1986 г. Август *Том 149, вып. 4*

УСПЕХИ ФИЗИЧЕСКИХ НАУК

ФИЗИКА НАШИХ ДНЕЙ

530.145+621.142

КВАНТОВОМЕХАНИЧЕСКИЕ ЭВМ *)

Р. Ф. Фейнман

ВВЕДЕНИЕ

Задача этой работы — проанализировать некоторые ограничения возможностей ЭВМ, обусловленные законами физики. Например, Беннет подробно исследовал диссипацию свободной энергии, которая должна сопровождать процесс вычислений [1], и обнаружил, что практически диссипация вообще отсутствует. Он задал мне вопрос о том, какие ограничения квантовая механика и соотношение неопределенностей накладывают на работу ЭВМ. Изучая эту проблему, я обнаружил, что в данном случае нет принципиальных ограничений возможностей ЭВМ, за исключением естественных размерных ограничений, если считать, что логические элементы ЭВМ не могут быть меньше одного атома. Мы рассматриваем здесь идеальные устройства; влияние небольших неидеальностей будет обсуждаться позднее. Мы будем исследовать весьма общие вопросы; наша задача состоит в том, чтобы написать гамильтониан определенной системы, которая может служить в качестве ЭВМ. Вопросы о том, является ли такая система наиболее эффективной и как она может быть наилучшим способом реализована, нас здесь не интересуют.

Так как законы квантовой механики обратимы во времени, мы должны рассматривать вычислительные устройства, подчиняющиеся такому закону обращения. Это было отмечено уже Беннетом [1] и Фредкином и Тоффоли [2], и этому вопросу уделялось немалое внимание. Поскольку не предполагаю у читателя знакомства с этими работами, я дам здесь краткий обзор проблемы, включая результаты, полученные Беннетом [1,3], которые ниже будут подтверждены и для нашей квантовомеханической системы.

Из теории вычислений следует, что универсальная ЭВМ может быть построена в виде достаточно сложной сети взаимосвязанных первичных элементов. Следуя стандартному классическому анализу, мы представим эти взаимные связи в виде проводников, по которым могут передаваться два стандартных напряжения, представляющие нуль и единицу. Нам достаточно иметь всего два таких первичных элемента — выполняющих операции НЕ и И (вообще говоря, достаточно всего одного элемента НЕИ = НЕ И, по-

*) Feynman R. P. Quantum Mechanical Computers. — Доклад на пленарном заседании совместного совещания Международного совета по квантовой электронике и Комиссии по лазерам и электрооптике 19 июня 1984 г., Анахейм, США.— Перевод Б. Ф. Полковникова под ред. И. И. Мазина.

скольку, если на один вход подана единица, выход элемента НЕИ представляет операцию НЕ по отношению к другому входу). Это проиллюстрировано на рис. 1. С логической точки зрения мы должны рассмотреть и механизм

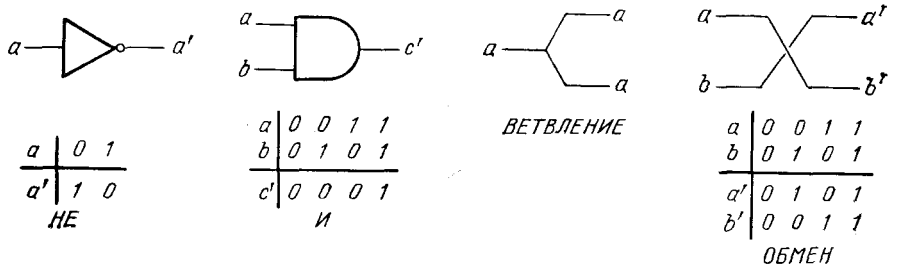


Рис. 1. Первичные элементы НЕ, И, ВЕТВЛЕНИЕ и ОБМЕН

действия проводников, ибо в других системах, в частности в нашей квантовомеханической системе, проводников как таковых может и не быть. Мы видим, что в действительности имеем еще два первичных элемента, выполняющих операции ВЕТВЛЕНИЕ, когда два проводника соединяются с одним, и ОБМЕН, когда два проводника перекрещиваются. В обычных ЭВМ операции НЕ и НЕИ осуществляются транзисторами, например, как показано на рис. 2.

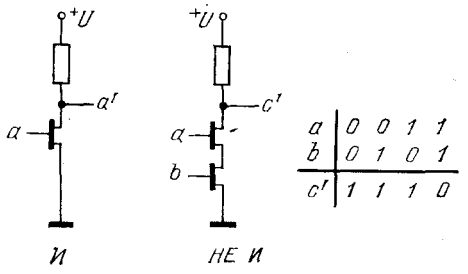


Рис. 2. Транзисторные цепи для элементов НЕ и НЕИ

Какова минимальная свободная энергия, которая должна быть затрачена при работе идеальной ЭВМ, состоящей из таких элементов? Поскольку при выполнении, например, операции И на выходной линии c' появляется одно из двух значений, вне зависимости от того, какое значение на ней было перед этим, изменение энтропии составляет $\ln 2$. При температуре T это соответствует выделению тепловой энергии $kT \ln 2$. Многие годы считалось, что это и есть то минимальное количество теплоты, которое должно быть затрачено для выполнения одной элементарной операции.

