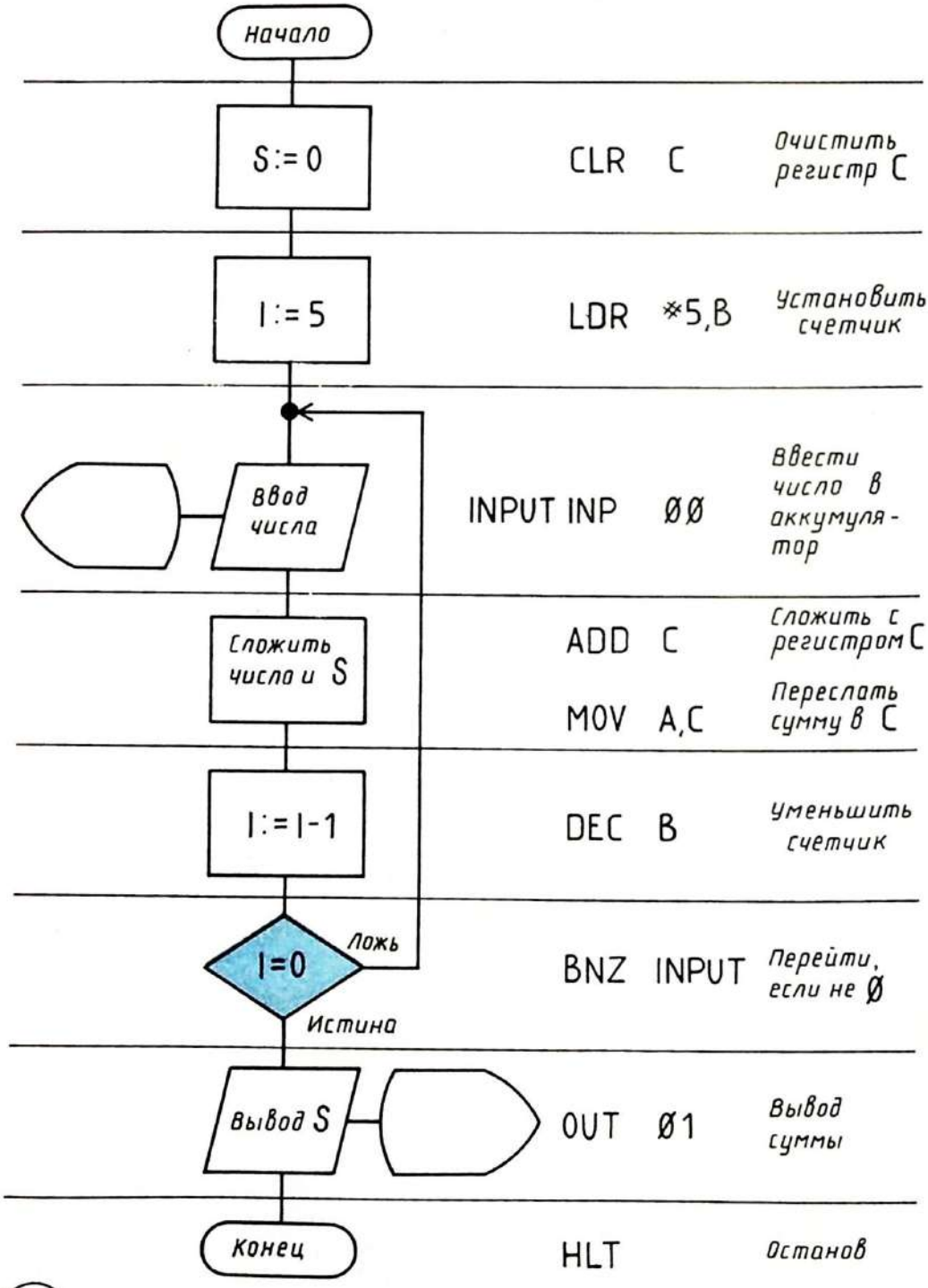
постановки задачи структурную схему преобразуем к виду, представленному на рисунке 56, *а*.

Третий этап — определение типов данных. Для простоты здесь сделаем предположение, что поступающие данные будут целыми положительными числами, а сумма их не должна превысить 127_{10}. В этом случае для операций достаточно 8-разрядного регистра, т. е. выбранным типом машинного данного будет байт. В этом месте нам необходимо сделать еще одно упрощение. Дело в том, что данные, вводимые с клавиатуры терминала, поступают в порт в коде КОИ-7, который соответствует каждой вводимой цифре, причем цифра эта десятичная. Вообще для операции сложения код КОИ-7 не подходит, поэтому практически необходимо поступающие числа преобразовывать из кода КОИ-7 в двоичный код. Эта операция реализуется сама по себе уже программой, которая по размеру не меньше нашего примера с суммированием. Поэтому, чтобы не усложнять и без того нелегкий процесс программирования, предположим, что данные поступают в порт уже преобразованными к двоичному коду. И сумму будем подавать в порт вывода также в двоичном коде.

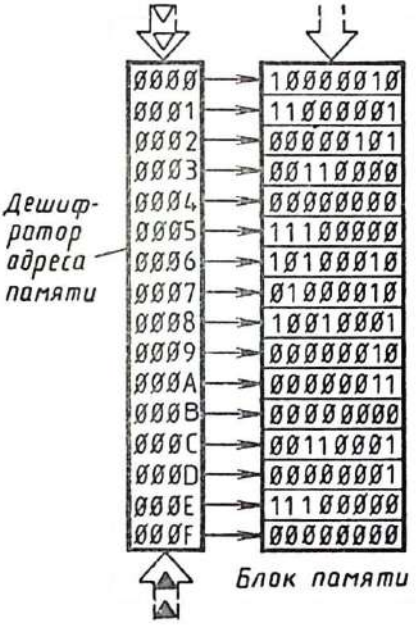
Четвертый этап — анализ архитектуры конкретной ЭВМ. Полученную программу мы будем выполнять на нашей воображаемой машине, с архитектурой которой мы только что познакомились.

Пятый этап — выбор языка программирования. Здесь пока двух мнений быть не может, так как мы знаем только язык ассемблера.

Шестой этап — кодирование. Вначале надо определиться с использованием программно-доступных ресурсов машины. Для накапливания суммы выберем регистр С. Для ведения индекса *I* воспользуемся регистром В. Регистр аккумулятора, согласно архитектурным особенностям нашего микропроцессора, применим для организации ввода-вывода данных, а также операции их суммирования. Теперь можно перейти


Начало
S:=0
I:=5
Ввод числа
Сложить числа и S
I:=I-1
I=0
Ложь
Истина
Вывод S
Конец
CLR C
Очистить регистр C
LDR #5,B
Установить счетчик
INPUT INP ØØ
Ввести число в аккумулятор
ADD C
Сложить с регистром C
MOV A,C
Переслать сумму в C
DEC B
Уменьшить счетчик
BNZ INPUT
Перейти, если не Ø
OUT Ø1
Вывод суммы
HLT
Останов

а) б)



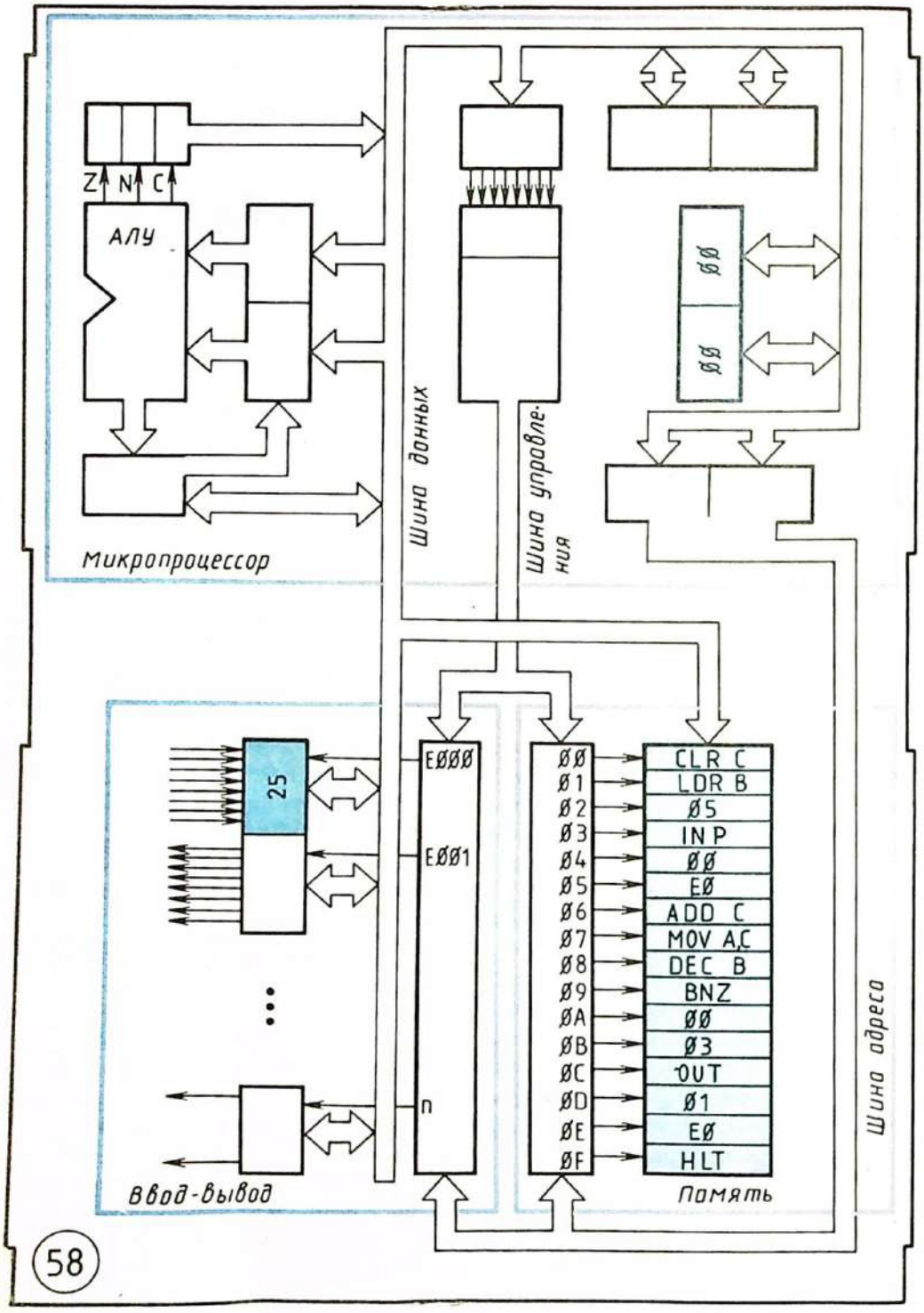
57

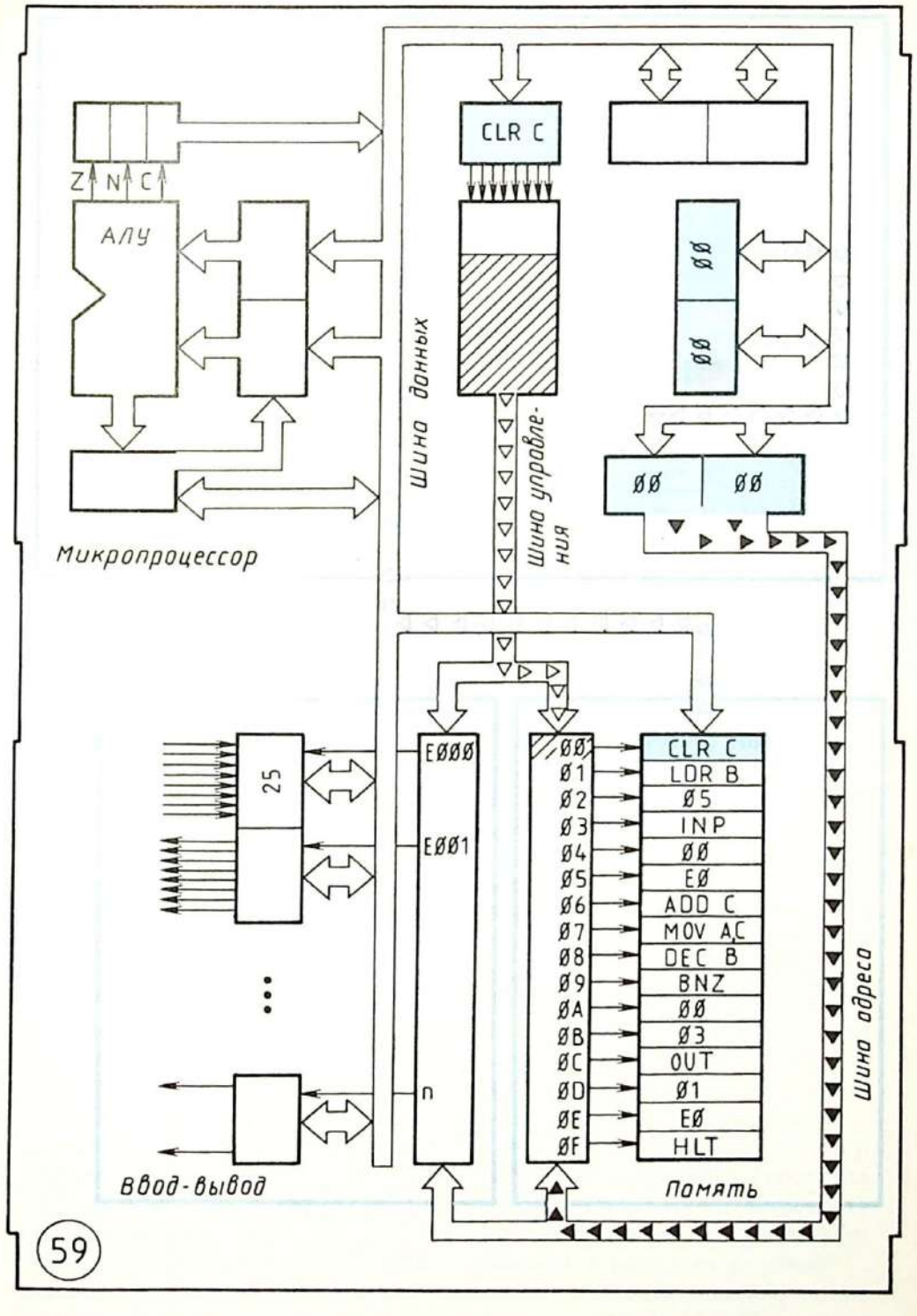
непосредственно к кодированию программы. Этот процесс заключается в замене элементов структурной схемы некоторыми командами ассемблера. Сформированная таким образом программа приведена на рисунке 56, *б*. Далее, для того чтобы получить двоичный код программы, она должна быть обработана транслятором-ассемблером. В результате в памяти разместится двоичный код программы, а на устройство печати будет выдан *листинг* программы — *документ*, содержащий исходный текст программы, а также адреса памяти и их содержимое в шестнадцатеричной системе (см. табл. 30).

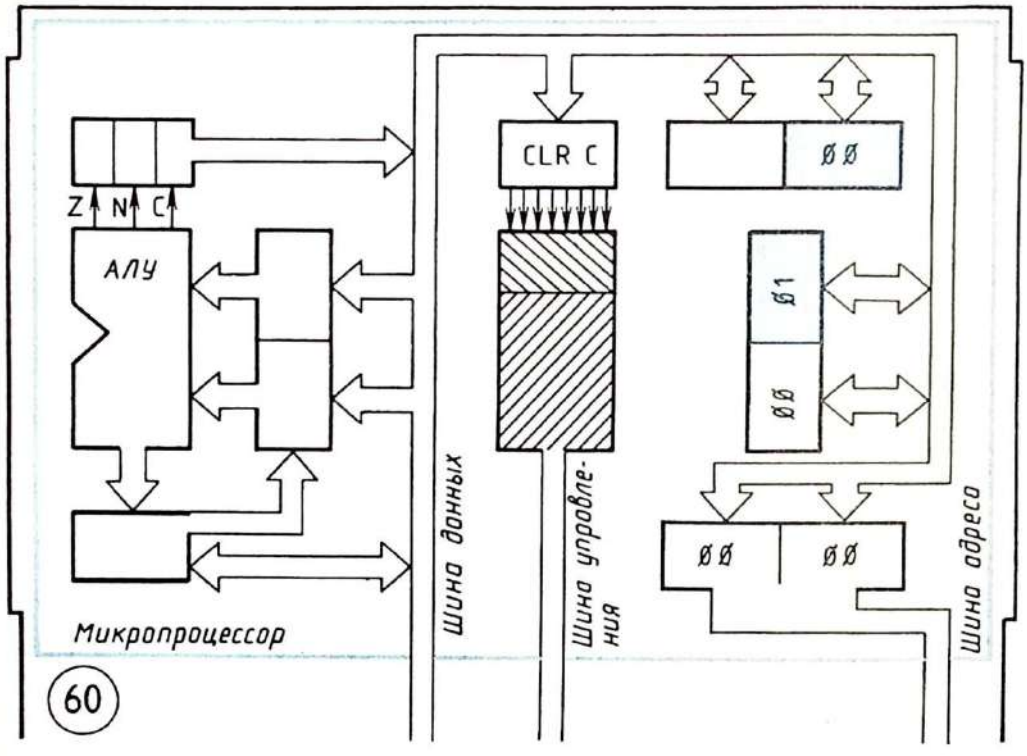
Проанализируем *листинг* программы. Наша программа займет в памяти 16 байт — адреса с 0000 по 000F. Во время обработки транслятор заменил символьное обозначение команд, регистров и адресов их двоичным эквивалентом. В команде перехода BNZ INPUT транслятор определил значение метки INPUT как адрес 0003 и разместил его во втором и третьем байтах команды.

На рисунке 57 приведено размещение программы в ячейках блока памяти микропроцессора. В каждой ячейке с адреса 0000 по 000F находится двоичный код программы. Эта последовательность нулей и единиц и будет управлять действиями микропроцессора во время выполнения программы.