В настоящее время этот вопрос носит чисто академический характер. В реальных машинах вопрос о рассеянии энергии является весьма серьезным, но системы на транзисторах на самом деле рассеивают около $10^{10}kT$! Как указывал Беннет, это возникает потому, что для изменения напряжения на проводнике мы замыкаем его на землю через сопротивление [3]; для восстановления напряжения на проводнике мы передаем на него заряд вновь через сопротивление. Расход энергии можно значительно снизить, если запасать энергию в индуктивности или в другом реактивном элементе. Однако ясно, что в настоящее время весьма трудно создавать индуктивные элементы на кремниевых подложках. Даже сама Природа, копируя информацию с помощью ДНК, затрачивает около $100\ kT$ на каждый бит. Таким образом, нам еще очень далеко до значения $kT \ln 2$, и кажется просто смешным рассуждать о том, что даже это значение является слишком высоким и минимум на самом деле равен нулю. Мы, однако, намерены дальше заниматься еще более смехотворными вещами, рассматривая плотность записи в один бит на атом (в настоящее время эта плотность составляет один бит на 10^{11} атомов). Профессорам, вроде меня, такая ерунда представляется чрезвычайно увлекательной. Надеюсь, что и читатель найдет ее увлекательной и любопытной.

Беннет показал, что приведенный выше предел $kT \ln 2$ является неверным, так как нет необходимости использовать необратимые элементы. Вычисления могут производиться обратимыми машинами, содержащими только обратимые элементы. При выполнении этого условия минимум требуемой свободной энергии не зависит от сложности или числа логических шагов вычислений. Если даже этот минимум существует, он, скорее, равен kT на один бит выходной информации. Но даже и это значение, которое можно рассматривать как свободную энергию, необходимую для того, чтобы очистить ЭВМ для дальнейшего использования, может рассматриваться как затраченное в процессе последующей обработки результата, т. е. как необходимое для передачи выходной информации в некоторую другую точку. Этот предел достигается только в идеальном случае, если мы проводим вычисления с обратимым компьютером, работающим с бесконечно малой скоростью.

ВЫЧИСЛЕНИЯ С ПОМОЩЬЮ ОБРАТИМОЙ МАШИНЫ

Рассмотрим теперь три обратимых первичных элемента, которые могут быть использованы для создания универсальной ЭВМ [4]. Первым элементом является элемент НЕ, который очевидным образом не теряет информацию

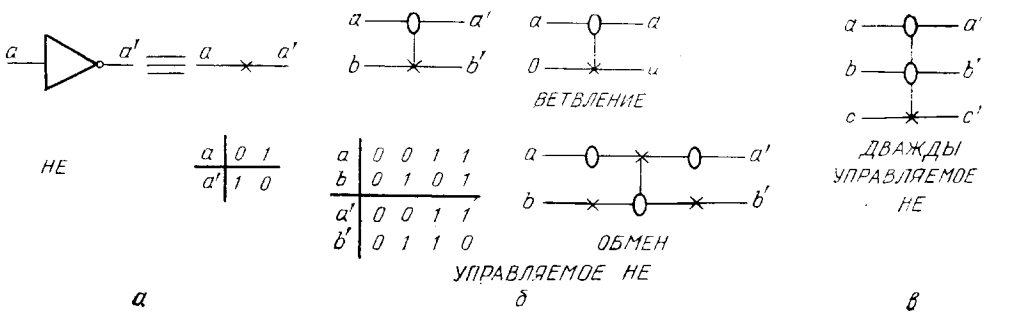


Рис. 3. Обратимые первичные элементы НЕ, УПРАВЛЯЕМОЕ (НЕ ВЕТВЛЕНИЕ И ОБМЕН) и ДВАЖДЫ УПРАВЛЯЕМОЕ НЕ

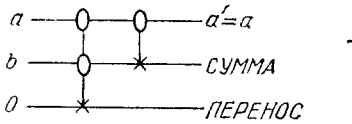
и является обратимым; обратной операцией является также операция НЕ. Поскольку стандартное обозначение этого элемента не является симметричным, мы используем знак $\times$ (рис. 3).