Как же выполняется программа? Напомним, что для каждой команды микропроцессор производит цикл выборки ее первого байта и размещение его в регистре команд. В результате дешифрации команды устройством управления начинается цикл ее выполнения, который в зависимости от типа команды может состоять из одного или нескольких шагов (для команд HLT вообще никакие шаги не предпринимаются). На последующих рисунках отмечены регистры, ячейки


Z N C
АЛУ
Микропроцессор
Шина данных
Шина управления
ØØ
ØØ
Шина адреса
25
EØØØ
EØØ1
n
Ввод-вывод
ØØ CLR C
Ø1 LDR B
Ø2 Ø5
Ø3 INP
Ø4 ØØ
Ø5 EØ
Ø6 ADD C
Ø7 MOV A,C
Ø8 DEC B
Ø9 BNZ
ØA ØØ
ØB Ø3
ØC OUT
ØD Ø1
ØE EØ
ØF HLT
Память


Z
N
C
АЛУ
Микропроцессор
CLR C
Шина данных
Шина управле-
ния
ØØ
ØØ
ØØ
ØØ
Шина адреса
25
EØØØ
EØØ1
n
ØØ
Ø1
Ø2
Ø3
Ø4
Ø5
Ø6
Ø7
Ø8
Ø9
ØA
ØB
ØC
ØD
ØE
ØF
CLR C
LDR B
Ø5
INP
ØØ
EØ
ADD C
MOV AC
DEC B
BNZ
ØØ
Ø3
OUT
Ø1
EØ
HLT
Ввод-вывод
Память



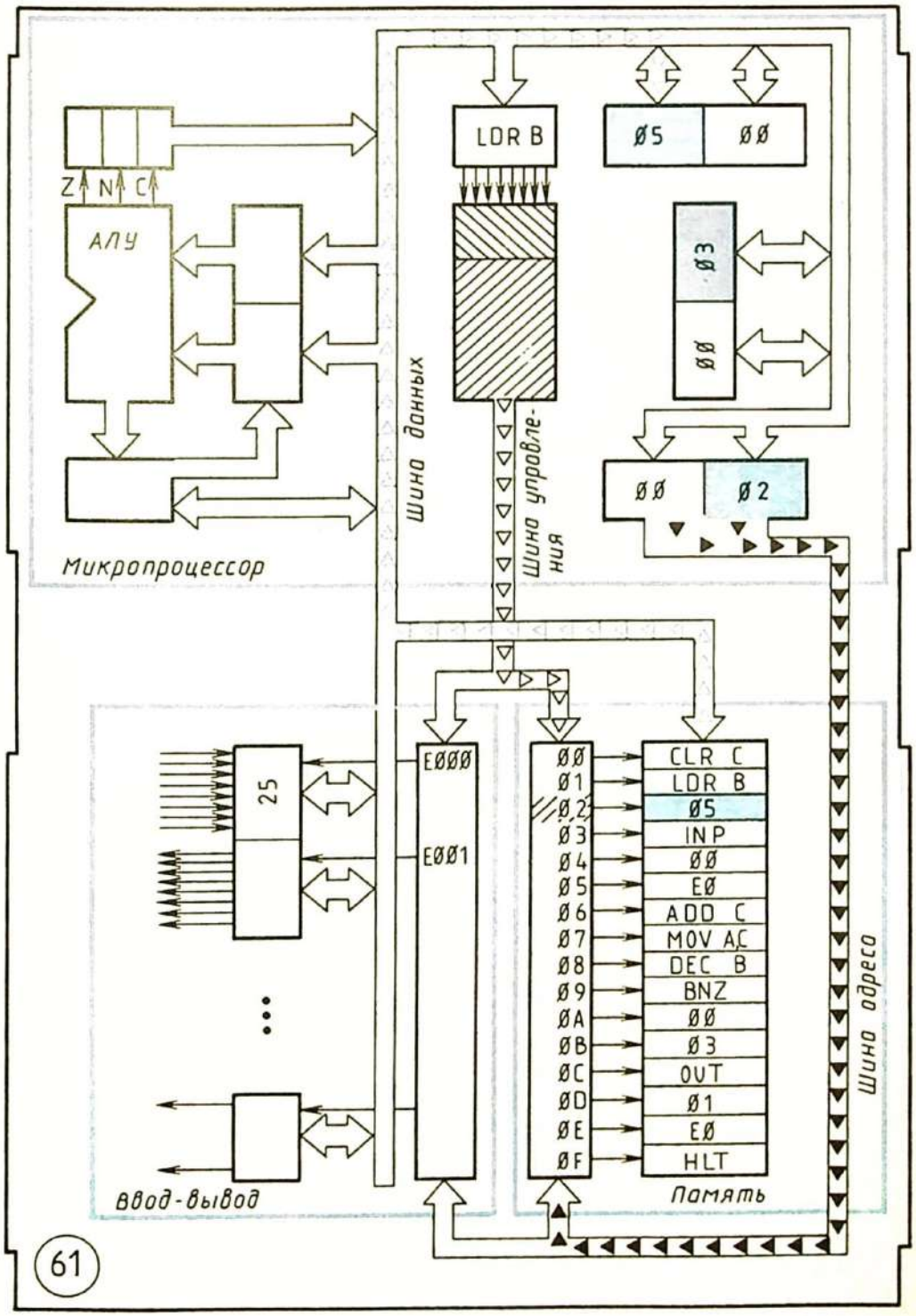
памяти и участки шины, которые активизируются в момент выполнения команды. И еще надо сказать вот о чем. Для большей наглядности содержимое регистров и ячеек памяти на этих рисунках приводится в символьном виде в шестнадцатеричном коде, но при этом мы помним, что в действительности все выглядит так, как на рисунке 57. А теперь посмотрим иллюстрации выполнения программы.

На рисунке 58 показана ЭВМ перед началом выполнения программы. Программа загружена в память в адреса ØØØØ, в порт ввода поступило число 25, регистр СК указывает на начало программы — адрес ØØØØ.

Работа начинается с выполнения цикла выборки первой команды. В регистр адреса из регистра СК загружается адрес команды — ØØØØ (рис. 59), который поступает по шине адреса в дешифратор адреса памяти. Из определенной таким образом ячейки команда поступает по шине данных в регистр команд.

Затем осуществляется цикл выполнения выбранной команды CLR C (рис. 60). В регистре С все разряды сбрасываются в нуль, а регистр СК получает приращение на 1, указывая теперь на следующую команду. Операции производятся устройством управления на основе дешифрации команды — определения кода операции и операнда.

Выборка следующей команды — LDR Ø5, B осуществляется аналогично рисунку 59, а в результате ее выполнения данные из ячейки с адресом ØØØ2 загружаются в регистр В, при этом регистр СК увеличивается на 2, так так команда была двухбайтовая (рис. 61).


Z N C
АЛУ
Микропроцессор
LDR B
Ø5 ØØ
Ø3
ØØ
ØØ Ø2
Шина данных
Шина управле-ния
Шина адреса
25
EØØØ
EØØ1
ØØ CLR C
Ø1 LDR B
Ø2 Ø5
Ø3 INP
Ø4 ØØ
Ø5 EØ
Ø6 ADD C
Ø7 MOV AC
Ø8 DEC B
Ø9 BNZ
ØA ØØ
ØB Ø3
ØC OUT
ØD Ø1
ØE EØ
ØF HLT
Ввод-вывод
Память

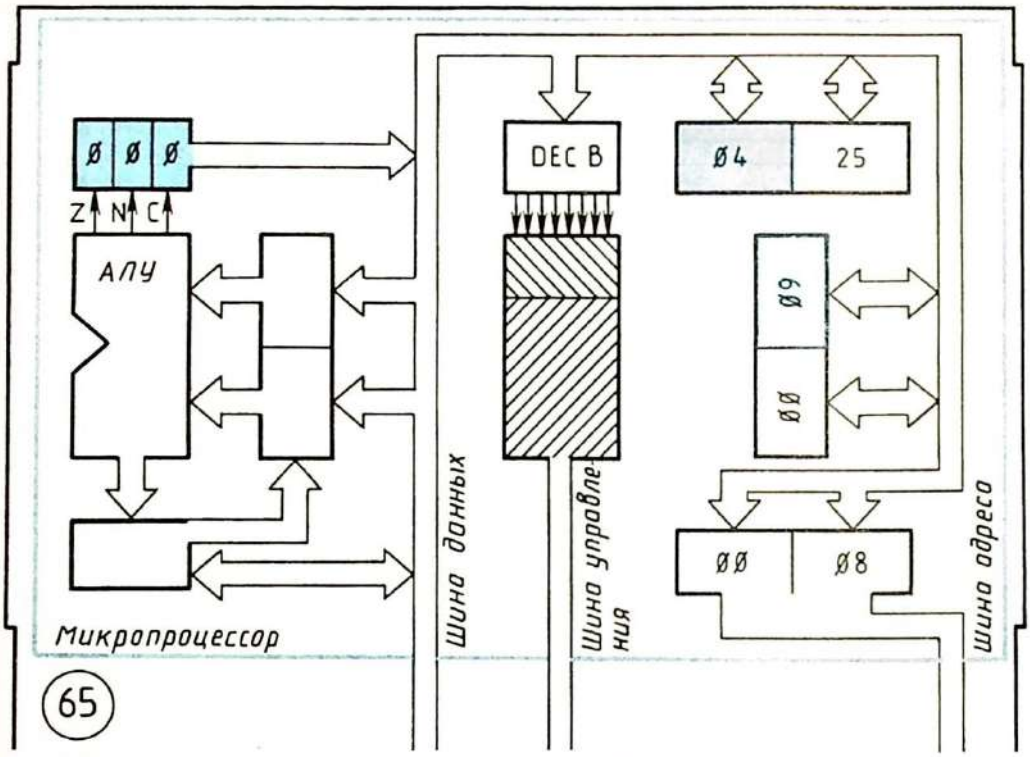
Z N C
АЛУ
25
Микропроцессор
INP
Ø5 ØØ
Ø6
ØØ
EØ ØØ
Шина данных
Шина управления
EØØØ
EØØ1
n
ØØ CLR C
Ø1 LDR B
Ø2 Ø5
Ø3 INP
Ø4 ØØ
Ø5 EØ
Ø6 ADD C
Ø7 MOV A,C
Ø8 DEC B
Ø9 BNZ
ØA ØØ
ØB Ø3
ØC OUT
ØD Ø1
ØE EØ
ØF HLT
Шина адреса
Ввод-вывод
Память

Ø Ø Ø
Z N C
АЛУ
ØØ
25
25
ADD C
Ø5 ØØ
Ø7
ØØ
ØØ Ø6
Шина данных
Шина управле-ния
Шина адреса
Микропроцессор

63

Ø Ø Ø
Z N C
АЛУ
ØØ
25
25
MOV A,C
Ø5 25
Ø8
ØØ
ØØ Ø7
Шина данных
Шина управле-ния
Шина адреса
Микропроцессор

64



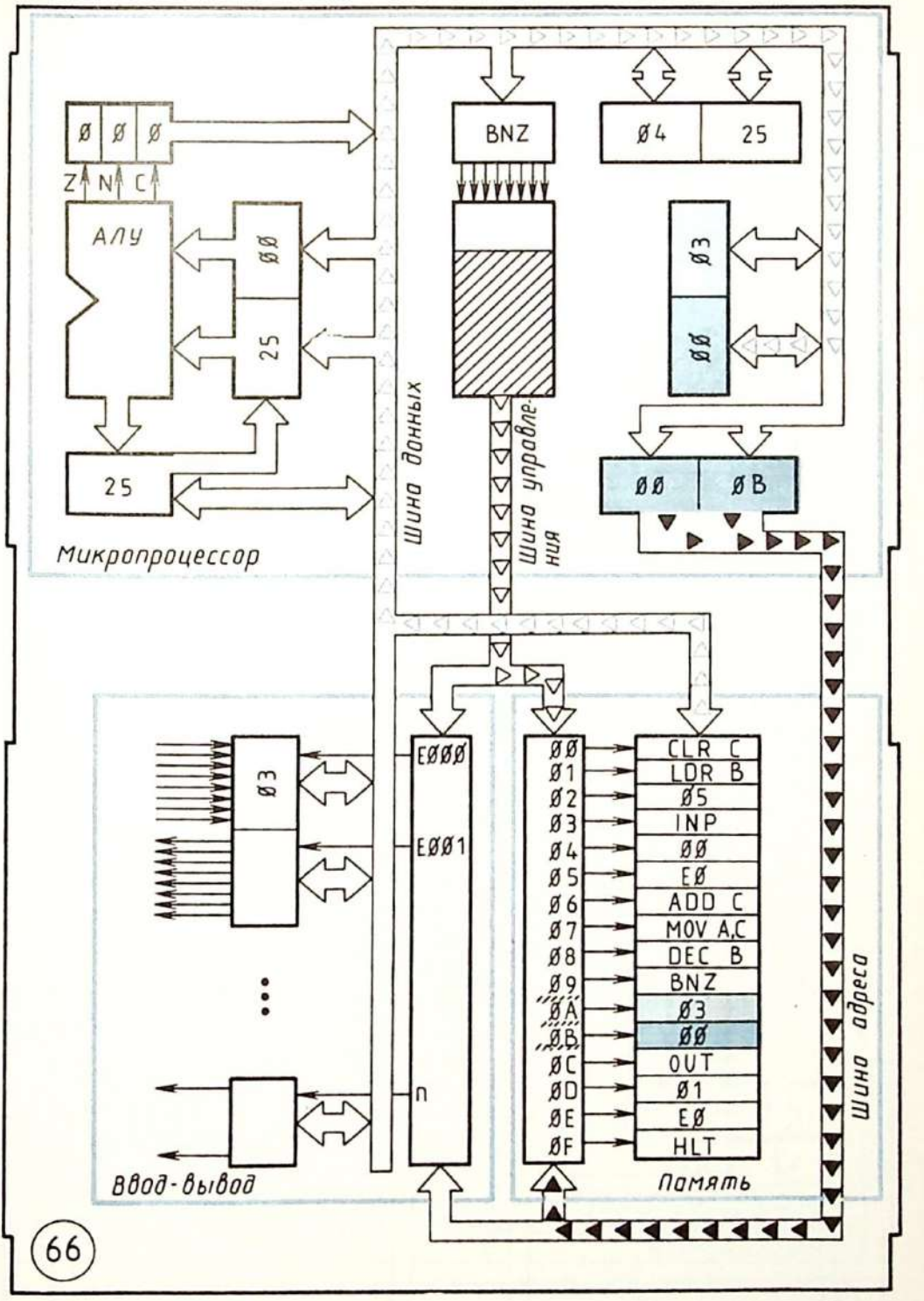
Вновь производится выборка команды (теперь это INP) по той же схеме, что и две предыдущие. Цикл выполнения этой команды сложнее. За два шага в регистр адреса — сначала в младшие разряды, а затем в старшие — загружается адрес порта ввода. На третьем шаге число 25 из порта поступает в аккумулятор по шине данных (рис. 62). Наконец, регистр СК получает приращение на 3, так как команда была трехбайтовая.

Снова цикл выборки, и в регистре команд размещается код команды ADD C. В цикле выполнения этой команды данные из аккумулятора и регистра С через буферные регистры поступают в АЛУ, откуда сумма этих данных помещается в аккумулятор (рис. 63).

Следующей выбранной командой является MOV A, C. В цикле выполнения этой команды данные из аккумулятора копируются в регистр С (рис. 64).

И вновь цикл выборки, а за ним цикл выполнения, но теперь уже команды DEC B. В результате содержимое регистра В уменьшается на 1 (рис. 65).

Выполнение команды BNZ, выбранной на следующем шаге, имеет интересную особенность. Во время первого шага цикла выполнения анализируется значение разряда Z регистра состояния, и так как это значение равно Ø, то в регистр СК начинает загружаться адрес перехода — его младший байт. На втором шаге адрес формируется окончательно — загружается старший байт (рис. 66). И в цикле следующей выборки содержимое СК, как обычно, поступает в регистр адреса, и по

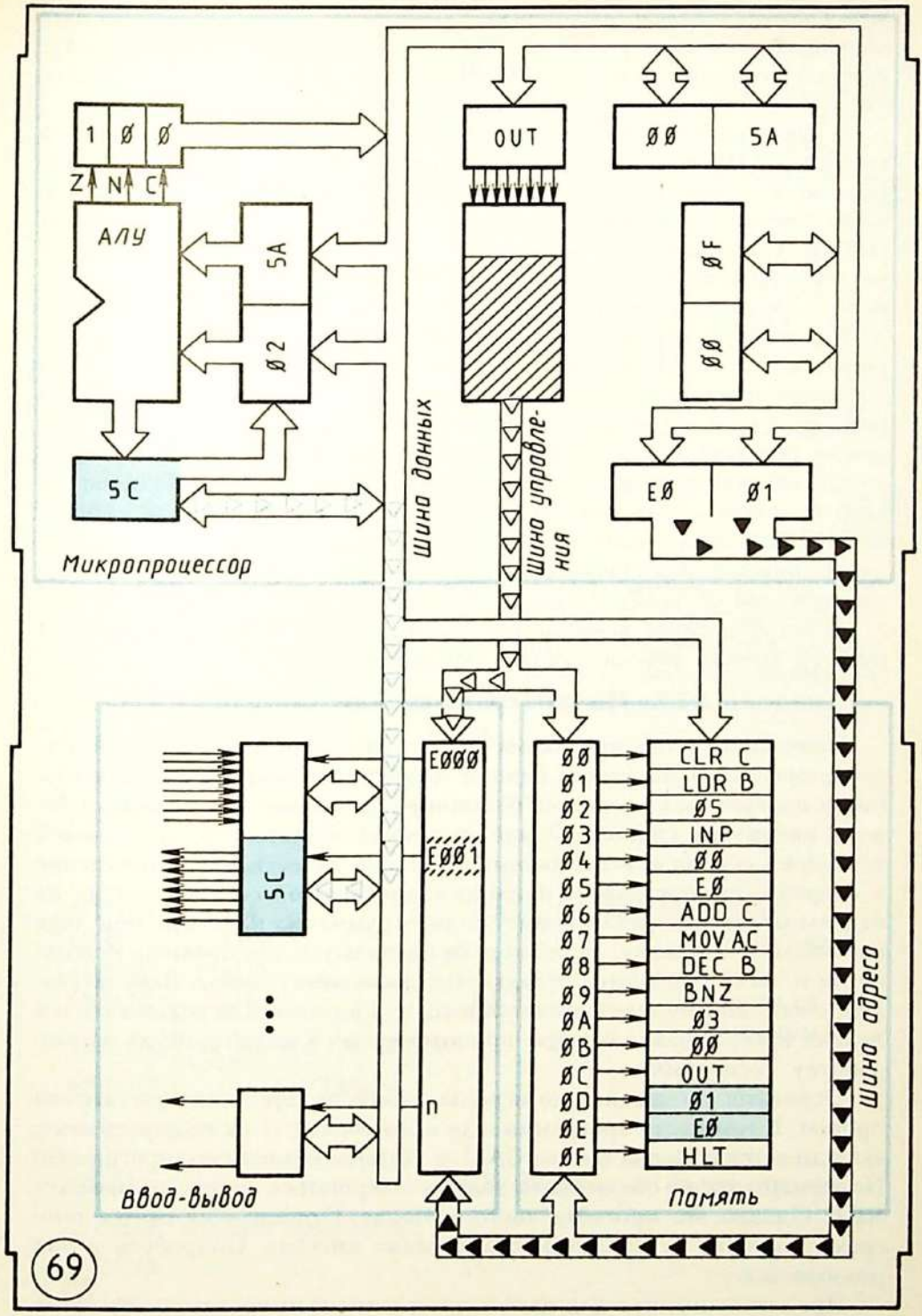

Ø Ø Ø
Z N C
АЛУ
ØØ
25
25
Микропроцессор
BNZ
Ø4 25
Ø3
ØØ
ØØ ØB
Шина данных
Шина управления
Шина адреса
EØØØ
EØØ1
n
Ø3
Ввод-вывод
ØØ CLR C
Ø1 LDR B
Ø2 Ø5
Ø3 INP
Ø4 ØØ
Ø5 EØ
Ø6 ADD C
Ø7 MOV A,C
Ø8 DEC B
Ø9 BNZ
ØA Ø3
ØB ØØ
ØC OUT
ØD Ø1
ØE EØ
ØF HLT
Память

Ø Ø Ø
Z N C
АЛУ
25
Ø3
28
BNZ
Ø3 28
Ø3
ØØ
ØØ ØB
Шина данных
Шина управле-
ния
Шина адреса
Микропроцессор

67

1 Ø Ø
Z N C
АЛУ
5A
Ø2
5C
BNZ
ØØ 5A
ØC
ØØ
ØØ Ø9
Шина данных
Шина управле-
ния
Шина адреса
Микропроцессор

68


1 Ø Ø
Z N C
АЛУ
5A
Ø2
5C
Микропроцессор
OUT
ØØ 5A
ØF
ØØ
EØ Ø1
Шина данных
Шина управле-ния
EØØØ
EØØ1
n
5C
Ввод-вывод
ØØ CLR C
Ø1 LDR B
Ø2 Ø5
Ø3 INP
Ø4 ØØ
Ø5 EØ
Ø6 ADD C
Ø7 MOV AC
Ø8 DEC B
Ø9 BNZ
ØA Ø3
ØB ØØ
ØC OUT
ØD Ø1
ØE EØ
ØF HLT
Память
Шина адреса

этому адресу извлекается команда. Ею вновь оказывается команда INP, хранящаяся по адресу ØØØ3. Таким образом, наша машина приступает ко второму шагу цикла программы, заключающемуся во вводе числа и его суммировании с ранее введенными.

В цикле выполнения команды INP вводится число, например 3, по командам ADD и MOV это число суммируется с предыдущим числом, а затем 25 и сумма помещается в регистр С. Еще на 1 уменьшается значение счетчика шагов цикла в регистре В после выполнения команды DEC B. Поскольку в регистре В еще не Ø, то значение разряда Z в регистре состояния не изменится – останется равным Ø (рис. 67), и начнется следующий шаг цикла программы – загрузка команды INP.