Следующий элемент мы назовем УПРАВЛЯЕМЫМ НЕ (см. рис. 3). Он имеет два входа a и b и два выхода a' и b'. Выход a' всегда совпадает с a, который является управляющим параметром. Если на нем есть сигнал ($a = 1$), то выход b' представляет собой НЕ b. В противном случае (видите, я не профессиональный программист, а то бы я сказал: «Иначе...») b остается неизменным, $b' = b$. Обратная операция для этого элемента совпадает с прямой. Величина b' в действительности является симметричной функцией a и b, называемой ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR): выход равен единице, если a или b, но не оба вместе равны единице. Это похоже на сумму a и b по модулю 2 и может быть использовано для сравнения a и b, выдавая на выходе единицу как сигнал того, что они различны. Обратите внимание, что сама по себе функция ИСКЛЮЧАЮЩЕЕ ИЛИ не является обратимой. Например, если значение b' равно нулю, мы не можем сказать, является ли оно результатом $(a, b) = (0, 0)$ или $(a, b) = (1, 1)$; однако наличие неизменной линии $a' = a$ позволяет нам избежать двузначности.

Мы представим оператор УПРАВЛЯЕМОЕ НЕ, изобразив значок 0 на управляющем проводнике, связанный вертикальной линией со значком $\times$ на управляемом проводнике. С помощью этого элемента можно также осуществлять операцию ВЕТВЛЕНИЕ, ибо если $b = 0$, то и a', и b' равны a. Эта функция КОПИЯ далее окажется весьма важной. Можно также смоде-

лировать функцию ОБМЕН — три таких элемента, соединенных последовательно с попеременным изменением управляющей линии, дают обмен информацией на линиях (см. рис. 3).

Однако оказывается, что комбинаций только этих двух элементов недостаточно для того, чтобы выполнять произвольные логические операции. Необходимы элементы, содержащие три линии. В качестве такого элемента мы выбрали элемент, который можно назвать ДВАЖДЫ УПРАВЛЯЕМЫМ НЕ. Здесь (см. рис. 3) мы имеем две управляющие линии a и b, которые дублируются на выходе и которые приводят к выполнению операции НЕ на третьей линии только тогда, когда сигнал есть на обеих управляющих линиях ($a = b = 1$). В противном случае $c' = c$. Если на третий вход подан нуль, то он превращается в единицу, только если и a и b равны единице, т. е. для них выполняется операция И (см. таблицу на с. 677). Три комбинации a и b, а именно (0, 0), (0, 1) и (1, 0), дают один и тот же результат (нуль) после выполнения функции И над входами a и b, так что требуется 2 бита информации для разрешения неоднозначности. Значения входов a и b сохраняются на выходах линий a' и b', так что эта функция может быть обратима (обратна, по сути дела, самой себе). Заметим, что функция И (a, b) есть перенос суммы a и b в следующий разряд, так как $1 + 1 = 2_{10} = 10_2$.



a b	a' b' c'
0 0	0 0 0
0 1	0 1 0
1 0	1 1 0
1 1	1 0 1

Рис. 4. Сумматор

Известно, что из этих элементов и их комбинаций может быть составлена любая логическая цепь, и, действительно, в теории вычислений доказывается существование универсального компьютера. Проиллюстрируем это

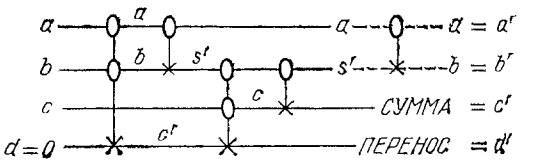


Рис. 5. Полный сумматор

небольшим примером. Прежде всего, разумеется, как можно видеть из рис. 4, мы можем построить сумматор, последовательно используя элементы ДВАЖДЫ УПРАВЛЯЕМОЕ НЕ и УПРАВЛЯЕМОЕ НЕ. Такая схема, имея на входе a, b и нуль, даст повторение a на одной линии, сумму a и b на другой и перенос в следующий разряд на третьей.

Более сложным устройством является полный сумматор (рис. 5), который складывает a и b с учетом переноса и прибавляет к ним третье значение c; на вход четвертой линии d подается нуль. В целом такая схема состоит из четырех элементов. Помимо полной суммы — результата сложения трех входов линий a, b и c и переноса в следующий разряд — мы на двух других выходах получаем еще два элемента информации. Один — это a, которое было в самом начале, другой — некоторая промежуточная величина, возникшая в процессе вычислений. Это характерное свойство таких обратимых систем: они дают не только то, что требуется, но и определенное количество «мусора». В этом конкретном случае и, как оказывается, во всех других случаях этот мусор может легко быть преобразован во входной сигнал, в данном случае — простым добавлением элемента УПРАВЛЯЕМОЕ НЕ

на выход первых двух линий, как это показано штриховой линией на рис. 5. Мы видим, что мусор превращается в a и b, которые были входными сигналами по крайней мере двух линий. (Вообще говоря, эта схема может быть упрощена, но мы привели ее именно в таком виде с иллюстративными целями.)

Действуя таким образом, мы с помощью различных комбинаций первичных элементов можем создать любой логический блок, который переводит n битов информации в n битов обратимым образом. Если проблема, которую мы пытаемся решить, сама по себе носит обратимый характер, то дополнительного мусора может и не оказаться, но, вообще говоря, нам понадобится

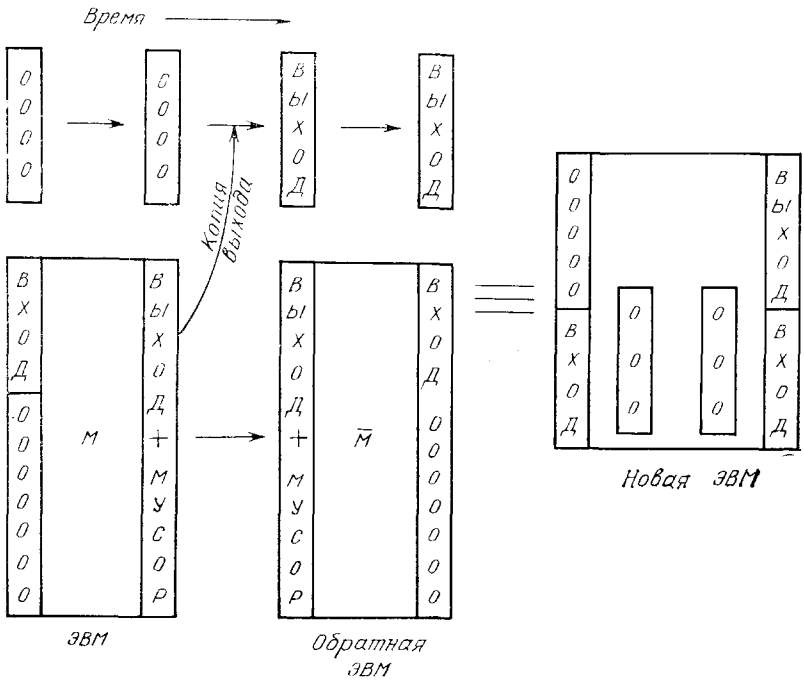


Рис. 6. Очищение от мусора; нули, необходимые для M, могут считаться внутренними в новой полной ЭВМ

несколько дополнительных линий для хранения информации, необходимой для обращения операций. Другими словами, мы можем выполнять любые функции, доступные обычным компьютерам, создавая при этом мусор. Этот мусор содержит информацию, необходимую для обращения операций. Сколько же должно быть этого мусора? В общем случае оказывается, что если входные данные, которые нам необходимы, содержат k битов информации, то, начиная с входной информации и k нулей, мы можем получить на выходе требуемый результат плюс копию входной информации без дополнительного мусора. Эта операция является обратимой, поскольку знание входной и выходной информации, разумеется, позволяет нам все раскрутить обратно. Это утверждение также всегда является обратимым. Доказательство этого иллюстрируется на рис. 6.

Рассмотрим машину M, при подаче на вход которой исходной информации и большого числа нулей мы получим требуемую выходную информацию плюс определенное количество дополнительной информации, которую мы называем мусором. Поскольку мы видели, что возможна операция копирования, состоящая в последовательном применении операторов УПРАВЛЯЕМОЕ НЕ, то, имея дополнительно пустой регистр на k битов информации для записи выхода с машины M, мы можем, после того как процессор M закончит

работу, скопировать результат в этот новый регистр. После этого мы можем построить противоположную машину $\overline{M}$ — обратную M, M наоборот, которая превратит выходную информацию машины M и мусор в исходную информацию и нули. Следовательно, рассматривая все это вместе как одну машину, мы будем иметь вначале k нулей дополнительного регистра и исходную информацию, а в конце — дополнительный регистр, занятый выходной информацией, и повторение исходной информации на выходе. Нулевые линии, которые были необходимы для работы машины M (для приема данных, превращаемых в мусор в процессе работы) в конце опять обнуляются и могут, следовательно, рассматриваться как внутренние линии в новой машине (M, $\overline{M}$ и регистр копирования). Итак, в конце концов мы выполнили то, что собирались сделать, и, следовательно, доказали, что мусор всегда может быть представлен в виде простого повторения исходных данных.

КВАНТОВОМЕХАНИЧЕСКИЙ КОМПЬЮТЕР

Рассмотрим теперь, как такой компьютер может быть построен с использованием законов квантовой механики. Нам надо записать гамильтониан для системы взаимодействующих частей, которые ведут себя как одна большая система, работающая в качестве универсального компьютера. Разумеется, большая система тоже подчиняется законам квантовой механики, однако взаимодействие с тепловым резервуаром и некоторые другие причины могут сделать ее существенно необратимой. Мы хотим создать ЭВМ как можно более простую и как можно меньших размеров. Наш гамильтониан будет подробно описывать все внутренние вычислительные действия, но, разумеется, не взаимодействие с внешними устройствами, связанное с вводом информации (приготовлением начального состояния) и считыванием результатов.