Наконец, после очередного выполнения команды DEC B, значение регистра В становится равным Ø. Тут же в регистре состояния разряд Z примет значение 1. В цикле выполнения команды BNZ после анализа разряда Z произойдет не выборка адреса перехода, а обычное приращение регистра СК на длину команды (рис. 68).

Следующей выбранной командой будет команда OUT. На первом и втором шагах цикла регистр адреса загрузит адрес порта ввода – ЕØØ1. На третьем шаге цикла из аккумулятора в порт вывода передастся накопленная сумма, предположим, что она равна 5С (в шестнадцатеричной системе). Результат этих действий виден на рисунке 69.

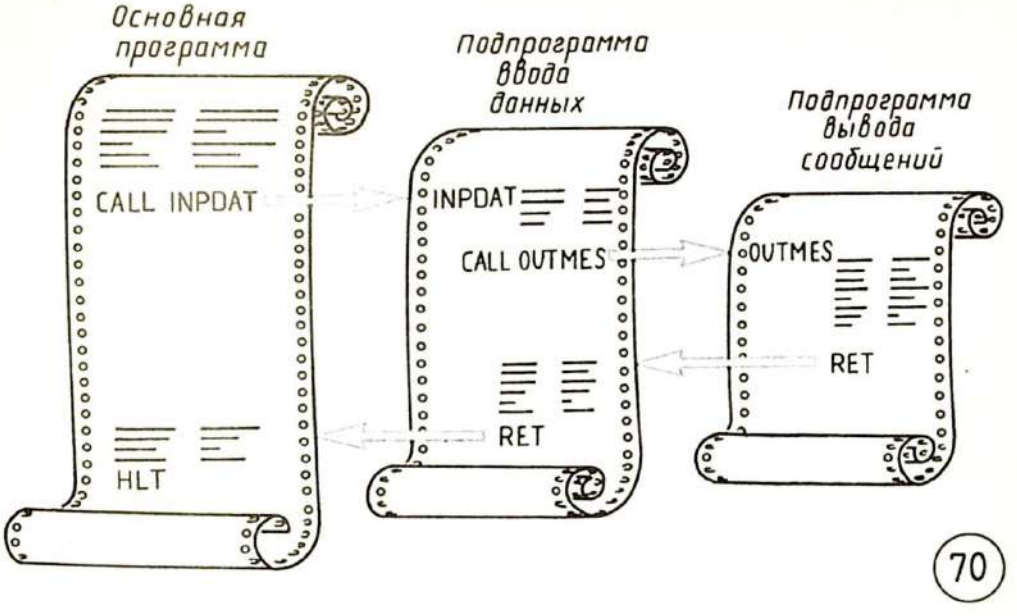
И вот последняя команда программы – HLT. После ее выборки в регистр команд работа машины останавливается.

4.2.2. Подпрограммы, или модули

Написанная нами программа очень мала, поэтому можно считать, что разработали мы ее без единой ошибки. В практической деятельности программы получаются большими и, к сожалению, ошибок избежать никому не удается. Одной из причин является то, что человек теряется в обилии команд, данных и адресов. Естественное стремление к сокращению программы привело к дроблению всего алгоритма на отдельные этапы, реализуемые подпрограммами, или, как они еще называются, *модулями*. Такие модули получаются, как правило, небольшими и легко поддаются отладке – исправлению ошибок. Замечательным свойством модулей является и то, что их можно использовать и в других программах, сокращая при этом время и затраты труда на разработку последних.

Основные соглашения по использованию подпрограмм достаточно просты. В том месте программы, где надо обратиться к подпрограмме, записывается команда вызова CALL с указанием имени подпрограммы. Подпрограмма же обязательно должна завершаться командой возврата RET. Однако эта простота чисто внешняя. Организация связей программ, скрытая от программиста, гораздо сложнее. Попробуем в ней разобраться.

Предположим, что для нашей программы суммирования исходные данные необходимо считывать с магнитного диска. Функции работы с МД выполняет подпрограмма INPDAT (от INP*ut* DAT*a* – ввод дан-



70

ных). В свою очередь подпрограмма INPDAT вызывает другую подпрограмму, имя которой OUTMES(от OUT*put* MES*sage*—вывод сообщения). С ее помощью на печатающее устройство выводятся сообщения о фактах считывания очередного данного. Схема этого взаимодействия программ показана на рисунке 70. Оттранслируем отдельно каждую программу и получим листинги.

Листинг для модуля INPDAT:

```
ØØØØ    ...     INPDAT    ...
  .       .               .
  .       .               .
  .       .               .
ØØØB    Ø4                CALL OUTMES  Вызвать
ØØØC    ØØG                            подпрограмму
ØØØD    ØØG
ØØØE    Ø5                RET          Возврат из под-
                                       программы
```

Листинг для модуля OUTMES:

```
ØØØØ ...      OUTMES    ...
  .    .                  .
  .    .                  .
  .    .                  .
ØØØ8 Ø5                  RET  Возврат из подпрограммы
```

Листинг основной программы:

```
ØØØØ    ...      ...
  .       .        .
  .       .        .
  .       .        .
ØØØ3    Ø4       CALL INPDAT  Вызвать подпрограмму ввода
```

```
ØØØ4     ØØG
ØØØ5     ØØG
  .        .       .
  .        .       .
  .        .       .
ØØØF     ØØ      HLT          Останов
```

Обрабатывая тексты программ, ассемблер обнаружил неопределенные символы — INPDAT и OUTMES. Действительно, в командах эти имена используются в качестве операндов, но ни разу они не встречаются в качестве метки. Поэтому транслятор не может поставить этим именам в соответствие конкретные адреса. В соответствующие байты команд записываются нули, а эти байты помечаются символом G (*Globe* — глобальный), указывая, что эти имена являются внешними для модуля. В таком виде программы использоваться пока не могут, а как быть дальше, мы выясним немного позже.

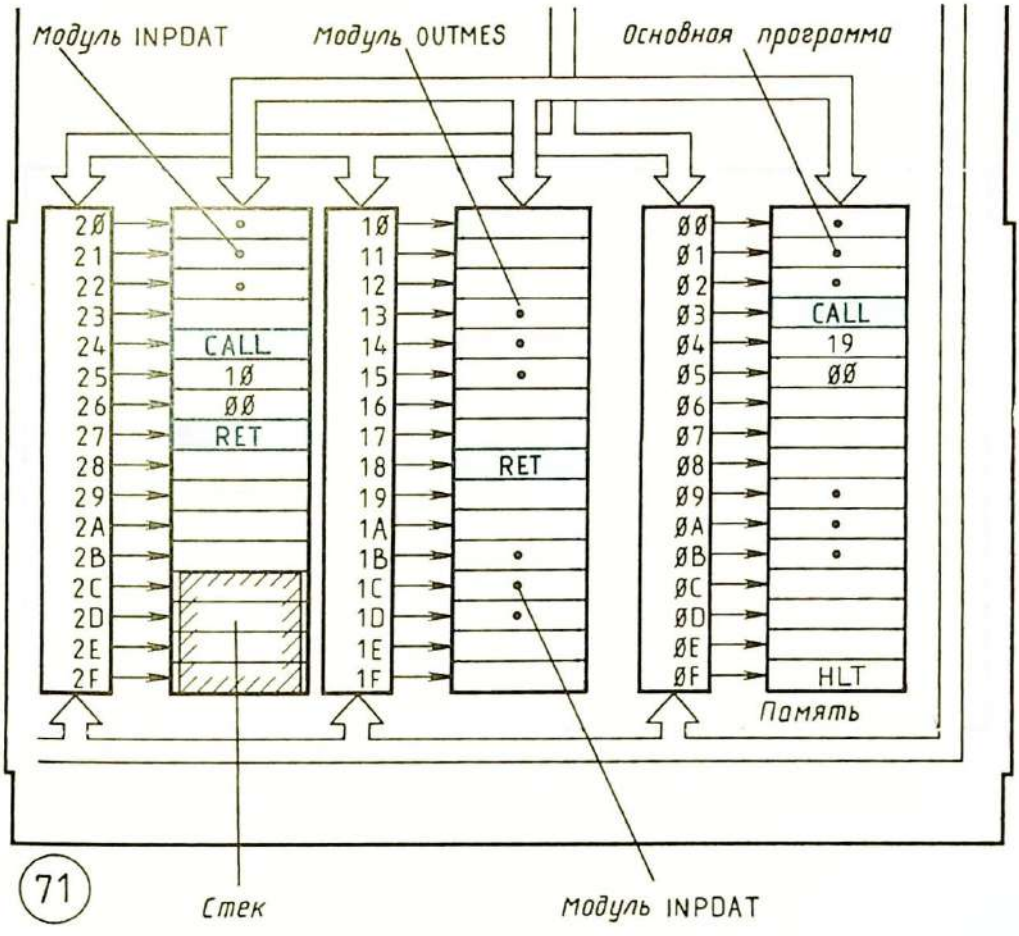
Программы после трансляции получили начальный адрес ØØØØ.Такой адрес в памяти единственный, поэтому получается, что модули претендуют на одно и то же место в ней. Разрешить этот конфликт можно, разместив модули в разных ее участках. Для этой цели служит еще одна специальная программа — *компоновщик*, называемая также *редактором связей*. Ее назначение в том, что она должна расположить отдельные модули последовательно в памяти и настроить их на конкретные адреса. Таким образом, кроме формирования двоичного кода, который получил название *объектный*, ассемблер строит таблицу неопределенных символов. Редактор связей использует эту таблицу для записи в загрузочный код действительных адресов.

После компоновки модулей *редактором связей* (основная программа, подпрограмма OUTMES, подпрограмма INPDAT) вся программа займет место в памяти так, как показано на рисунке 71. Редактор связей установил адреса глобальных символов — OUTMES, INPDAT и записал их в соответствующие байты программ.

Теперь программа готова к выполнению. Проследим по рисункам 72—75 последовательность вызова подпрограмм. Для упрощения условимся, что в регистре указателя стека уже находится адрес вершины стека — ØØ2F.

В ходе выполнения основной программы встречается команда CALL. В начале цикла ее выполнения содержимое регистра УС пересылается в регистр адреса и значение регистра СК, указывающего на следующую команду, записывается по этому адресу в память. Значение регистра УС увеличивается на 2 (рис. 72). На третьем и четвертом шагах цикла выполнения в регистр СК записывается содержимое второго и третьего байтов команды, указывая на следующую команду для выборки, а ею оказывается первая команда подпрограммы INPDAT (рис. 73).

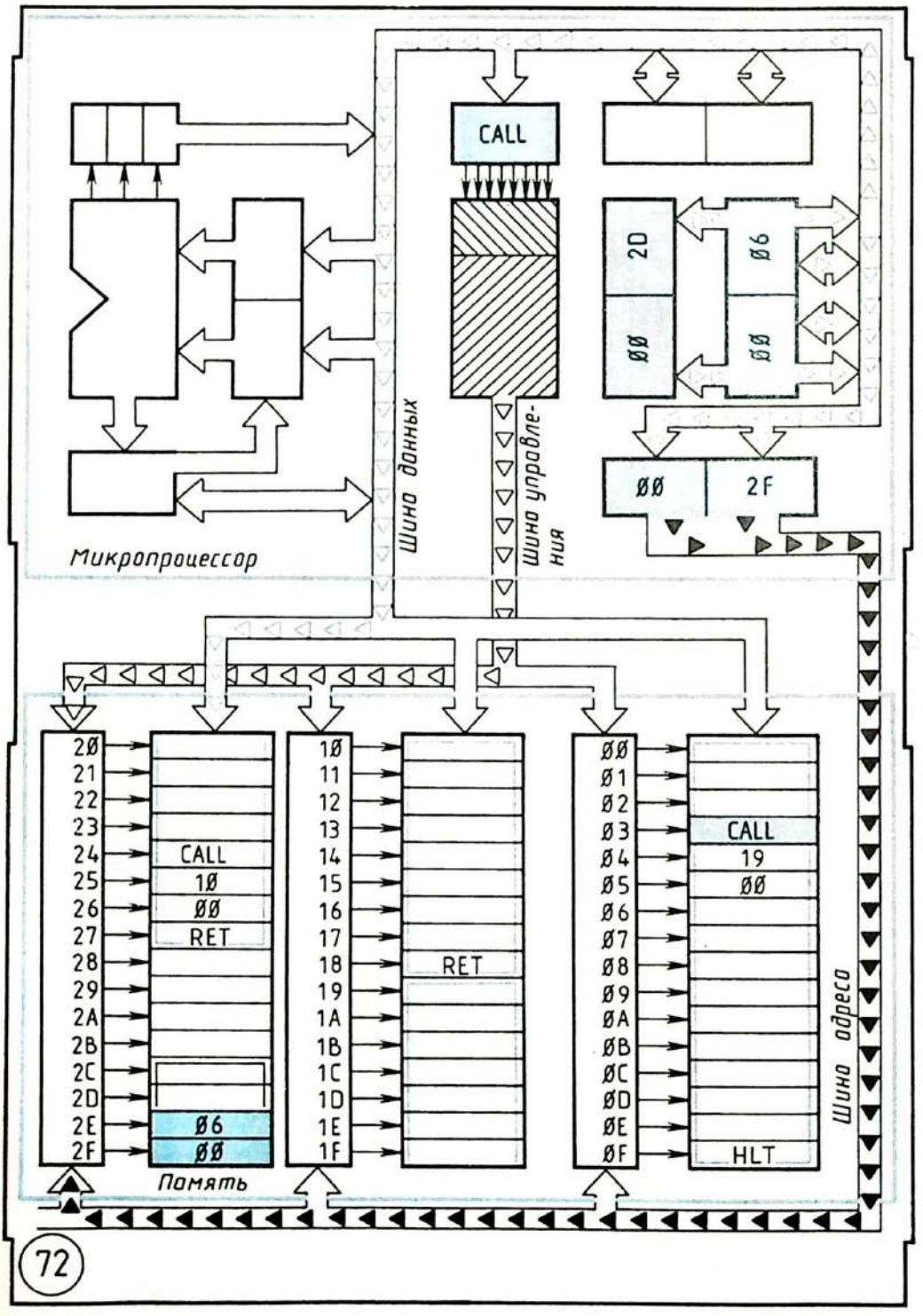
Через несколько команд и здесь встречается команда CALL. Цикл ее выполнения аналогичен предыдущей. В стек помещается адрес

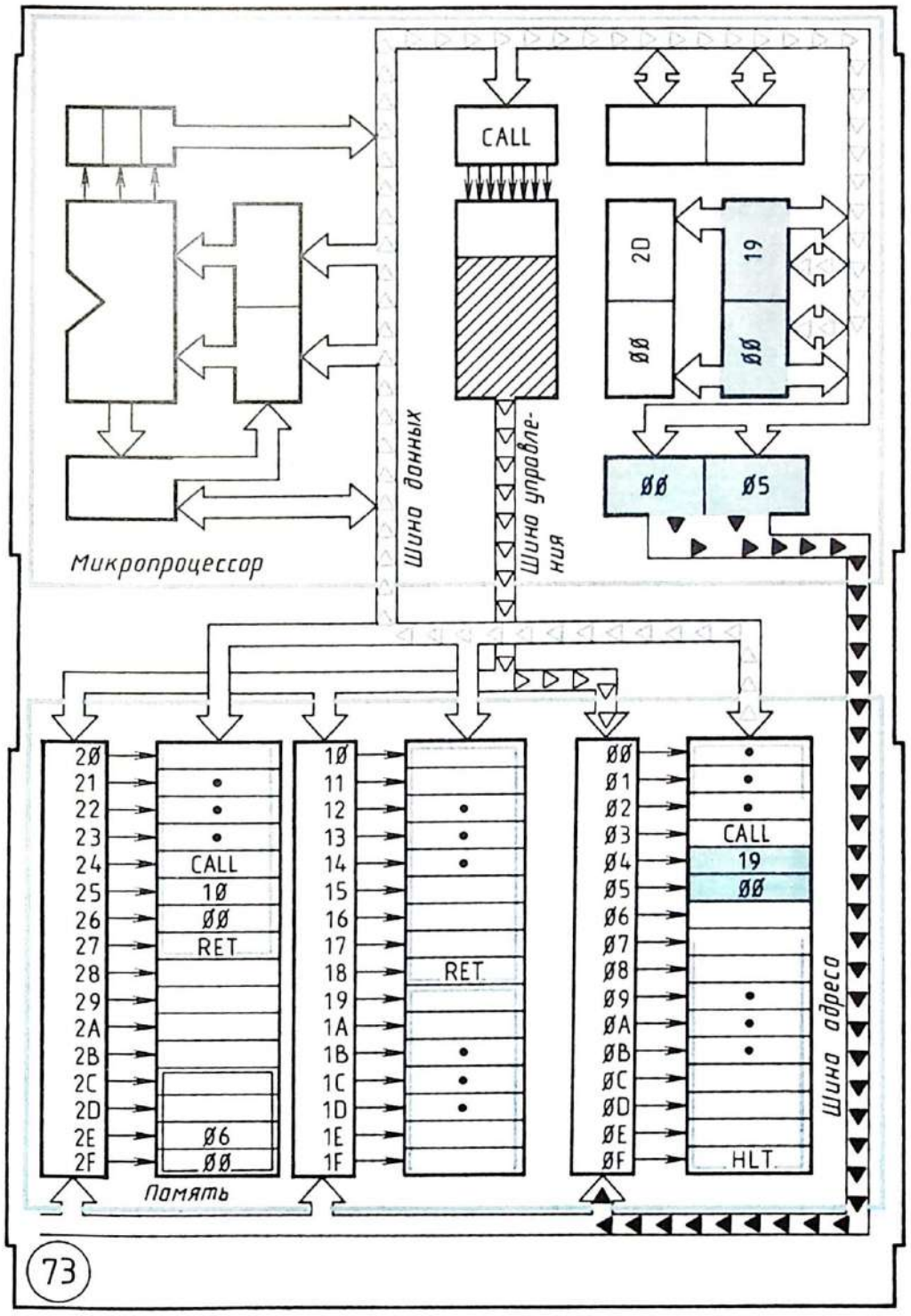


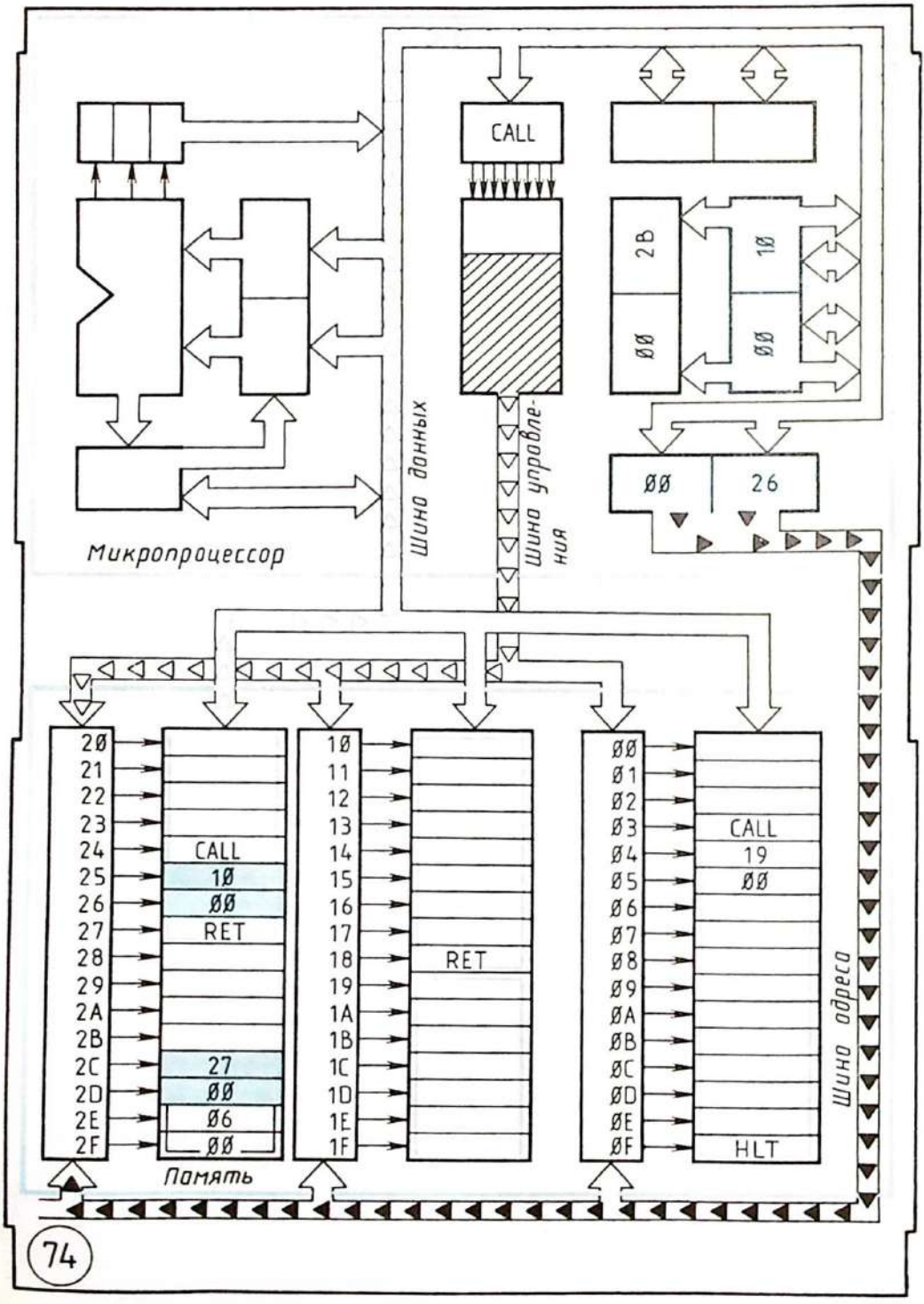
команды, записанной после CALL; модифицируются значения регистров УС и СК (рис. 74). Теперь следующей командой для выборки стала первая команда подпрограммы OUTMES.

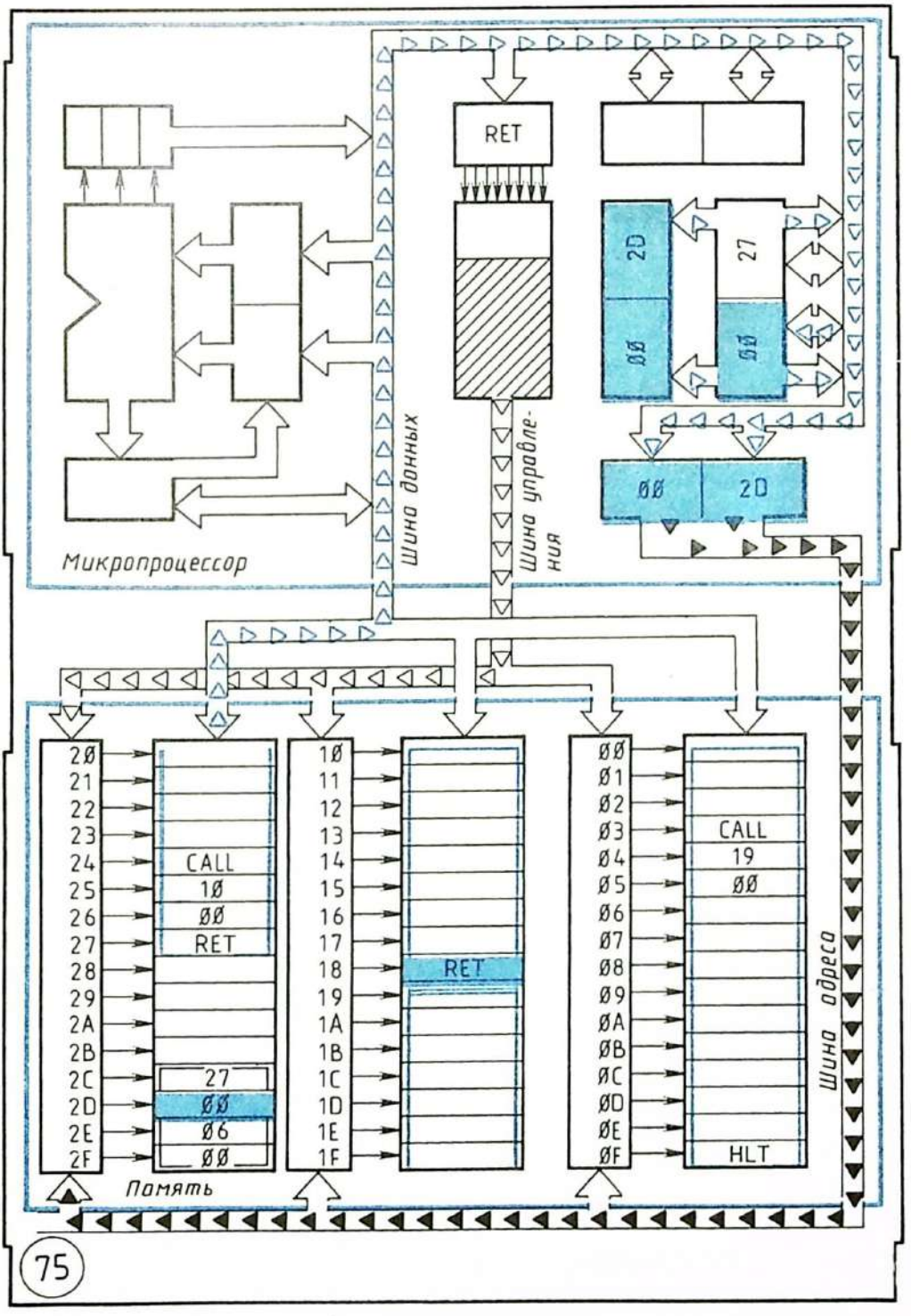
Далее посмотрим, как происходит возврат из подпрограмм. Последней командой подпрограммы OUTMES будет команда RET. В начале ее выполнения регистр УС получает приращение на 1. Полученное значение передается в регистр адреса, содержимое которого теперь указывает на ячейку стека, в которой находится младший байт адреса команды прерванной программы. Этот байт передается в регистр СК. Завершается выполнение команды RET передачей из стека старшего байта адреса команды прерванной программы; при этом значение регистра УС увеличивается еще на 1. Сформированный в регистре СК адрес указывает на команду, перед которой было прервано выполнение вызывавшей программы (рис. 75).

А по адресу ØØ27, имеющемуся в регистре СК, находится команда RET подпрограммы INPDAT. Ее выполнение аналогично предыдущей команде, и в результате в регистре СК будет адрес ØØØ6, выбранный из стека и указывающий на следующую команду основной программы.


CALL
2D
Ø6
ØØ
ØØ
ØØ
2F
Микропроцессор
Шина данных
Шина управления
20
21
22
23
24
25
26
27
28
29
2A
2B
2C
2D
2E
2F
CALL
10
ØØ
RET
Ø6
ØØ
Память
10
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F
RET
ØØ
Ø1
Ø2
Ø3
Ø4
Ø5
Ø6
Ø7
Ø8
Ø9
ØA
ØB
ØC
ØD
ØE
ØF
CALL
19
ØØ
HLT
Шина адреса


CALL
2D
19
ØØ
ØØ
ØØ
Ø5
Шина данных
Шина управления
Микропроцессор
2Ø
21
22
23
24
25
26
27
28
29
2A
2B
2C
2D
2E
2F
CALL
1Ø
ØØ
RET
Ø6
ØØ
Память
1Ø
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F
RET
ØØ
Ø1
Ø2
Ø3
Ø4
Ø5
Ø6
Ø7
Ø8
Ø9
ØA
ØB
ØC
ØD
ØE
ØF
CALL
19
ØØ
HLT
Шина адреса


CALL
2B
ØØ
1Ø
ØØ
ØØ
26
Шина данных
Шина управления
Микропроцессор
2Ø
21
22
23
24
25
26
27
28
29
2A
2B
2C
2D
2E
2F
CALL
1Ø
ØØ
RET
27
ØØ
Ø6
ØØ
Память
1Ø
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F
RET
ØØ
Ø1
Ø2
Ø3
Ø4
Ø5
Ø6
Ø7
Ø8
Ø9
ØA
ØB
ØC
ØD
ØE
ØF
CALL
19
ØØ
HLT
Шина адреса


RET
2D
27
ØØ
ØØ
ØØ
2D
Микропроцессор
Шина данных
Шина управления
2Ø
21
22
23
24
25
26
27
28
29
2A
2B
2C
2D
2E
2F
CALL
1Ø
ØØ
RET
27
ØØ
Ø6
ØØ
Память
1Ø
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F
RET
ØØ
Ø1
Ø2
Ø3
Ø4
Ø5
Ø6
Ø7
Ø8
Ø9
ØA
ØB
ØC
ØD
ØE
ØF
CALL
19
ØØ
HLT
Шина адреса
75

4.3. ОТ ФОРТРАНА ДО ФОКАЛА

Широким массам пользователей необходимо средство общения с машиной, которое дало бы возможность сокращать время программирования и отладки. Таким средством являются *языки высокого уровня*. При всем своем многообразии они основываются на сходстве с естественными языками, совместимы с математическими обозначениями и, как правило, не зависят от типа ЭВМ; пригодны для решения широкого круга задач. Использование таких языков стало первым шагом на пути автоматизации программирования и упрощения процесса общения с ЭВМ. Эффективные средства подготовки программ увлекли специалистов и математиков. В ноябре 1954 г. группа американских специалистов в области программирования, возглавляемая профессором Дж. В. Бэкусом, подготовила доклад, который считается сейчас первым сообщением, описывающим язык Фортран (*Fortran — Formula Translation* — преобразование формул). Язык был задуман для использования в сфере научных и инженерно-технических вычислений. Он быстро завоевал мир программирования благодаря своей близости, с одной стороны, к языку обычной алгебры, с другой — к естественному человеческому языку, а именно к английскому.

В 1958 г. была создана новая версия Фортрана, со значительным расширением языка, которая названа Фортран-II. В 1961 г. появился Фортран-III, включавший инструкции обработки сложных логических выражений и буквенно-символьной информации, а в 1962 г. появился Фортран-IV, имеющийся сегодня почти на всех ЭВМ. Длительный путь развития языка привел к появлению многочисленных диалектов. Для устранения этого разнообразия началась работа по его стандартизации, которая привела к опубликованию нового стандарта языка, названного Фортран-77.

После утверждения стандарта Фортрана-77 работа по развитию и совершенствованию языка не прекратилась. Было принято решение, что новый стандарт должен вводиться каждые пять лет. Однако огромный фронт задуманных работ по Фортрану-82 не позволил уложиться в ранее намеченные сроки. И только в 1985 г. была представлена на обсуждение новая версия, широкую эксплуатацию которой планируется начать к концу этого десятилетия.

Фортран оказал влияние на развитие многих других языков, в частности на язык Бейсик (*BASIC — Beginners Allpurpose Symbolic Instruction Code* — символический командный универсальный код для начинающего), который был разработан в 1963 г. небольшой группой студентов последнего курса Дортмутского колледжа под руководством профессора Дж. Кемени. Сейчас имеется множество различных вариантов языка Бейсик, сравнимых по количеству с потоком вариантов Фортрана 50-х гг. Сегодня этот язык стал более мощным, сохранив свою первоначальную простоту.

Первой серьезной попыткой в разработке языка для применения в экономических расчетах стал язык Кобол (*COBOL — Common Business*

Oriented Language — общий язык, ориентированный на экономические применения). Он был разработан специальной комиссией Конференции по языкам информационных систем, в состав которой входили изготовители ЭВМ, несколько крупных пользователей и другие заинтересованные организации. Отчет этой комиссии содержал первый вариант языка, названный Кобол-60, где число 60 обозначает год его создания. С того времени Комиссией по языкам программирования разработано несколько новых вариантов этого языка.

Значительным событием в мире программирования стало принятие в 1960 г. Международной конференцией в Париже языка Алгол-60 *(ALGOL — ALGOrithmic Language)*.

Алгол-60 привлек к себе всеобщее внимание благодаря некоторым новым общим идеям, и в 1968 г. коллектив ученых под руководством рабочей группы по Алголу разработал международный универсальный алгоритмический язык Алгол-68.

В 1964 г. объединенной комиссией фирмы IBM и разработчиками проекта SHARE FORTRAN впервые был предложен язык ПЛ/1 (PL/1 — *Programming Language One* — язык программирования 1). Сначала комиссия ставила своей целью расширение Фортрана путем добавления специальных средств обработки символов и операций с матрицами. Так как достижение этой цели оказалось нелегкой задачей, была начата разработка нового языка. ПЛ/1 является универсальным языком широкого применения, достоинства которого основаны на сочетании в нем многих свойств различных языков, таких, как Алгол, Фортран, Кобол и др.

На рубеже 60—70-х гг., в связи с расширением применения мини-, микро- и персональных ЭВМ, появилась потребность в новых языках, которые обеспечивали бы снижение стоимости программ и повышение их надежности. Одним из таких средств являются *диалоговые* языки, позволяющие пользователю эффективно взаимодействовать с ЭВМ через видеотерминал. Типичным представителем таких языков явился язык Фокал *(FOCAL — FOrmula CALculator)*.

Некоторые языки сильно упрощены и теряют свою ценность для опытных программистов, но для нас, начинающих пользователей, такие языки, несомненно, принесут удовлетворение от быстрых успехов в овладении программированием.

Какие же свойства языков высокого уровня дают возможность легко его усваивать и использовать?

4.4. ПРОСТОТА И ЕСТЕСТВЕННОСТЬ

Основной недостаток машинно-ориентированных языков программирования — их психологическая неестественность, а отсюда необходимость повышенного внимания и напряжения при работе с ними, что легко ведет к ошибкам и неточностям. Поэтому закономерным представляется стремление создателей языков высокого уровня к естественности в языке, т. е. введению некоторых аналогий традиционным, веками применявшимся методом математических описаний и расчетов.

Компонентами таких языков являются, прежде всего, наборы символов от A до Z, соответствующие алфавиту естественного языка, а также цифрам от 0 до 9 и знакам арифметических операций, знакомых, пожалуй, любому жителю Земли; плюс еще общепринятые синтаксические знаки (запятая, точка, точка с запятой). Слова в языках программирования взяты непосредственно из школьных учебников. Так, в Бейсике (наиболее доступном сегодня языке) применяются слова LET — пусть, IF — если, THEN — то, FOR — для и т. д. Большинство из них можно встретить и в других языках. Слова, связанные со спецификой вычислительной техники, также широко применяются в обиходной речи: PRINT — печать, INPUT — ввод, RETURN — возврат.

Приближает к естественности и возможность составлять из допустимых в каждом языке слов и символов целые предложения, которые в совокупности дают хоть и оригинальный, но вполне понятный человеку текст.

Полюбившийся нам пример с суммированием пяти чисел на Бейсике запишется так:

```
1Ø LET   S=Ø
2Ø LET   I=5
3Ø INPUT D
4Ø LET   S=S+D
5Ø LET   I=I-1
6Ø IF    I>Ø THEN 3Ø
7Ø PRINT S
8Ø END
```

Бесспорно, этот текст в наглядности выигрывает по сравнению с ассемблерным. Да и короче стала программа. Но ее размер можно еще уменьшить:

```
1Ø LET   S=0
2Ø FOR   I=1 TO 5
3Ø INPUT D
4Ø LET   S=S+D
5Ø NEXT  I
6Ø PRINT «СУММА=»; S
7Ø END
```

Конечно, в машине эта программа займет не меньше места, чем упоминаемая ассемблерная, так как вместо каждого оператора Бейсика транслятор подставит необходимое количество машинных команд, ставших затем все той же последовательностью нулей и единиц. Таким образом, чем больше программа, тем заметнее выигрыш от применения языка высокого уровня во времени написания программы, так как она короче, и по этой же причине выигрыш во времени ее отладки. Но объем памяти, занимаемой двоичным кодом программы, переведенной с Бейсика, и скорость ее выполнения будут, пожалуй, раза в два выше, чем для ассемблерной программы. Обусловлено это той избыточ-

ностью, которая неизбежно попадает в двоичный код при трансляции из-за универсальности операторов. Например, оператор PRINT имеет десяток режимов выполнения, и те машинные команды, которые его заменяют, учитывают это многообразие, даже если используется в данном месте один режим.

Кроме возможности составлять предложения-операторы, сокращающие текст программы, в языках высокого уровня применяются библиотечные подпрограммы, написанные разработчиками этих языков. Например, при введении различных вычислений часто бывает необходимо определить значение стандартных математических функций, таких, как синус угла, корень квадратный числа и т. д. В Бейсике такие вычисления не представляют труда благодаря встроенным функциям — подпрограммам, которые были заранее запрограммированы и включены в язык. К таким функциям, в частности, относятся:

SIN (X) — вычисляет синус угла, представленного в радианах;

COS (X) — вычисляет косинус угла, представленного в радианах;

LOG (X) — вычисляет натуральный логарифм числа;

EXP (X) — вычисляет результат возведения числа *e* в требуемую степень;

SQR (X) — вычисляет корень квадратный от положительного числа.

Еще одной отличительной особенностью языков высокого уровня является диалоговая работа с ЭВМ, которая уменьшает время ответа машины на запросы пользователя и увеличивает скорость решения задач. Благодаря тому что языки просты и доступны, можно проводить различные расчеты на персональной ЭВМ быстро и эффективно, не требуя помощи профессиональных программистов и специального оборудования вычислительного центра. Сам процесс написания программы допускает *прямолинейную* запись, избавляя человека от сложных логических построений и изучения каких-то дополнительных соглашений. Пользоваться таким языком, как Бейсик, можно от случая к случаю, легко восстанавливая в памяти основные правила языка.

4.5 ПРОГРАММИРОВАНИЕ. НЕ ВЕДАЕМ, ЧТО ПИШЕМ

Итак, мы написали программу на каком-либо языке программирования. Является ли программа правильной? Практический опыт человечества позволяет утверждать, что ошибки при программировании неизбежны. На заре вычислительной техники ошибки программирования объяснялись отсутствием опыта в освоении новой техники и казались временным явлением. Но, однако, положение к лучшему долго не менялось.

Главным направлением научных поисков в программировании стал основной и неизбежный вопрос — создание правильных программ. Определенный круг специалистов пытается решить его на этапе их составления, в противовес старым и примитивным методам *отладки — дампу и трассировке.*

Отладка — это процесс поиска ошибок в программе путем прогона ее на ЭВМ.

Дамп (dump — разгрузка) — это вывод содержимого памяти на печать или экран терминала в шестнадцатеричной (или восьмеричной) системе. Дамп памяти дает статическую картину состояния ее ячеек программы в определенный момент времени, например в момент останова программы из-за ошибки. Анализ дампа, особенно больших программ, требует кропотливого, длительного напряженного труда.

Трассировка (trace — след) прослеживает в динамике ход выполнения программы, выводя на печать содержимое регистров и некоторых ячеек памяти после каждой команды. При этом расходуется много бумаги при минимальной полезной информации.

Объединить полезные свойства дампа и трассировки удалось в специальной программе — *диалоговом отладчике,* с помощью которой можно организовать пошаговое выполнение отлаживаемой программы с возможностью просмотра на экране терминала после каждой команды содержимого регистров и любой ячейки памяти. Кроме того, можно выполнять программу по участкам, задавая точки останова программы.

С помощью этих методов ошибки находятся те, которые ищутся. Как же быть с теми ошибками, о которых программист не подозревает? Увы, они могут проявиться и после длительного времени эксплуатации программы. Такая неприятность может возникнуть и благодаря случайному набору входных или выходных данных, ранее не встречавшихся программе. Попыткой предотвратить эту ситуацию является *тестирование (test* — испытание) программы на различных наборах данных, называемых *контрольными примерами.* Этот процесс проводится после этапа отладки программы, когда программист убеждается в отсутствии явных ошибок.