Насколько миниатюрной такая ЭВМ может быть? Насколько малой, например, может быть запись числа? Разумеется, число может быть представлено битами — нулями и единицами. А теперь представим себе, что мы имеем некоторые двухуровневые системы. Назовем такие системы атомами. Тогда число, состоящее из n битов, представится некоторым состоянием «регистра» — набора из n двухуровневых систем. Любое число может быть представлено некоторым набором атомов, каждый из которых находится в состоянии 1 или 0. Число это может быть считано с такого регистра путем определения (измерения) состояний всех атомов. Следовательно, один бит будет представлен единственным атомом, находящимся в одном из двух состояний $|1\rangle$ или $|0\rangle$.

То, что мы должны сделать дальше, легче всего пояснить на примере оператора ДВАЖДЫ УПРАВЛЯЕМОЕ НЕ. Пусть G — некоторая операция над тремя атомами a, b и c, которая переводит исходное состояние a, b, c в новое необходимое нам состояние a', b', c', так что связь между a, b, c и a', b', c' как раз такая, которая существует между входами a, b, c и выходами a', b', c' элемента ДВАЖДЫ УПРАВЛЯЕМОЕ НЕ. Здесь необходимо иметь в виду, что сейчас мы не пытаемся передать данные из одной позиции в другую, мы только собираемся изменить их. В отличие от ситуации в обычной проводниковой ЭВМ, где напряжение на одном проводнике переходит затем в напряжение на другом проводнике, то, что мы сейчас делаем, несколько проще: три атома находятся в некотором конкретном состоянии и выполняется некоторая операция, которая меняет их состояние на новое a', b', c'. В математическом смысле можно сказать, что оператор G, действуя на состояние $|a, b, c\rangle$, дает состояние $|a', b', c'\rangle$. В квантовой механике операторы линейны, так что предположим, что G — линейный оператор. В таком случае его можно описать матрицей G, матричные элементы которой $G_{a,b,c,a',b',c'}$ равны нулю, за исключением тех, которые выписаны в таблице на следующей странице (они равны единице).

Это та самая таблица, которая описывала действие элемента ДВАЖДЫ УПРАВЛЯЕМОЕ НЕ. Очевидно, что действие оператора G обратимо, что может быть записано в виде утверждения $G^+G = 1$, где крест в индексе означает эрмитово сопряжение, т. е. G является унитарной матрицей. (На самом деле наше G представляет собой действительную ($G^* = G$) матрицу, но это лишь частный случай.) Назовем матрицу, которую мы хотели записать для нашего конкретного G, матрицей $A_{ab,c}$. Мы будем использовать матрицы с различным числом индексов для описания других первичных элементов.

a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Например, операция НЕ, которой должно отвечать A_a, представляется матрицей

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Это матрица 2×2, которая может быть записана многими различными способами; мы здесь используем способ записи через операторы рождения и уничтожения. Чтобы не тратить лишних букв, назовем матрицей $\underline{a}$ следующую матрицу:

$$\underline{a} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix},$$

которая уничтожает единицу на атоме a и превращает ее в нуль, т. е. оператор $\underline{a}$ переводит состояние $|1\rangle$ в состояние $|0\rangle$. Однако если атом находится в состоянии $|0\rangle$, то результатом действия оператора $\underline{a}$ будет нуль. Иначе говоря, оператор не меняет состояние, а просто приводит к численному результату, равному нулю, при действии на это состояние. Эрмитово-сопряженный ему оператор

$$\underline{a}^+ = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

является их оператором рождения в том смысле, что при действии на состояние нуль он переводит его в состояние единица, т. е. осуществляет переход от $|0\rangle$ к $|1\rangle$. Если же этот оператор действует на состояние $|1\rangle$ и более высоких, чем $|1\rangle$, состояний нет, то осуществляется переход в состояние $|0\rangle$. Любой оператор, которому отвечает матрица 2×2, может быть выражен через эти два оператора рождения и уничтожения. Например, произведению $\underline{a}^+\underline{a}$ отвечает матрица

$$\underline{a}^+\underline{a} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix},$$

которую можно обозначить N_a. Она дает единицу при действии на состояние $|1\rangle$ и нуль при действии на состояние $|0\rangle$, т. е. дает то число, которому отвечает данное состояние. Аналогично произведение

$$\underline{a}\underline{a}^* = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

представляет собой $1 - N_a$ и дает нуль для верхнего состояния и единицу для нижнего. Для представления единицы мы используем диагональную матрицу

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Из сказанного следует, что $\underline{a}\underline{a}^+ + \underline{a}^+\underline{a} = 1$. Теперь очевидно, что матрица оператора, выполняющего операцию НЕ, запишется в виде $A_a = \underline{a} + \underline{a}^+$. Эта матрица обратима и унитарна ($A_a^+ A_a = 1$).