При составлении контрольных примеров необходимо начинать с простых данных, постепенно переходя к более сложным наборам. При этом обязательно надо включать крайние значения данных. Если величина А может принимать значения от 10 до 100, то следует проверить такие значения, как 9, 10, 100, 101. При определении данных контрольного примера надо учитывать логику программы для того, чтобы каждый оператор был выполнен. Результаты выполнения должны быть известны до начала прогона контрольного примера. Затруднительно судить о работоспособности программы, если любой результат будет неожиданным. К сожалению, составить и подготовить контрольные примеры на все случаи жизни практически невозможно. Значит, опасность встречи с ошибкой остается.

Ученые многих стран стали задумываться над вопросом: не является ли сам процесс написания программ неправильным? Ведь математическая основа программирования очень проста. Конечная последовательность нулей и единиц подвергается конечному числу операций. Почему же человек неизбежно допускает ошибки в столь «простом» деле? Ответ нашли в размерности проблемы программиро-

вания. Память машины настолько велика, а скорость выполнения операций еще больше, что человек теряется в представлении этих величин. Его голова несоизмеримо мала по сравнению с тем объектом информации, который обрабатывает машина даже за доли секунды. Необозримость этих скоростей и объемов наложила некоторый магический отпечаток на деятельность людей, отважившихся связать свою судьбу с программированием.

Программирование начиналось как искусство одаренных. Новые программисты учились ему, наблюдая, как работают другие, овладевая приемами и навыками, но мало задумываясь над теоретическими основами и фундаментальными принципами программирования. Энтузиазм пионеров помогал преодолевать титанические трудности при выявлении ошибок в многобайтных двоичных кодах (дампах программ), что похоже на поиск иголки в стоге сена; он же подкреплялся мимолетными радостями успешных прогонов программ, сменяемых очередными изменениями и доработками их (очередными ошибками тоже) по воле неумолимого заказчика. «Жизнь трудна, потому что конкретна» — гласит народная пословица. Какие же трудности приходится преодолевать программисту, который конкретными методами решает задачи, выдвигаемые жизнью!

Парадоксально, но эта уникальная технология, пожалуй, не имеющая себе равных среди человеческих занятий, в течение десятилетий не имела научной основы. Так долго продолжаться, конечно, не могло. Переломными стали 60-е гг. этого столетия, особенно их начало, которое было связано с проектом языка Алгол-60. Этот язык уже перестал быть полезен, но математическое единство как самого языка, так и его описания до сих пор оказывает глубокое влияние на развитие программирования. И к концу тех же годов это единство стало очень важной областью научных исследований, основанных на теории автоматов и формальных языков. Сегодня еще рано говорить о больших успехах в этой области, но понятия, принципы и средства, на которых может базироваться программирование, уже начали проясняться.

Пока такие средства разрабатываются, теоретики программирования не сидят без дела; вопрос качества программ не исчерпывается лишь доказательством их правильности. Он включает в себя и такие понятия, как эффективность вычислений, удобство в эксплуатации и сопровождении и, наконец, экономическая оценка программ, обусловленная длительностью и трудоемкостью их разработки. Попытки решения этих проблем привели к переводу процесса разработки программ в статус *технологии программирования,* которая представляет собой совокупность правил и приемов, позволяющих составлять программы как изделия высокого качества при минимальных затратах.

Перевод программирования на «технологические рельсы» только начинается, и у тебя есть счастливая возможность принять участие в этом процессе.

4.6. НАШИ ПРОГРАММЫ И ПРОГРАММЫ МАШИН

«Раньше назначение программ заключалось в управлении нашими вычислительными машинами, теперь назначение вычислительных машин состоит в исполнении наших программ». В этом выражении Дейкстры отражены те изменения, которые произошли в программных средствах ЭВМ за время их развития.

Первые программисты ЭВМ вынуждены были преодолевать двойные трудности. Кроме кодирования своих расчетов на машинном языке, им необходимо было включать в свои программы средства управления самой машиной — загрузку программы в память, доступ к оборудованию. Эту малоприятную работу приходилось выполнять каждому пользователю, да и работать на машине они могли только по очереди. При этом наблюдались большие потери времени на смену программ в ЭВМ и эффективность ее использования была очень низка.

Все возрастающее число пользователей хотело быстро и удобно писать свои программы и их выполнять. Идя навстречу естественным требованиям потребителей, разработчики вычислительной техники не нашли ничего лучшего, как возложить решение этих проблем на плечи самих же программистов. В виде компенсации группе программистов, решивших посвятить себя столь нелегкому делу, присвоили квалификацию *системных программистов;* упоминание этих слов до сих пор приводит в трепет многотысячную армию простых пользователей ЭВМ, не имеющих ни малейшего желания вникать во все тонкости вычислительной техники.

Легендарные *системные программисты,* преодолев многие трудности, создали программы, получившие название *операционные системы.* Дать точное и правильное определение операционной системы затруднительно. Наиболее близким, пожалуй, является восприятие операционной системы как *набора специальных программ,* являющихся неотъемлемой частью машины и служащих для управления как ее ресурсами, так и программами пользователя. К главным ресурсам машины, как мы уже знаем, относят процессор, оперативную память и внешние устройства. Тогда основные функции операционной системы заключаются в следующем: планирование работы процессора, загрузка и выполнение программ, распределение памяти между программами, управление внешними устройствами.

На рисунке цветной вклейки VI,*б* операционная система представлена как основной элемент программного обеспечения ЭВМ. Операционная система является тем двигателем, который позволяет вращаться остальным программам вокруг аппаратного стержня.

Программы, названные нами как *остальные,* представляют две группы. Первая — это непосредственно наши пользовательские программы, которые получили название *прикладных.* Собственно, ради их обеспечения существуют и операционная система, и большая вторая группа специальных программ, с некоторыми из них мы уже знакомы. Они предназначены главным образом для автоматизации подготовки и

отладки программ; каждая из них соответствует определенному этапу этого процесса. Таких этапов в общем случае четыре: подготовка текста, трансляция, сборка программы и ее отладка (см. рисунок цветной вклейки VII,*а*). Последовательность действий программиста (шагов) при разработке программы показана на рисунке цветной вклейки VII,*б*.

Вначале с клавиатуры терминала запускается программа *текстовой редактор*. С помощью вводимых команд редактор позволяет формировать строки программы, изменять их или удалять (шаг 1). Составленный таким образом текст просматривается на экране терминала (шаг 2), а затем с помощью того же текстового редактора помещается на магнитный диск (шаг 3) в виде символьного файла. Если программист удовлетворен проделанной работой, он запускает программу — транслятор (шаг 4), которая считывает указанный символьный файл с диска (шаг 5), обрабатывает его и результат (листинг) выводит на устройство печати (шаг 6). Одновременно им формируется на магнитном диске *модуль* машинных команд, соответствующий исходному тексту разрабатываемой программы,— файл с объектным модулем (шаг 7).

В процессе обработки транслятор может обнаружить ошибки синтаксиса программы, о чем он тут же сообщает на экране терминала (шаг 8). Можно, оторвав лист с текстом программы, отойти от машины и проанализировать допущенные ошибки, после чего программист вновь инициирует редактор текста для их исправления (шаги 9, 10). Далее процесс повторяется (шаги 11, 12, 13, 14). Последовательность этих шагов применяется до тех пор, пока транслятор не засвидетельствует отсутствие ошибок синтаксиса. В любом случае листинг программы потребуется в дальнейшем при модификациях программы.

Следующий этап начинается с запуска программы *редактор связей* (шаг 15), который объединяет несколько объектных модулей в единый загрузочный модуль, связывая между собой их адреса. Сообщим ему имя нашего файла с объектным модулем. Этот файл будет считан с диска (шаг 16) и обработан. При этом на экране формируется карта загрузки — текст, содержащий перечень модулей, глобальных символов и соответствующих им адресов (шаг 17). Карта загрузки может при желании появиться и на устройстве печати (шаг 18). На диске будет создан загрузочный модуль (шаг 19).

Просматривая карту загрузки, программист определяет *ошибки связывания,* представляющие собой неразрешенные ссылки. Если они имеются, необходимо снова вернуться к этапу изменения текста и трансляции.

Ошибки, определяемые редактором связей, встречаются редко, поэтому мы предполагаем, что наша программа составлена с первого раза. Можно переходить к следующему этапу — *отладке*. Здесь существенную помощь оказывает программа-*отладчик*. Инициируя отладчик (шаг 20), ему передают имя отлаживаемой программы, что приведет к считыванию с диска загрузочного файла (шаг 21), расположению его в памяти и созданию условий для отладки. Далее можно просматривать

на экране (шаг 22) ход выполнения программы, обнаруживая ошибки. Для их исправления вновь вызываем редактор текста, изменяем текст, транслируем программу и связываем ее. Дальше ничего другого не остается, как снова запустить отладчик и продолжить отладку программы.

Такова схема подготовки программы на ЭВМ. Но это лишь схема. Реально последовательность этапов выглядит несколько по-другому при программировании на разных языках. Так, например, транслятор с Бейсика содержит все четыре этапа вместе. Объясняется это тем, что он представляет специальный вид трансляторов, называемый *интерпретатором*. Замечательной особенностью их является то, что создаваемая программа выполняется сразу, т. е. как только написан оператор. При этом Бейсик незамедлительно сигнализирует об ошибках и дает возможность исправить оператор. Кроме того, программа сразу формируется в оперативной памяти. Однако такой подход, обладая достаточной простотой, имеет ряд ограничений. Поэтому наибольшее распространение получили трансляторы второго типа — *компиляторы;* создание программ на их языках полностью соответствует схеме из четырех этапов. К таким языкам относится, например, Фортран.

Создание программ с использованием компиляторов требует наличия внешней памяти на магнитных дисках или лентах.

Применение внешних запоминающих устройств требует для управления операциями ввода-вывода преобразования форматов данных и упорядоченного хранения специальных программ, совокупность которых составляет *файловую систему.* Для манипулирования файлами на дисках или лентах, т. е. для копирования, удаления, просмотра их содержимого, применяются специальные программы — *утилиты (utility* — обслуживающая программа).

Сейчас мы знаем, что назначение вычислительных машин состоит в исполнении наших программ, и это стало возможным благодаря только тому, что написаны программы, которые управляют нашими вычислительными машинами.

4.7. ПАСКАЛЬ, АДА И ДРУГИЕ

Уже с момента начала работ над первыми языками программирования, особенно во время создания Алгола-60, начала зреть уверенность в том, что язык можно разработать на чисто научной основе. Появились попытки построения формальных определений языков программирования. В этих построениях играли большую роль понятия теории автоматов и формальных языков.

К концу 60-х гг. стало ясно, что в области математического обеспечения программирования имеются огромные проблемы. Размеры и сложность проектов непомерно разрастались, что не соответствовало возможностям программирования. Это приводило к ненадежному программному обеспечению, ведущему к нарушению плановых сроков и стоимости разработок. И таким образом стало ясно, что Алгол-60 был

шагом в неверном направлении и что необходимо создать более простой язык. Первым языком нового направления стал язык программирования Паскаль, в котором нашли отражение концепции программирования, определенные Э. Дейкстрой и Т. Хоором из Белфастского университета. Язык был разработан Никлаусом Виртом в Швейцарском техническом институте в Цюрихе. Создатель языка Паскаль назвал его так в честь Блеза Паскаля, первого конструктора устройства, которое теперь относится к классу цифровых вычислительных машин. В настоящее время этот язык широко применяется как язык эффективного программирования, а также как хорошее средство обучения программированию.

Язык Паскаль оказался настолько изящным и почти универсальным, что у теоретиков программирования возникла идея создать универсальный язык. И действительно, в июле 1980 г. появился проект языка Ада. По мнению разработчиков, язык рассчитан на применение в самых разных приложениях: от инженерных расчетов до системного программирования. Однако перед разработчиками стоит целый ряд проблем по его упрощению, что может привести к длительному его становлению.

Более удобным, выразительным и гибким языком для программирования широкого класса задач стал язык Си. Характерной особенностью его является отсутствие ограничений на написание системных программ, а так как он соответствует возможностям современных машин, то программы на нем достаточно эффективны и нет необходимости прибегать к языку Ассемблера.

Хотя язык Си основан на особенностях современных машин, однако он не связан с какой-либо конкретной архитектурой, поэтому на нем достаточно просто писать программы, которые будут легко переносимы между различными машинами.

Разумеется, что эти языки не решают всех проблем, но они продолжают развиваться, преодолевая свои отрицательные стороны, которые имеются в области надежности, работы в режиме реального времени и др.

Итак, дорогой читатель, ты узнал об истории развития нескольких языков программирования, ставших сегодня основным инструментом общения с современными ЭВМ. В настоящее время языков разработано множество, и этот процесс продолжается. Но постоянное стремление к упрощению средств взаимодействия с вычислительной машиной не оставляет теоретиков и специалистов в покое. Заветная цель — программирование без программистов — ощутимо близка. Одним из методов достижения ее является *аппаратная реализация* традиционных функций программных средств, предполагаемая в машинах пятого поколения.

Проверь познания и чувства,
Отъедини от бреда явь.
И труд на уровень искусства
Рукою мастера поставь.
Гляди вперед добрей и шире,
Переступи тоску и страх.
Все совершенство — в этом мире,
И мир стоит на мастерах.

М. Дудин

ГЛАВА 5. ЭВМ: ПРИМЕНЕНИЕ

5.1. И ВСЕ-ТАКИ МИКРОПРОЦЕССОРЫ

Благодарное человечество будет вспоминать 70-е гг. этого столетия как время нового направления в развитии ЭВМ, а именно микропроцессорной техники. Прогресс в области технологии интегральных схем (см. рисунок цветной вклейки VIII), приведший к появлению схем средней интеграции (СИС), а затем большой (БИС) и сверхбольшой (СБИС) позволил реализовать много разных функциональных возможностей машин. Эти существенные достижения прежде всего привели к появлению *микрокалькуляторов*.

Микрокалькулятор — это микроЭВМ, работающая в «однотактном» режиме. Вычислительный процесс в ней управляется человеком. Первый советский микрокалькулятор появился в конце 1971 г. В 1973 г. был выпущен микрокалькулятор «Электроника БЗ-04» для выполнения арифметических операций, а затем появился инженерный «Электроника БЗ-18» (1976 г.), способный производить вычисление функций, и в 1978 г. был создан микрокалькулятор «Электроника БЗ-21», положивший начало семейству программируемых микрокалькуляторов. Прогресс в производстве этих устройств был таков, что если в 1972 г. микрокалькулятор стоил в 4 раза дороже механического арифмометра, то уже в 1974 г. он стал стоить в 10 раз дешевле того же арифмометра. С 1979 г. выпускается школьный микрокалькулятор МКШ-2. Кроме него, в школьной практике находят применение программируемые микрокалькуляторы «Электроника БЗ-34», МК-54 и МК-56.

Основным аппаратным узлом любой микроЭВМ является микропроцессор, развитие которого направлено в сторону увеличения интеграции элементов в одном кристалле и увеличения разрядности слова при относительном снижении его стоимости. Так, если первый микропроцессор *Intel*-4004 имел разрядность 4 бита, то уже в середине 70-х гг. строились ЭВМ на 8-разрядных микропроцессорах, а в 1981 г. были выпущены персональные ЭВМ на основе 16-разрядных микропроцессоров. В конце 80-х гг. предполагается появление 64-разрядных микропроцессоров. Число элементов в кристалле микропроцессора при этом возросло с 2300 у 4-разрядного до 450 000 у 32-разрядного и достигнет одного миллиона у 64-разрядного.

Кроме микропроцессора микроЭВМ содержит запоминающее устройство и систему ввода-вывода данных. Однокристальные ЭВМ содержат эти три компонента в одной БИС. Первая модель семейства однокристальных микроЭВМ MCS-48 была создана фирмой *Intel* (США) в 1976 г. Дальнейшим развитием этих ЭВМ стала разработка в 1980 г. микроЭВМ MCS-51. На практике отсутствуют вычислительные системы общего назначения, базирующиеся на однокристальных микроЭВМ, так как по своей природе они не могут обеспечить разработку собственных программ. Программы для таких машин разрабатывают на более мощных вычислительных системах, а сами однокристальные микроЭВМ применяются как *микроконтроллеры,* размещаемые внутри изделий для управления ими. Диапазон примеров таких изделий весьма широк — от электронных игрушек до автомобилей и промышленных роботов.

Микроконтроллеры, содержащие однокристальную микроЭВМ, имеют еще дополнительное электронное оборудование и в целом размещаются на одной плате печатного монтажа. Поэтому они называются также одноплатными контроллерами.

МикроЭВМ, предназначенные для вычислительных задач и обработки данных, имеют более сложную конструкцию и размещаются на нескольких платах. Отсюда и название их — многоплатные микроЭВМ. Многоплатными являются первые отечественные микроЭВМ семейства «Электроника», из которых широкое применение получила машина «Электроника-60М». Среди одноплатных ЭВМ известна «Электроника-41».

Наиболее развитыми моделями являются машины типа «Электроника НЦ-80». Одна из них «Электроника НЦ-80-20/2» (другое название ДВК-2М) является диалоговым вычислительным комплексом, у которого быстродействие — 500 тыс. операций, память — 56 килобайт; имеется возможность программировать на языках Фортран, Бейсик, Фокал.

Первой отечественной ЭВМ, предназначенной для применения в быту, стала микроЭВМ «Электроника БК-0010». Быстродействие — 300 тысяч операций, память — 32 килобайта. Программирование ведется на языках Бейсик, Фокал. В качестве дисплея используется обычный телевизор, а накопителем информации является кассетный магнитофон.

5.2. И ПЕРСОНАЛЬНЫЕ ЭВМ

Машины типа ДВК и БК-0010 рассчитаны на одного пользователя и в этом смысле относятся к *персональным ЭВМ.*

Сегодня персональные ЭВМ не имеют четкого определения. Обычно термин «персональная ЭВМ» относят к машине, которая в своей основе имеет электронные схемы, построенные на базе современной микропроцессорной технологии. Ее программное обеспечение обладает возможностью поддержки диалогового режима работы с пользователем. При этом пользователь имеет доступные средства раз-

вития программного обеспечения, используя языки высокого уровня типа Бейсик, Фокал и др. Стоимость такой машины доступна для индивидуального покупателя, а работа с ней легко осваивается даже тем пользователем, который вообще не имеет предшествующего опыта общения с вычислительной техникой.

Еще в начале эры ЭВМ разработчики стремились к упрощению методов общения человека с машиной. Разработка удобных средств такого общения стала возможной лишь с развитием микроэлектроники, а также с созданием разнообразного программного обеспечения.

Сегодня в развитии персональных ЭВМ выделяют два поколения.

ПЭВМ *первого поколения* основывались на 8-разрядном микропроцессоре, имели оперативную память до 64 кбайт. Ярким представителем этого поколения служат машины *Appl,* созданные в 1976 г. молодыми американцами Стивеном Джобсом и Стивом Возняком в домашнем гараже под яблоней. Удачные технические решения, заложенные авторами, позволили в короткий срок «раскатиться» этим «яблокам» по всему миру.

ПЭВМ *второго поколения* базируются на 16-разрядных микропроцессорах. Их производительность 1 млн. операций в 1 с, оперативная память 1 Мбайт. В настоящее время емкость оперативной памяти у некоторых моделей достигает 16 Мбайт, а внешней измеряется десятками Мбайт. Основоположниками машин *второго поколения* явились фирмы IBM и *Intel,* которые разработали в конце 1981 г. в США модель IBM PC *(Personal Computer).* Эта ПЭВМ, построенная на базе микропроцессора *Intel*-8086, производит сложение за 1,1 мкс при длине слова в 16 разрядов; имеет до 1 Мбайт оперативной памяти, распознает 133 команды. Кроме монохроматического дисплея и устройства печати к машине подключаются гибкие диски, цветной графический монитор или бытовой телевизор и даже громкоговоритель, через который можно прослушивать музыку, генерируемую программами. ЭВМ имеет встроенный интерпретатор языка Бейсик, но может работать и с другими языками под управлением операционной системы.