Точно таким же способом можно построить матрицу $A_{a,b}$ для операции УПРАВЛЯЕМОЕ НЕ. Если заглянуть в таблицу значений для УПРАВЛЯЕМОГО НЕ (см. рис. 3), то легко можно видеть, что она записана в виде

$$A_{a,b} = \underline{a}^+\underline{a}\,(\underline{b} + \underline{b}^+) + \underline{a}\underline{a}^+.$$

Здесь в первом члене справа произведение $\underline{a}^+\underline{a}$ обеспечивает то условие, что если $a = 1$, то оператор НЕ, записанный в виде $\underline{b} + \underline{b}^+$, приложен к линии b. Второй член $\underline{a}\underline{a}^+$ обеспечивает то условие, что если $a = 0$, то с линией b ничего не должно произойти, т. е. на b должна действовать единичная матрица. Это выражение можно также переписать в виде

$$A_{a,b} = 1 + \underline{a}^+\underline{a}\,(\underline{b} + b^+ - 1),$$

где единица представляет все линии, которые проходят устройство «насквозь»; в случае $a = 1$ мы исправляем это выражение, введя операцию НЕ вместо того, чтобы оставить линию b неизменной.

Матрица для ДВАЖДЫ УПРАВЛЯЕМОГО НЕ запишется в виде

$$A_{ab,c} = 1 + \underline{a}^+\underline{a}\underline{b}^+\underline{b}\,(\underline{c} + \underline{c}^+ - 1),$$

что, возможно, читатель и сам уже сообразил.

Следующий вопрос — какова будет матрица общего логического блока, состоящего из последовательности вышеописанных элементов? В качестве примера рассмотрим полный сумматор, уже описанный ранее (см. рис. 5). Теперь в самом общем случае мы имеем четыре линии a, b, c, d. Вовсе не обязательно, что всегда $d = 0$, поскольку мы хотим описать действие этого блока в самом общем случае (если значение d переходит в единицу, d' переходит в НЕ d'). Это дает нам новые числа a', b', c', d'. Представим себе, что имеется четыре атома, обозначенных a, b, c и d, в состоянии, обозначенном $|\,a, b, c, d\,\rangle$, и что матрица M переводит эти четыре атома в состояние $|\,a', b', c', d'\,\rangle$, которое характерно для этого логического блока. Иначе говоря, если $|\,\psi_{\text{in}}\,\rangle$ представляет входное состояние, содержащее информацию в 4 бита, то M — матрица, которая генерирует выходное состояние $|\,\psi_{\text{out}}\,\rangle = M\,|\,\psi_{\text{in}}\,\rangle$ для этих же четырех битов. Например, если входное состояние есть состояние $|\,1, 0, 1, 0\,\rangle$, то, как мы знаем, выходным должно быть состояние $|\,1, 0, 0, 1\,\rangle$. Первые два выхода, a' и b', должны иметь значения единица и нуль, поскольку первые две линии проходят устройство насквозь, а последние два выхода, c' и d', должны иметь значения нуль и единица, поскольку они представляют собой сумму и перенос для первых трех входов a, b и c при $d = 0$. Теперь матрица сумматора M легко может быть представлена как результат пяти последовательных первичных операций, и, следовательно, она принимает вид матричного произведения пяти последовательных матриц, представляющих эти первичные операции:

$$M = A_{a\,b}A_{b\,c}A_{bc\,d}A_{a,b}A_{ab,d}.$$

Первым действует оператор $A_{ab,d}$, записанный с правого края и представляющий собой операцию ДВАЖДЫ УПРАВЛЯЕМОЕ НЕ, в которой a и b — управляющие линии, а операция НЕ осуществляется на линии d. Глядя на рис. 5, легко видеть, что́ описывают остальные операторы в этой последовательности, например последний (крайний слева в правой части) множитель $A_{a,b}$ отвечает УПРАВЛЯЕМОМУ НЕ с управляющей линией a и операцией НЕ на линии b. Матрица M является унитарной, $M^+M = 1$, так как все входящие в нее сомножители являются унитарными матрицами. Иначе говоря, M описывает обратимую операцию, и M^+ — обратная ей операция.

Наша основная задача ставится следующим образом. Пусть A_1, A_2, A_3, . . ., A_k — последовательность операций, необходимых для функционирования некоторого логического блока с n входами и выходами. Приводящая к тому же результату матрица $2^n \times 2^n$ является произведе-

нием $A_k \ldots A_3A_2A_1$, где каждое A_i представляет собой простую матрицу. Каким образом мы можем физически создать матрицу M, если мы знаем, как создавать более простые элементы?

В общем случае в квантовой механике для системы с гамильтонианом H конечное состояние через время t есть $e^{iHt}\psi_{in}$, где ψ_{in} — исходное состояние (в момент $t = 0$). Представляется весьма нелегкой задача о построении такого гамильтониана, чтобы для данного момента времени t $M = e^{iHt}$, где M — описанное выше произведение некоммутирующих матриц, т. е. о построении, основанном лишь на некоторых простых свойствах этих матриц.

Мы понимаем, однако, что если мы разложим e^{iHt} в ряд (например, в виде $1 + iHt - (H^2t^2/2) - \ldots$) в любой конкретный момент времени, мы обнаружим члены, в которые оператор H входит произвольное неограниченное число раз — один раз, два раза, три раза и т. д., и окончательное состояние определяется суммой всех этих членов. Отсюда следует, что мы можем решить проблему совместного действия операторов A следующим образом. Добавим к n атомам, составляющим наш регистр (n-регистр), еще один дополнительный набор из $k + 1$ атомов, которые мы назовем «узлами программного счетчика». Пусть q_i и q_i^+ — операторы рождения и уничтожения для i-го узла программного счетчика, где i пробегает значения от нуля до k. Хорошим примером здесь может быть электрон, движущийся из одного пустого узла в другой. Если узел i занят электроном, это соответствует состоянию $|1\rangle$, если узел свободен — состоянию $|0\rangle$.

Запишем наш гамильтониан в виде

$$H = \sum_{i=0}^{k} q_{i+1}^+ q_i A_{i+1} + \text{к. с.} =$$

$$= q_1^+q_0A_1 + q_2^+q_1A_2 + q_3^+q_2A_3 + \ldots + q_0^+q_1A_1^+ + q_1^+q_2A_2^+ + q_2^+q_3A_3^+ + \ldots \quad (1)$$

Первое, что следует отметить, это что если все узлы программного счетчика не заняты, т. е. все атомы программы исходно находятся в состоянии $|0\rangle$, то в системе ничего не происходит, ибо каждый член гамильтониана (1) начинается с оператора уничтожения, который, действуя на пустой узел, дает нуль на выходе.

Второе, что следует отметить, это что если только один узел программного счетчика занят (находится в состоянии $|1\rangle$), а прочие узлы свободны (состояние $|0\rangle$), то такое положение сохраняется всегда. На самом деле вообще число узлов программы, находящихся в состоянии $|1\rangle$, является сохраняющейся величиной. Предположим, что при действии нашей ЭВМ либо ни одного узла программы не занято (и тогда ничего не происходит), либо занят только один узел; при нормальных операциях два или более узлов программного счетчика никогда не занимаются одновременно.

Начнем с исходного состояния, когда нулевой узел занят, т. е. находится в состоянии $|1\rangle$, а все остальные узлы пусты, т. е. находятся в состояниях $|0\rangle$. Если позднее, в какой-то момент времени последний, k-й узел окажется в состоянии $|1\rangle$ (и, следовательно, все прочие в состоянии $|0\rangle$), мы утверждаем, что n-регистр был умножен на матрицу $M = A_k \ldots A_3A_2A_1$, как мы того и хотели.

Сейчас я объясню, как все это работает. Положим, что вначале n-регистр находится в каком-то начальном состоянии ψ_{in} и нулевой узел программного счетчика занят. Тогда при однократном действии оператором гамильтониана (1) на это состояние ненулевой вклад дает только первый член $q_1^+q_0A_1$. Оператор q_0 превращает нулевой узел в незанятый узел, в то время как q_1^+ переводит первый узел в занятое состояние. Таким образом, произведение $q_1^+q_0$ просто переводит занятое состояние из позиции 0 в позицию 1. Но оно (произведение) еще умножено на матрицу A_1, которая действует только на атомы n-регистра, и, следовательно, исходное состояние атомов n-регистра умножается еще на A_1. Если мы теперь применим этот гамиль-

тониан второй раз, этот первый член не произведет никакого действия, поскольку оператор q_0 даст нуль на узле 0, так как тот не занят. Зато теперь включается второй член гамильтониана $q_2^\dagger q_1 A_2$, который может переместить занятую точку дальше. Эту занятую точку будем называть «курсор». Курсор может переместиться с узла 1 на узел 2, но теперь на регистр подействует матрица A_2, так что в конечном счете этот регистр подвергается действию матрицы $A_2 A_1$. Итак, глядя на первую строчку гамильтониана (левая часть второй строки (1)), видим, что если бы праой части вообще не было, то при последовательном действии гамильтониана курсор перемещался бы последовательно от узла 0 до узла k программного счетчика, а на атомы n-регистра последовательно действовали бы одна за другой матрицы A в том самом порядке, который требуется для построения полного M.