Современные персональные ЭВМ равны по своим вычислительным возможностям большим машинам 60-х гг. и мини-ЭВМ 70-х гг.

В нашей стране развернута широкая программа разработки и выпуска персональных ЭВМ. Только в XII пятилетке намечено выпустить 1 млн. 100 тыс. машин. С учетом требований применения ПЭВМ в различных сферах нашей жизни намечен выпуск нескольких моделей. Но все они подчинены единому требованию персонального компьютера, где основными компонентами являются:

— портативная одноблочная конструкция, габариты которой определяются требованием настольного расположения и удобства пользования человеком;

— клавиатура как манипулятор для управления устройством;

— экран как средство динамического отображения алфавитно-цифровой и графической информации;

— устройство внешней памяти со сменным носителем, допускающим многократное чтение и запись данных;

— устройство печати для получения твердых копий содержимого экрана.

Перечисленные компоненты нам знакомы. Это и дисплей на основе электронно-лучевой трубки с клавиатурой, устроенной по принципу пишущей машинки, и гибкие магнитные диски, и мозаичные или термографические безударные принтеры.

Программное обеспечение ПЭВМ основано на простых и надежных операционных системах, обеспечивающих диалоговый интерфейс с пользователем, и на применении нескольких языков высокого уровня, понятных и легко изучаемых. В таблице 31 показаны основные характеристики отечественных ПЭВМ, с которыми ты встретишься в своей учебной или производственной деятельности.

5.3. РАБОТАЕМ С ЭВМ

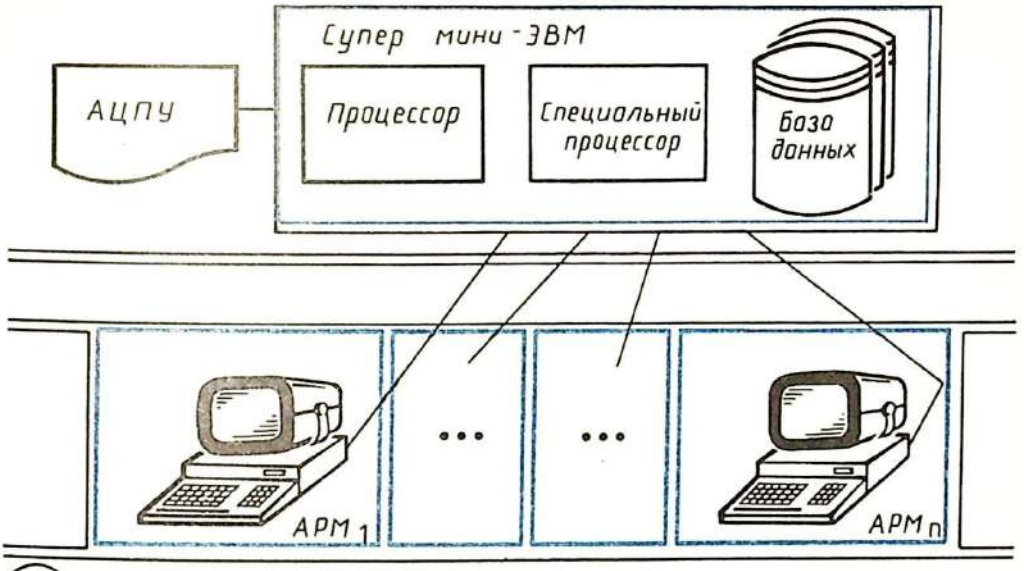
Сейчас ты, читатель, владеешь многими понятиями и терминологией вычислительной техники. Теперь мы смело можем начать экскурсию по *автоматизированным рабочим местам* (АРМ), подойти поближе к станкам с программным управлением и роботам. Многое тебе уже будет понятно, но кое-что ты узнаешь и новое. Перевернув последнюю страницу, ты все еще останешься учеником. Впереди еще много книг, много часов работы за терминалом. А пока сделаем первые шаги за порог школы, первые шаги к мастерству. Итак, каковы же функции АРМов в различных системах? Где находят себе применение микропроцессоры?

5.3.1. Считаем

Научно-технические расчеты (НТР), ведущиеся во множестве организаций, имеют широкий диапазон — от простых вычислений, доступных калькулятору, до крупных длительных расчетов. Учитывая потребности вычислительных приложений, система НТР может быть построена так, как показано на рисунке 76. Множество АРМов, расположенных в различных помещениях, играют роль мощных калькуляторов для простых и средних вычислений. При этом исходные данные могут вводиться как с клавиатуры терминала, так и из главной базы данных (БД), находящейся на большой машине, роль которой с успехом выполняет супермини-ЭВМ, имеющая большую арифметическую мощность за счет включения в состав машины дополнительного специального процессора, выполняющего при повышенной точности операции с высокой скоростью над числами с плавающей запятой. Для выполнения больших расчетов задание вводится с терминала АРМа, а сам расчет производится на главной машине. Результаты расчетов выводятся как на экран терминала, так и на алфавитно-цифровое печатающее устройство (АЦПУ) главной машины и помещаются в общую БД.

Таблица 31

Характеристики ПЭВМ	Модели ПЭВМ					
	ДВК-3М2	ДВК-4	Электроника-85	ЕС-1840	Искра-1030	Нейрон-И9.66
Быстродействие, тыс. операций / с	800–1200	800–1200	600	1000	1000	1000
Число команд	64–72	64–72	138	133	133	133
Оперативная память, кбайт	64–4096	4096	512–4096	256–640	512–1000	256–1000
Емкость дисковой памяти, кбайт	440–800	800–1600	5000	320	320	320
Тип дисплея	Монохромный графический	Цветной графический	Цветной графический	Монохромный алфавитно-цифровой, графический	Монохромный алфавитно-цифровой, графический	Монохромный алфавитно-цифровой, графический
Операционная система	ОС ДВК	ОС ДВК	РАФОС	М86	АДОС	Нейрон-ДОС
Языки программирования	Ассемблер Бейсик Паскаль	Ассемблер Фортран Бейсик Паскаль Модула-2	Ассемблер Фортран Бейсик Паскаль Модула-2	Ассемблер Фортран Бейсик Паскаль Си	Ассемблер Бейсик Паскаль Си	Ассемблер Бейсик Паскаль



76

Для программирования научно-технических расчетов с наилучшей стороны зарекомендовал себя язык Фортран, который, собственно, и создавался для таких целей. Эти расчеты имеют две особенности: высокая точность обрабатываемых данных и значительное количество циклов (итераций), широко используемых во многих числовых методах решения инженерных и научных задач. Поэтому основная проблема, которую приходится решать программисту, создающему программы для НТР,— это, используя возможности Фортрана, обеспечить минимальное время выполнения программ при обработке данных большой длины и громадном объеме итераций.

5.3.2. Исследуем

Вычислительная техника впервые была применена в научных исследованиях. И сейчас наука, вооруженная ЭВМ, играет основную роль в ускорении прогресса. Все более усложняющиеся научные задачи помогают решать автоматизированные системы научных исследований (АСНИ). Главную часть таких систем представляют системы автоматизации научных экспериментов (САНЭ), основной особенностью которых является экспресс-анализ и обработка данных, поступающих от лабораторного оборудования, а также мгновенное принятие решения по управлению ходом эксперимента. Наиболее остро эта проблема стоит при быстротекущих процессах в физике элементарных частиц и химических реакций. Автоматизированное рабочее место экспериментатора (рис. 77) оборудуется дополнительно устройством сопряжения ЭВМ с датчиками, измерительными приборами и средствами регулирования исследуемого процесса. Управление экспериментом производится в соответствии с алгоритмом управляющей программы на основе поступающих от оборудования данных, а также команд эксперимента-

77

тора, ведущего работу с ЭВМ в диалоговом режиме. Обработку и хранение результатов эксперимента можно вести и на большой ЭВМ, соединенной с АРМом.

Программист, работающий в области автоматизации эксперимента, озабочен в первую очередь временем отклика его программ на поступающие от лабораторного оборудования сигналы. Здесь реакция может измеряться сотыми и даже тысячными долями секунды. За это время надо успеть не только принять сигнал, но и произвести его предварительную обработку, выдать сообщение на терминал или на другое оборудование. Программы, находящиеся «на переднем крае» таких систем, пишутся, как правило, на Ассемблере, обеспечивающем наиболее оптимальный двоичный код программы, а следовательно, и наивысшую скорость выполнения.

5.3.3. Проектируем

Проектирование в любой отрасли — это прежде всего построение всевозможных графических изображений. *Машиностроение* — это обязательное наличие чертежей деталей и изделий, *приборостроение* — это электрические схемы и монтаж печатных плат, *строительство* — это и планировка, и архитектурный поиск внешнего вида здания.

Современные системы автоматизации проектирования (САПР) позволяют вести проектирование комплексно, начиная с постановки задачи и кончая выдачей готовых чертежей.

В АРМ проектировщика входят графический дисплей, графопостроитель, устройство ввода графической информации. С помощью этих устройств он может составлять библиотеки стандартных графических элементов, а затем размещать их на плоскости (раскрой материала, расстановка оборудования в цехе, компоновка печатной платы и т. д.), составлять трехмерное изображение изделия и поворачивать его на экране в различных плоскостях. Например, чертеж зубчатого колеса легко строится путем вращения одного графического элемента — чертежа зубца по окружности колеса. ЭВМ преобразует изображение, позволит получить зеркальное отображение любого элемента, используя различные типы линий, автоматически произведет штриховку сколь угодно сложной фигуры, проставит размеры, радиусы и углы. Поиск наилучшего решения ведется на экране дисплея, а затем

выводится на графопостроитель. Это устройство позволяет получать на бумаге или на лавсановой кальке высококачественные цветные чертежи при помощи шариковых ручек или ручек с жидкими чернилами. При этом скорость вычерчивания при опущеном пере в рабочую позицию достигает 40 см/с, а скорость свободного хода пера — до 80 см/с. Встроенный микропроцессор, или контроллер графопостроителя, уменьшает нагрузку на управляющую ЭВМ за счет самостоятельного вычерчивания стандартных элементов по одной команде от ЭВМ.

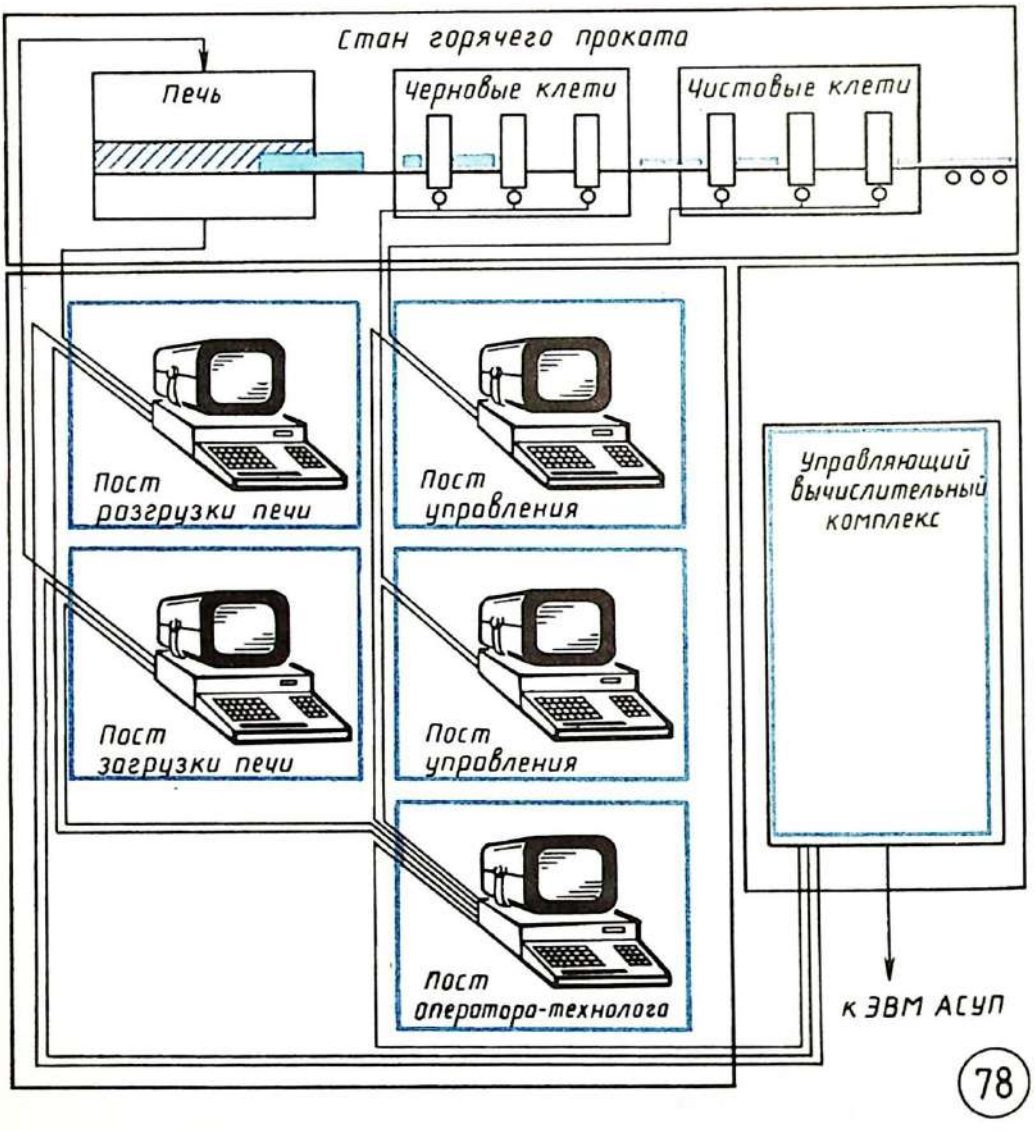
В нашей стране серийно выпускаются такие комплексы САПР, как АРМ-Р для конструирования радиоэлектронной аппаратуры, АРМ-М для конструирования изделий машиностроения, АРМ-С для проектирования зданий и сооружений в строительстве.

Большая часть программного обеспечения САПР предназначена для обработки графической информации. Для этих целей широко применяется язык Фортран, на котором, в частности, написаны многие пакеты графических программ, поставляемых в составе специализированных комплексов. Программисту, создающему программы для САПР, необходимо, владея такими языками, как Фортран, Паскаль, знать методы обращения к стандартным программам, ориентированным на графические построения и поддержку графических устройств.

5.3.4. Производим

Превращение раскаленного куска металла в длинную и абсолютно ровную по всей протяженности трубу, черной болванки в изящную блестящую деталь, разрозненных мотков провода в монтажный блок ЭВМ для непосвященного имеет магический оттенок. Но ничего, конечно, сверхъестественного в этом нет. Крепкие «руки» машин и механизмов, управляемые человеком, способны и не на такие чудеса. Применение вычислительных машин для управления технологическими процессами открывает необозримые перспективы в производстве.

Основные функции, возлагаемые на ЭВМ, которые работают в составе автоматизированных систем управления технологическими процессами (АСУ ТП),— это функции контроля, управления и оптимизации процессов, управления отдельными агрегатами и автоматическими линиями. На рисунке 78 показана схема применения ЭВМ в системе управления станом горячего проката металла. Стан представляет собой печь (в ней раскаляются заготовки) и ряд клетей, где производится под действием валков раскатка металла. Здесь требуется высокая точность расположения валков, чтобы добиться равномерной толщины листа. По заложенным программам машина сравнивает текущие значения параметров листа, поступающие от датчиков, с заранее определенными величинами — уставками и в случае выхода за допустимые пределы вырабатывает сигналы для оператора. Сигналы могут передаваться непосредственно на орган управления, минуя оператора. Это повышает скорость реакции системы и исключает ошибки, свойственные человеку. Технологические АРМы подключаются к управляющему вычислительному комплексу на базе мини-ЭВМ, который ведет учет расхода



материалов и полученных изделий, а также непосредственно управляет некоторыми технологическими операциями.

Выпуск изделий осуществляется на промышленном предприятии — заводе, фабрике. Предприятия состоят из цехов, в которых и устанавливаются технологические линии. Чтобы произвести изделие, надо выполнить множество разнообразных операций, а значит, требуется много технологических линий и цехов. Кроме них, на предприятии есть ряд других служб, которые обеспечивают материалами, сырьем, инструментами, создают нормальные условия труда людей, а также производят расчет их зарплаты. Таким образом, современное предприятие нуждается в согласованности действий всех своих подразделений. В этом ему помогает автоматизированная система управления

всем предприятием (АСУП). Задачи, решаемые АСУП, сводятся к оперативной обработке больших объемов информации, стекающейся со всех подразделений, и к представлению необходимой информации на различные уровни управления — от директора до бригадира. Во главе АСУП ставится большая машина класса ЕС ЭВМ с обширной магнитной памятью на лентах и дисках, где располагаются громадные базы данных. Основное время машины уходит на поиск необходимых данных и на их накопление. К главной ЭВМ подключаются УВК АСУ ТП, а также АРМы служб предприятия, на которых производится предварительная обработка данных и их ввод. Для получения необходимых справок производится обращение к большой машине, которая возвращает на дисплей или печатающее устройство АРМа запрошенную информацию.

Требование к программам, функционирующим в АСУ ТП,— обеспечение реального времени отклика на сигналы от датчиков, установленных на технологической линии. В этих случаях, как мы уже знаем, применяется язык Ассемблер; кроме того, в последнее время эффективные программы конструируются и на языках Си, Паскаль.

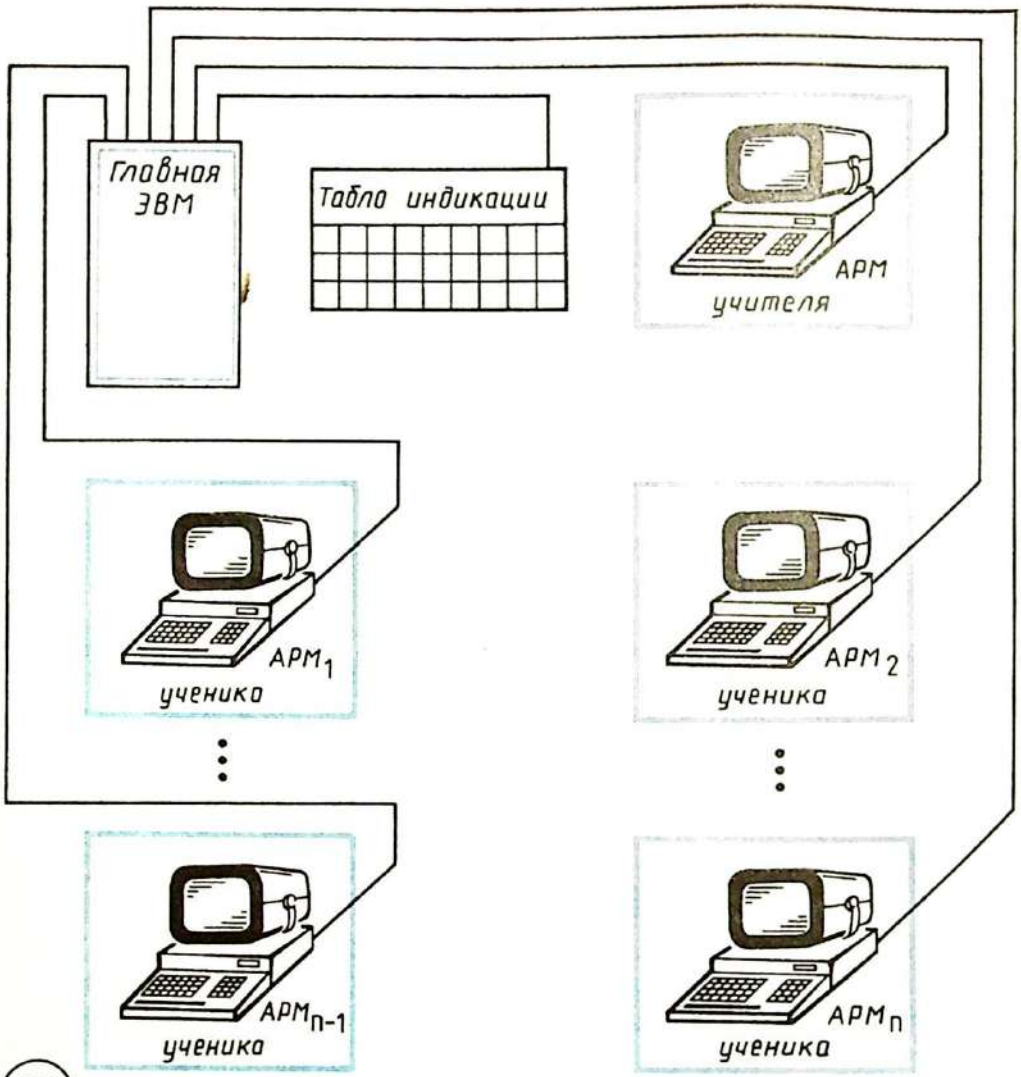
Программист, работающий в АСУП, должен в совершенстве владеть языками Кобол, ПЛ/1. Обработка экономической информации, составляющая сущность подобных систем, наиболее эффективно реализуется именно в этих языках.

Кроме знания языков, программист применяет в своей работе методы доступа к системам управления базами данных, а также навыки работы со стандартными программами ввода и вывода больших объемов данных.

5.3.5. Обучаемся

Прогресс в обучении в последнее время связан с применением технических средств обучения (ТСО). В классы пришли телевизоры и кинопроекторы, магнитофоны и всевозможные автоматические экзаменаторы. Однако эти меры являются промежуточным этапом на пути к повышению качества учебного процесса.

Объединить преимущества различных методов обучения позволяет автоматизированная система обучения (АСО), которая управляет самостоятельной работой ученика. Процессом обучения управляют программа ЭВМ и учитель. Учитель заполняет своей деятельностью все то, что не учтено в обучающей программе. Благодаря тому, что часть его функций возложена на программу, учитель получает возможность для индивидуальной работы с учениками, причем активный контакт учителя и ученика достигается за счет взаимодействия их ЭВМ, при этом сохраняется тайна общения. На рисунке 79 показана схема класса для АСО. Каждое место оборудовано персональной ЭВМ, в том числе и место учителя. Все машины связаны с главной, которая может находиться и в другом здании. Через нее учитель может определить общее задание для всех учеников, а затем подключиться к любому АРМу для индивидуальной работы. Результаты обучения, например оценки, могут



79

быть выведены на индикационное табло коллективного пользования. Кроме реализации обучающих программ, такой комплекс может вести журналы успеваемости, накапливать и обрабатывать статистическую информацию об успеваемости.

Обучающие программы благодаря возможностям ЭВМ намного превосходят простые учебники. Страницы учебника как бы оживают на экранах дисплеев, двигаясь и мерцая всеми цветами радуги. При этом машина ведет доверительный разговор, направляя и убеждая, подсказывая и шутя.

Программисты, посвятившие себя этому направлению, кроме владения языками высокого уровня — Паскалем, Бейсиком, Фокалом, должны обладать педагогическими навыками, способностями к импровизации,

широким кругозором. Немаловажное значение имеют и возможности программиста по «дизайну экрана» — созданию на экране дисплея живых, ярких, запоминающихся картинок. Тут надо чуть ли не владеть мастерством художника-мультипликатора!

5.3.6. Пишем и читаем

Постоянно расширяющийся круг знаний человечества влечет за собой стремительный рост числа текстовых документов. У нас в стране только годовой тираж книг достигает 2,5 млрд. экземпляров. Но, наверное, никто не считал, сколько еще выпускается проектно-конструкторской документации и научных отчетов, для чего имеется большая армия авторов, издателей, полиграфистов и людей других профессий. Однако дефицит необходимой текстовой информации растет. Основная причина этого явления — большая трудоемкость, а отсюда длительность выпуска документов. Из-за старения информации и отсутствия оперативности документ теряет свою ценность, будучи выпущенным не во время.

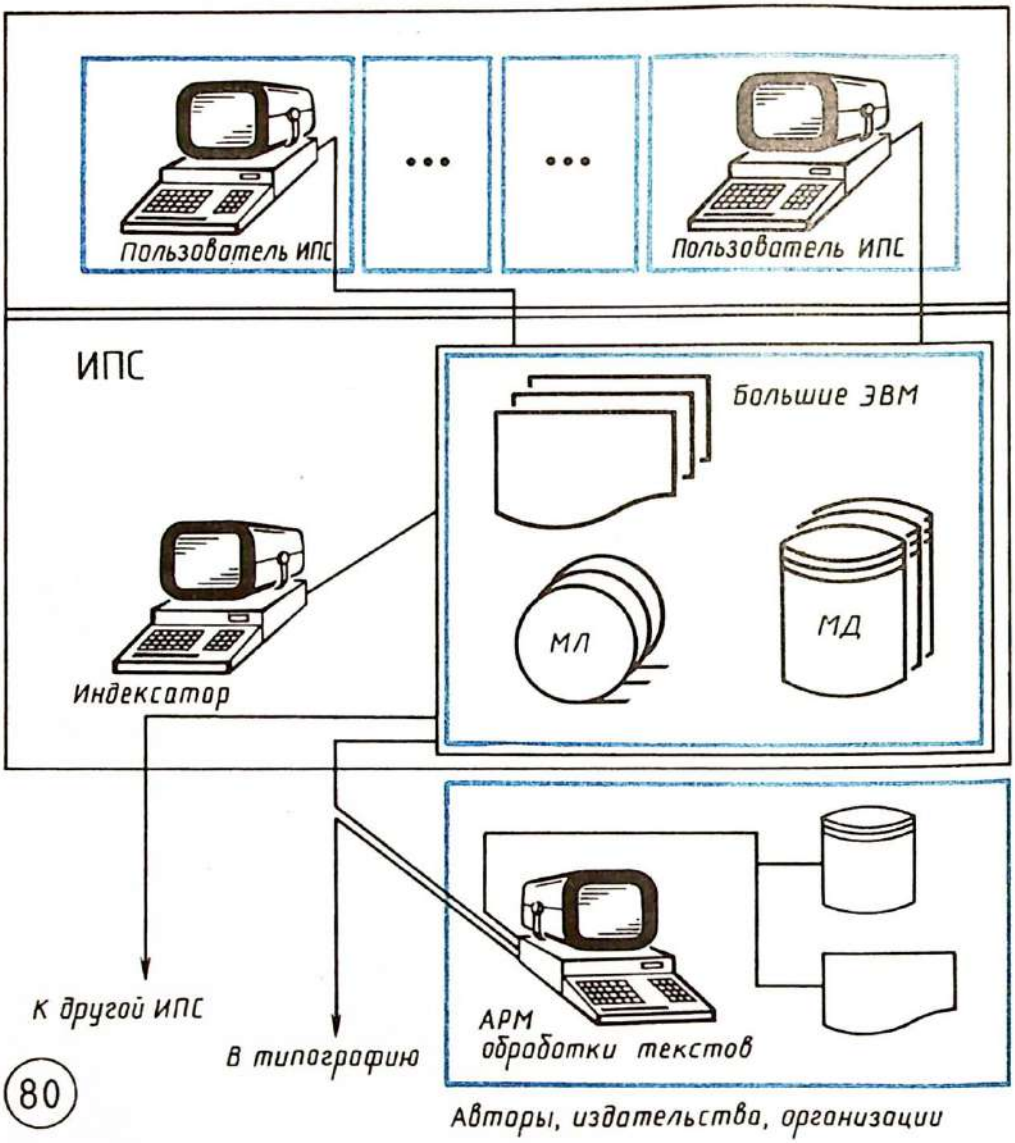
Наладить «текстовой конвейер» позволяют автоматизированные системы обработки текстовой информации (АСОТИ). На рисунке 80 приведена схема такой системы. Основным звеном ее является АРМ обработки текстов. Он строится на базе специального процессора для обработки текстов, а также включает дисплей, накопитель на магнитных дисках и печатающее устройство. Процессор позволяет производить операции форматирования текста, как выравнивание правой и левой границ, центрирование, разбивка на разделы, страницы, вставку и замену букв, строк и целых фрагментов текста. С помощью БД ведется учет документов и стандартных элементов текстов. Это позволяет формировать текст из элементов других текстов, что значительно ускоряет его подготовку.

Готовый текст может быть передан по линии связи в типографию, где автомат превратит его в пленки-формы для фотопечати. Кроме того, текст поступает в центральную автоматизированную информационно-поисковую систему (АИПС); там он помещается на хранение в базы данных больших ЭВМ. С рабочего места оператора ИПС ведется индексирование документа. По присвоенному индексу любой пользователь, подключенный к ИПС, может получить на свою ЭВМ документ. ИПС произведет и автоматический перевод текста на язык пользователя.

Создание систем обработки текстов — это работа с базами данных, линиями связи. Здесь некоторые программы или фрагменты программ пишутся на Ассемблере, а также широко применяются языки высокого уровня.

5.3.7. Укрепляем здоровье

Внедрение ЭВМ в медицину идет по двум направлениям. Первое связано с укрупнением лечебных центров, повышением объема медицинской информации. Здесь ЭВМ позволяет оптимальным образом



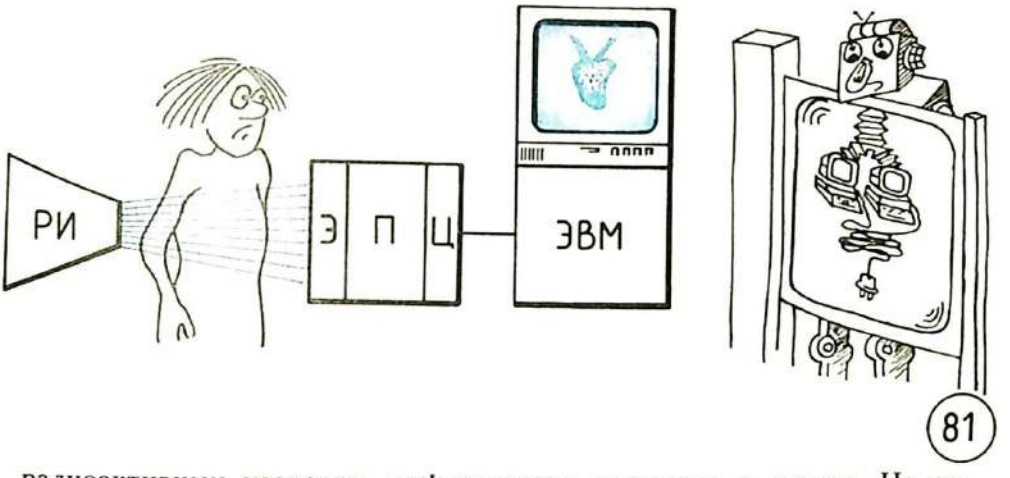
80

Авторы, издательства, организации

распределить нагрузку кабинетов, свести к минимуму время ожидания пациентом приема, ускорить и проконтролировать прохождение документов, автоматизировать накопление данных анализов и обследований, обеспечить своевременный вызов пациента на диспансерный осмотр и многое другое.

Но наиболее важным является применение ЭВМ в области диагностики заболевания, в частности в интроскопии — внутривидении, т. е. наблюдении объектов в оптически непрозрачных средах. Давняя мечта врача заглянуть внутрь организма без хирургического вмешательства сегодня становится явью.

Современная медицинская интроскопия использует разнообразные методы: рентгеновское излучение, гамма-излучение искусственных



81

радиоактивных изотопов, инфракрасное излучение и другие. Но во всех методах необходимо производить многоступенчатое преобразование изображений с целью приведения к виду, доступному для анализа оператором. На рисунке 81 в качестве примера представлена схема использования ЭВМ при рентгеновском исследовании.

Полученное на флюороскопическом экране изображение преобразуется в электрические сигналы. С помощью последующего преобразования эти сигналы превращаются в цифровую форму и затем передаются в память ЭВМ. Здесь начинается обработка полученной информации для формирования необходимого для анализа изображения. При этом исключается избыточная ненужная информация; производится сравнение нескольких изображений, вычитание изображений, их окраска.

Настоящую революцию в медицинской диагностике произвело создание томографа с ЭВМ. Это устройство позволяет получить изображение слоя, лежащего на определенной глубине исследуемого органа, другими словами, произвести его «рассечение». Двухмерное изображение слоя получают по серии разноракурсных одномерных сигналов путем проведения большого числа вычислений.

Как ты уже догадался, инженеру, автоматизирующему медицину, необходимо владеть и навыками построения быстрых программ, т. е. необходимо быть хорошим программистом. Он должен уметь вести инженерные расчеты, владеть методами обработки изображений, должен знать особенности работы базы данных. В общем, проблема компьютеризации здравоохранения не менее многообразна, чем сама проблема сохранения здоровья.

5.4. РАБОТАЕТ ЭВМ

Еще в 1973 г. ученые насчитали 2500 «профессий» ЭВМ, из которых 900 связаны с наукой и техникой, а 200 — с производством. Если удельный вес «рабочих профессий» вычислительной машины в то время был невелик — менее 10%, то сейчас, благодаря освоению микропроцессо-

ров, положение резко изменилось. Широкое применение встраивания микропроцессоров позволяет рассматривать любое производственное оборудование не иначе как устройство ввода-вывода ЭВМ. Смелое внедрение компьютеров дает не сотни, а тысячи возможных применений микропроцессора. И вновь, в который раз, хочется сказать о роботах.

5.4.1. В роботах

Первый промышленный робот (ПР) был выпущен в США в 1962 г. В 1972 г. в мире насчитывалось уже около 3 тыс. роботов, а в 1980 г. их было свыше 20 тыс. У нас в стране первые ПР появились в 1971 г.; это УМ-1 и «Универсал-50».

Промышленный робот строится по схеме «рука + мозг + глаз». Роль руки играет манипулятор — механическое устройство с захватом. «Мозг» робота — это, конечно, ЭВМ. В качестве «глаза» применяются какие-либо сенсорные приборы. Пока наибольшее распространение получили телекамеры, фотоэлементы.

В зависимости от назначения и сложности выполняемых операций, а также периода развития техники промышленные роботы по аналогии с ЭВМ условно делят на поколения.

Наиболее широко на производстве пока еще применяются роботы *первого поколения* — автоматические манипуляторы с жестким программным управлением. Их механические устройства и программные (управляющая микроЭВМ) относительно легко перестраиваются на разнообразные операции в широком диапазоне.

В стадии исследований и проектирования находятся роботы *второго поколения,* так называемые адаптивные роботы со специальной системой «очувствления». Такие роботы способны, в соответствии с заложенной программой, приспосабливаться к изменяющимся условиям технологического процесса и выбирать оптимальный режим работы.

Наибольший интерес представляют роботы *третьего поколения,* с так называемым искусственным интеллектом. Они способны распознавать неизвестную или меняющуюся обстановку, вырабатывать соответствующее решение о своих последующих действиях, «самообучаться» для накопления «опыта» и применения его в похожих ситуациях.

Фактически только роботы третьего поколения имеют «глаз». Наличие некоторых чувствительных устройств помогает роботу воспринимать форму, размеры, положение захваченной детали, что позволяет ему самостоятельно координировать свои действия. Такие роботы могут выполнять даже некоторые конструкторские работы, работы в заводской лаборатории и на испытательной станции.

Призвание робота не только в машиностроении. Уже появились робототехнические системы, заменяющие ручные операции в строительстве и монтажных работах, в добыче полезных ископаемых, в сельском хозяйстве.

Изучение и освоение морских и океанских глубин возможно только с применением подводных роботов. Кстати, первый советский робот,

созданный в 1968 г., предназначался для подводных работ и управлялся от ЭВМ. Адаптивный характер таких роботов поможет приспособиться к изменениям структуры дна, прозрачности воды, освещенности. Робот, например, «видит» в инфракрасных лучах. А это ценнейшее свойство для определения температуры в морских пучинах. Даже на глубине в несколько тысяч метров робот очертит зону с определенной температурой воды. Это поможет нанести на карту океанского дна районы подводного вулканизма, богатые полезными ископаемыми.

Космос — любимая фантастами среда обитания роботов. Он уже сегодня дал прописку первым манипуляторам, среди которых пионером является наш луноход. Дистанционно управляемые роботы необходимы для производственной и исследовательской деятельности в самых различных экстремальных условиях, в которых невозможно пребывание человека. Крайне низкие или высокие температуры, огромное давление или глубокий вакуум, вредные излучения встречаются не только в космическом пространстве, но и при некоторых технологических процессах на Земле.

Работа для робота находится во все бóльших сферах человеческой деятельности. Всего полторы минуты требуется роботу-антропометру для обмера фигуры при приеме заказов на индивидуальный пошив одежды. Еще несколько минут — и он сконструирует выкройку в соответствии с выбранным заказчиком фасоном, а затем автоматическое устройство — графопостроитель, управляемый роботом, вычертит выкройку на бумаге.

Все у́же круг действий, остающихся монополией человеческого интеллекта, где не попробовал своих сил робот. Есть роботы шахматисты, музыканты, переводчики. Не чужды роботу и человеческие увлечения. Большой популярностью у сборной ГДР по боксу пользуется на тренировках робот-боксер. Кожаный манекен может управляться по радио или кабелю, делать движения в сторону, вперед и назад. Сам робот, к счастью, ударов не наносит, однако неожиданные выпады по приказу тренера напоминают удары.

О роботах можно рассказывать долго, но давай вернемся к *промышленному производству,* чтобы выяснить, как оно преобразуется в условиях компьютеризации.

5.4.2. В гибких производствах

Цикл «научное исследование — проектирование — производство» до недавнего времени имел слабо связанные этапы; ограничение единства этого цикла заключалось в том, что большие объемы передаваемой информации — от задания ученым на разработку до склада готовой продукции — «неслись» в многотомной технической документации и преобразовывались на стыке этапов людьми. Не умаляя возможностей *Homo sapiens,* надо сказать о низких скоростях преобразования и ошибках. Большая доля ручного труда связана с тем, что из трех компонентов производственного процесса — материала, энергии и информации —

основное внимание уделялось первым двум. Об информации заговорили лишь тогда, когда изобрели ЭВМ.

Но появление вычислительной машины в заводских корпусах на первых порах ничего, кроме дополнительных трудностей, не принесло. Автоматизация отдельных фрагментов управления производством свелась к перемалыванию в мощных ЭВМ вводимых в нее ошибочных данных и выдаче этих ошибок в виде красивых листингов.

Только переход от одной центральной ЭВМ, перегруженной большим количеством задач, к распределенной многопроцессорной системе позволит повысить качество и надежность управления, обеспечит гибкость при изменении номенклатуры выпускаемых изделий. Эффект гибкости достигается также за счет программной реализации функций производственного оборудования. Ведь как было раньше? Функции станка закладывались в его конструкцию аппаратно. Малейшее изменение функций, т. е. требование выпуска новых изделий, приводило к замене механических частей оборудования, а то и всего станка, что, конечно, долго и дорого. Теперь в станок с программным управлением достаточно ввести новую программу, чтобы он перестроился на новые операции.

Подобное оборудование дает наибольший эффект, когда оно собрано в единый роботизированный технологический комплекс (РТК), действующий совместно с другими средствами автоматизации производства. Это может быть роботизированный конвейер или целый цех. Обычно в состав такого производства входят станки с числовым программным управлением (ЧПУ), промышленные роботы, измерительные машины (ИМ) и автоматизированный склад материалов и готовой продукции. РТК, объединяясь со средствами АСУ ТП и АСУП, ведут к развитию гибких автоматизированных производств (ГАП). Наконец, увязка в единый комплекс средств АСНИ, САПР и ГАП дает гибкую производственную систему (ГПС). На цветной вклейке II, *б* показана схема такой системы.

Наибольшее распространение РТК получают в машиностроении, особенно в мелкосерийном производстве, где часто меняется номенклатура и объемы выпускаемой продукции. Основа машиностроения — металлообрабатывающие станки. Наиболее развитые из них — обрабатывающие центры с автоматической сменой инструмента и станки с ЧПУ. Станок с ЧПУ состоит из пульта оператора, микропроцессора, памяти, таймера, терминала, блока управления приводом. Практически станок с ЧПУ отличается от робота тем, что он неподвижно закреплен на станине и не имеет манипулятора. Программы для ЧПУ разрабатываются в САПР и передаются на станок по линии связи, на магнитном гибком диске или на перфоленте.

Промышленные роботы в РТК выполняют работы, связанные с перемещением инструмента или предмета труда. Это сборка изделий, сварка, окраска. Типовой ПР содержит основание, которое может перемещаться, например, по рельсам, систему управления на базе микроЭВМ, механизмы переноса, угловой ориентации и захвата манипулятора.

Детали, изготовленные на автоматизированном оборудовании, необходимо контролировать. В машиностроении проблема контроля связана с измерением геометрических размеров. В этой области вычислительной технике тоже нашлось применение. Автоматические измерительные машины имеют в своем составе механическую часть для установки измеряемой детали и ее перемещения, измеряющую систему, которая определяет перемещение детали, систему ощупывания, фиксирующую касание щупом поверхности детали, ЭВМ, осуществляющую решение задач измерения. Например, достаточно сложный профиль имеет лопатка паровых или газовых турбин. Ее измерение производится в 25 сечениях по 30 точек в каждом. В памяти ЭВМ заложены данные о теоретическом профиле детали — значения этих 750 точек. Задача состоит в том, чтобы сравнить полученные значения замеров с теоретическим профилем и найти отклонения. ЭВМ решает эту задачу в реальном масштабе времени, т. е. параллельно с обработкой детали. Если замечено отклонение, то корректируется тут же программа станка с ЧПУ и дефект устраняется.

По затратам труда одно из первых мест на предприятии занимают погрузочно-разгрузочные и транспортно-складские работы. К таким работам относятся подача заготовок к технологическому оборудованию, разгрузка материалов, штабелирование и комплектование изделий в зоне хранения (на складе). Применение автоматизированного оборудования в этом хозяйстве позволяет решать задачи систематизации деталей, их поиска и транспортирования по заданной программе, размещение их на требуемых стеллажах.

Пожалуй, наиболее сложные и остроумные программы создаются для встраиваемых ЭВМ, например, в роботы, в станки с ЧПУ, измерительные машины и др. Для этих применений программист пишет программы в условиях жестких требований, например, ко времени их выполнения, объему занимаемой памяти (ведь программе выполняться в микропроцессоре, порой занимающем в автоматизируемом устройстве крошечный уголок), точности обрабатываемых данных. Здесь незаменим Ассемблер, но иногда приходится программировать непосредственно в двоичных кодах. Хорошее знание архитектуры микропроцессора, динамики его работы, устройств связи и исполнительного оборудования позволяет создавать эффективные программы.

Итак, гибкие производства входят в жизнь. Этот новый вид технических систем, кроме существенного экономического эффекта, значительно повысит культуру и производительность труда, поможет решить проблему недостатка рабочей силы. Тесная связь науки с производством, выраженная набором материальных средств хранения и переработки информации — прежде всего электронной вычислительной техникой,— непременное условие ускорения научно-технического прогресса.

5.4.3. В связи

Начиная с первой главы, мы часто упоминали о связи. Это и не удивительно. Ведь, с одной стороны, сейчас трудно представить нашу жизнь без телефона и телевизора, без радио и телеграфа. А с другой стороны, необходимость передачи и распределения все большего потока информации, да еще с высокой скоростью, поставила средства связи в трудное положение. Традиционный принцип аналоговой связи, т. е. передачи информации непрерывными уровнями напряжения или тока, меняющимися по величине для кодирования символов, исчерпал себя. Каналы связи стали буквально захлебываться под напором информации. Увеличение числа линий связи не спасало положения. Поползла вверх стоимость линейных сооружений связи и аппаратуры. А потоки информации все прибывали. Уже земной шар плотно опутали кабели и провода разной толщины и длины. Эфир насыщен радио- и телевизионными волнами. И все труднее дозвониться к товарищу; растут очереди у междугородных автоматов, все больше напрягается ухо в тщетной надежде различить в телефонной трубке знакомый голос среди хрипа, треска и бульканий. Да, еще одной проблемой связи стало качество передаваемой информации. Особенно необходимо повышенное качество при передаче цветного телевидения, факсимильной информации (передача графических изображений и текстов).

Путь к совершенству связи люди знали давно. Заключается он в использовании цифровых способов передачи и обработки информации, т. е. кодирования символов с помощью все тех же нулей и единиц. Понятно, возможным это стало только с развитием вычислительной техники. По сравнению с существующими сетями цифровые сети обеспечивают почти 20-кратное возрастание скорости передачи информации, резкое снижение расходов на эксплуатацию и техническое обслуживание. Но самое главное состоит в том, что цифровые способы передачи информации позволяют объединить различные виды электросвязи — телефонную, телеграфную, факсимильную, передачу данных между ЭВМ — в единое целое, получившее название *интегральная сервисная цифровая сеть связи* (ИСЦСС). Действительно, сегодня даже к простому жилому дому тянутся три вида связи. Под землей вьется телефонный кабель, между крышами провисают провода радиовещания, а сами крыши утыканы антеннами телевидения. Учреждения и предприятия используют еще большее количество видов связи. На создание связи тратится много ценных материалов, а обслуживание ее требует больших затрат человеческого труда. Цифровой же связи безразлично, что «бежит» по ее каналу — телевизионная картинка или поздравительная телеграмма, человеческая речь или байты данных. Специальная аппаратура приводит все виды сообщений к единому виду — цифровому пакету, который затем начинает путешествие по линиям связи. На приемном конце из этого пакета выделяется информация в первоначальном виде. И если продолжать говорить о проблемах связи, то стоит упомянуть еще о двух. Первая — это доставка информации потреби-

телю через промежуточные пункты. Между Москвой и Владивостоком, конечно, нет прямого кабеля. Сообщение на этом пути проходит через многие города, где оно анализируется и отправляется дальше по требуемому маршруту. Вот этот процесс коммутации наиболее просто выполняется тогда, когда информация представлена в цифровом виде, потому что роль коммутирующего устройства выполняет микропроцессор, определяющий по цифровому коду направление передачи информации.

А вторая проблема — техническое обслуживание и административное управление — эффективно решается с помощью все того же микропроцессора или дополнительной ЭВМ. Коммутационный процессор сам проведет тестовую диагностику аппаратуры в случае неисправности и укажет персоналу точное место ее возникновения. Он же передаст управляющей ЭВМ данные для начисления платы за пользование каналами связи, ведения статистики работы и т. д.

Многого из сказанного сегодня еще нет. Но научно-технические проблемы создания ИСЦСС уже начали решаться. Формирование Единой автоматизированной сети связи (ЕАСС) страны на базе новейших систем передачи информации делает первые шаги, и тебе дана возможность стать участником этого грандиозного свершения.

5.4.4. ЭВМ V

Ты приближаешься к концу книги. И последний параграф в ней оказывается небольшим не потому, что мало информации, а скорее наоборот. Но как это ни удивительно, предсказывать на ближайшее будущее значительно труднее, чем на отдаленную перспективу. Если, следуя писателю-фантасту Артуру Кларку, мы скажем, что к 2090 г. будет создан мировой электронный мозг, то, кроме восторгов такой прогноз ничем не чреват. Но заявление о начале выпуска в 1990 г. ЭВМ пятого поколения должно быть наиболее ответственным. Однако, как заметил другой писатель-фантаст Станислав Лем, «любая, даже самая точная наука развивается не только благодаря новым теориям и фактам, но и благодаря домыслам и надеждам ученых».

Безусловно, в прогнозировании вопрос вопросов — точность предвидения. Опыт последних лет показал, что степень оправдываемости прогноза вполне удовлетворительна. Поэтому прислушаемся к мнению смелых ученых о начале качественно нового этапа в развитии вычислительной техники.

Этот этап, связываемый с созданием пятого поколения ЭВМ, должен уменьшить диспропорцию в развитии базовых составляющих индустрии вычислительной техники. Их четыре:

— технология производства оборудования;
— архитектура ЭВМ;
— технология программирования;
— функциональные возможности (интеллектуализация) ЭВМ.

Последняя является характерной именно для проектов ЭВМ пятого поколения. Отставание этой составляющей от развития технической

базы видно из того, что, несмотря на достигнутую доступность средств вычислительной техники широкому кругу пользователей, эффективное ее применение требует специальной высокой квалификации программиста. Естественным путем решения этой проблемы является использование ЭВМ без программирования, т. е. по формуле «ЭВМ + естественный язык». Эта формула будет реализована на сочетании естественноязыковых, акустических, графических средств. Машина будет понимать пользователя буквально «с полуслова».

Такие успехи немыслимы без программной поддержки изготовителей ЭВМ. Их задача — разработать готовые стандартные модули-конструктивы, которые затем любой пользователь может собрать в необходимую ему конфигурацию — программу, настроив машину «для себя».

Классической неймановской архитектуре ЭВМ с последовательной обработкой команд уже не справиться с быстро усложняющимися требованиями к ЭВМ. Развитие — в децентрализации, распределенности вычислительного процесса в машине, обеспечении параллельности выполнения нескольких команд программы.

Пятое поколение ЭВМ связано и с новой элементной базой — сверхбольшими интегральными схемами (СБИС), содержащими $10^6...10^7$ полупроводниковых элементов в кристалле. Скорость обработки достигнет $10^{11}...10^{12}$ операций в секунду, объем оперативной памяти 10^{12} байт. Внедрение техники кремниевого программирования, т. е. превращения программы в аппаратную подсистему, позволит обеспечить реализацию «заказных» микросхем, достаточно просто включаемых в ЭВМ и расширяющих их функциональные возможности. По-видимому, приведенные концепции не только составят основу нового поколения ЭВМ, но и будут формировать представление о будущих поколениях. Тебе их создавать, тебе с ними работать!

Учебное издание

Нестеренко Александр Васильевич

ЭВМ И ПРОФЕССИЯ ПРОГРАММИСТА

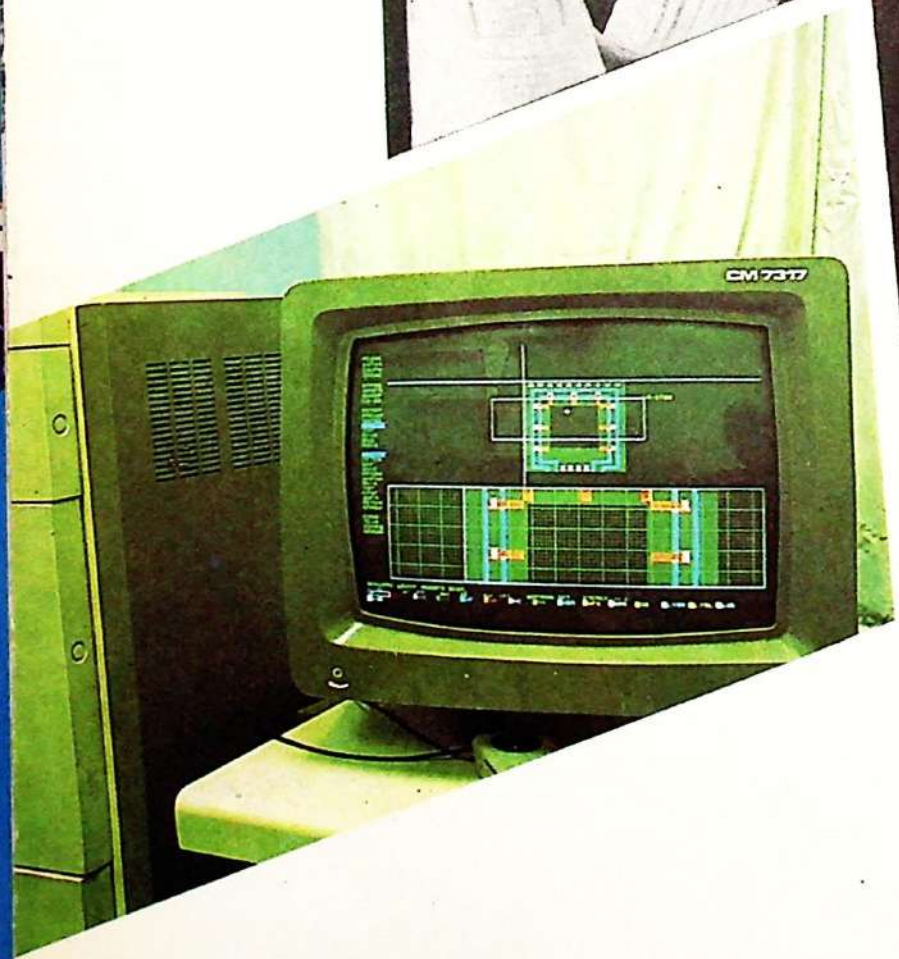
Зав. редакцией *В. А. Обменина*. Редактор *О. В. Серышева*. Младший редактор *О. В. Агапова*. Художники *В. С. Давыдов, О. М. Шмелев, А. В. Нестеренко*. Художественный редактор *В. М. Прокофьев*. Технический редактор *И. Е. Хилобок*. Корректор *И. Н. Панкова*.

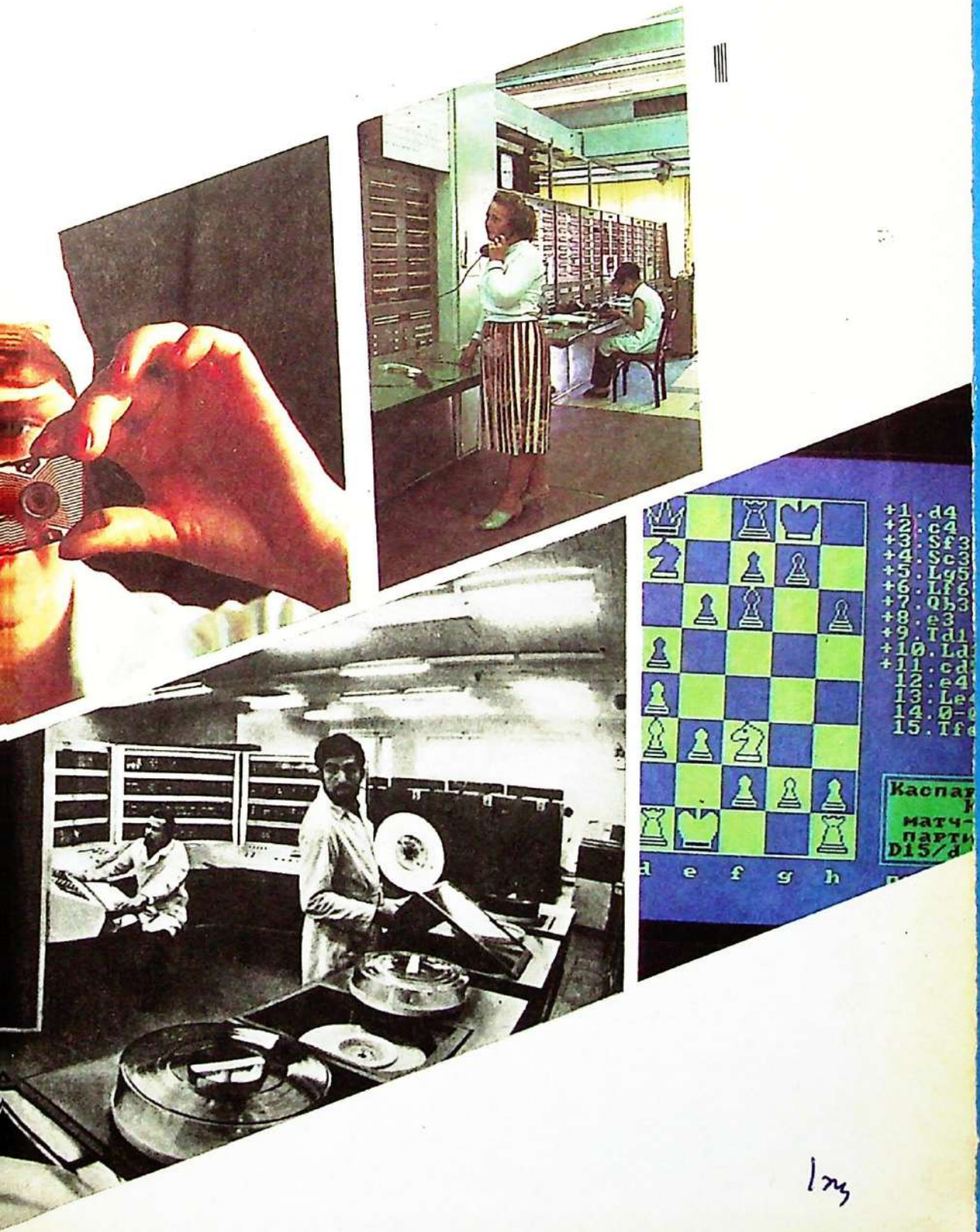
ИБ № 12061

Сдано в набор 23.03.89. Подписано к печати 14.12.89. Формат 60 × $90^1/_{16}$. Бумага офсетная № 2. Гарнитура Баскервиль. Печать офсетная. Усл. печ. л. 10,00 + 0,38 форз.+ 0,50 вкл. Усл. кр.-отт. 23,63. Уч.-изд. л. 11,07 + 0,31 форз. + 0,59 вкл. Тираж 150 000 экз. Заказ 2448. Цена 1 р.

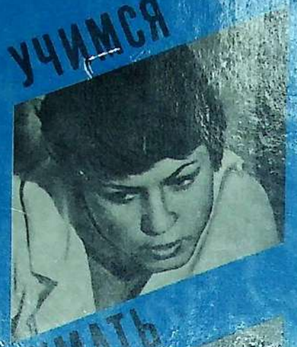
Ордена Трудового Красного Знамени издательство «Просвещение» Государственного комитета РСФСР по делам издательств, полиграфии и книжной торговли. 129846, Москва, 3-й проезд Марьиной рощи, 41.

Отпечатано с диапозитивов Ордена Трудового Красного Знамени ПО «Детская книга» Госкомиздата РСФСР. 127018, Москва, Сущевский вал, 49 на Смоленском полиграфкомбинате Госкомиздата РСФСР. 214020, Смоленск, ул. Смольянинова, 1.


CM 7317

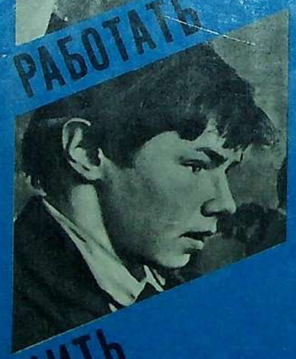

+1.d4
+2.c4
+3.Sf3
+4.Sc3
+5.Lg5
+6.Lf6
+7.Qb3
+8.e3
+9.Td1
+10.Ld
+11.cd
12.e4
13.Le
14.0-0
15.Tfe
Каспар
матч-
парти
D15/d
a e f g h

УЧИМСЯ



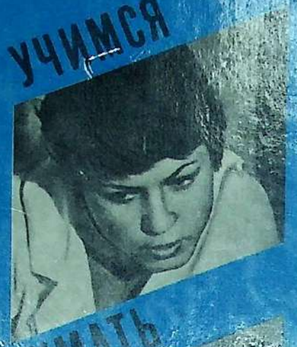
ДУМАТЬ

РАБОТАТЬ



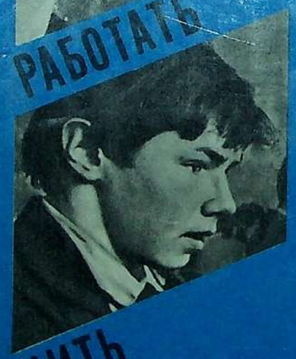
ЖИТЬ

УЧИМСЯ



ДУМАТЬ

РАБОТАТЬ



ЖИТЬ