Однако гамильтониан должен быть эрмитовым, следовательно, в нем должны присутствовать комплексные сопряженные всех операторов. Допустим, что на определенном этапе мы получили курсор на узле 2 и имеем матрицу $A_2 A_1$, действующую на регистр. Теперь оператор q_2, который собирается перевести это занятое состояние в новую позицию, не обязательно должен быть оператором первой половины гамильтониана (1), он может быть и оператором второй половины выражения (1). Это может быть оператор из члена $q_1^\dagger q_2 A_2^\dagger$, который переводит курсор из позиции 2 обратно в позицию 1. Но заметим, что, когда это происходит, на регистр действует оператор $A_2^\dagger$ и, следовательно, полный оператор, действующий на регистр в этом случае, есть $A_2^\dagger A_2 A_1$. Но $A_2^\dagger A_2 = 1$, так что это просто оператор A_1. Итак, если курсор вернулся в позицию 1, то в итоге на регистр действует только оператор A_1. Значит, по мере того как различные члены гамильтониана перемещают курсор вперед и назад, операторы A либо накапливаются, либо снова взаимно сокращаются. На любой стадии, например, если курсор продвигается до узла j, матрицы от A_1 до A_j последовательно действуют на атомы n-регистра. Не имеет значения, по какому пути курсор попал на узел j — прямо от нулевого узла, или пройдя узел j и вернувшись, или же двигаясь взад-вперед по какому-либо произвольному закону, лишь бы он в конце концов оказался на состоянии j. Следовательно, справедливо утверждение, что если курсор обнаружен на узле k, то общий результат для атомов n-регистра таков, как если бы матрица M действовала на исходные состояния атомов, как мы того и хотели.

Как же работать с такой ЭВМ? Начнем с подачи входных битов на регистр и с установки курсора на нулевой узел программы. Затем мы проверяем, скажем, с помощью рассеяния электронов, узел k — пуст или на нем есть курсор. В момент, когда на узле k обнаружится курсор, мы уберем его, так, чтобы он не смог вернуться обратно по строке программы, и тогда мы будем знать, что регистр содержит выходные данные. Теперь можем снять их, когда пожелаем. Разумеется, имеются еще внешние устройства, связанные с выполнением этих измерений и определением всего прочего, которые не являются частью нашего компьютера. Само собой, компьютер всегда будет находиться во взаимодействии с внешним миром, как при вводе, так и при выводе данных.

Математически оказывается, что перемещение курсора вперед и назад по строке программы будет происходить в точности так, как если бы операторы A вообще не входили бы в гамильтониан. Другими словами, движение курсора представляет собой просто волны, знакомые нам из теории распространения волн в одномерной системе жестко связанных электронов или одномерных спиновых волн, также хорошо нам известных. Волны могут распространяться вперед и назад по строке, образовывать волновые пакеты и т. п. Мы можем улучшить работу нашего компьютера, переведя его в баллистический режим следующим образом. Расширим дополнительно программный счетчик, т. е. добавим к тем узлам, которые мы реально используем для вычислений, новые (и слева, и справа), и пусть число этих узлов

будет достаточно велико. Иначе говоря, разрешим индексу i для q_i иметь значения, которые меньше нуля или больше k, причем соответствующие им матрицы A тождественно равны единице. Тогда мы получили бы более длинную цепочку спинов и могли бы начать, вместо помещения курсора точно на начальном узле 0, с помещения его с заданными вероятностями на различных узлах так, чтобы получилась входная спиновая волна в виде широкого волнового пакета с почти определенным моментом импульса. Эта спиновая волна затем пройдет сквозь весь компьютер и выйдет через другой его конец в тот «хвост», который мы добавили к программному счетчику, а там уже будет легче определить, пришла она или нет, направить ее куда-нибудь еще и перехватить курсор. Следовательно, эта логическая схема может функционировать баллистическим образом.

Это очень важный момент, указывающий, по крайней мере, специалистам по ЭВМ, что мы можем создать универсальную ЭВМ, так как они знают, что если мы можем реализовать любую логическую схему, то мы можем создать и универсальный компьютер. То, что таким образом можно создать универсальную ЭВМ с условными переходами и сложными операциями, не вполне очевидно неспециалисту, и я собираюсь еще вернуться к этому вопросу.

НЕИДЕАЛЬНОСТЬ И НЕОБРАТИМЫЕ ПОТЕРИ СВОБОДНОЙ ЭНЕРГИИ

Теперь надо обсудить вопрос о неидеальности нашей ЭВМ. В описанной ЭВМ имеется много источников неидеальности, но прежде всего мне хотелось бы рассмотреть возможность того, что параметры взаимодействия между элементами в различных местах программы не являются в точности одинаковыми. Программа может быть столь длинна, что при реальных вычислениях незначительные нерегулярности дадут небольшую вероятность рассеяния, и волны не будут распространяться строго баллистически, а будут перемещаться в обоих направлениях. Если система, например, устроена так, что эти узлы расположены на подложке из обычных физических атомов, то тепловые колебания этих атомов будут незначительно изменять связи и создавать неидеальность. (Подобный тепловой шум может даже быть полезен, так как в присутствии слабых статических дефектов появляются неглубокие локализованные состояния, в которых может быть захвачен курсор.) Предположим, что на каждом шаге вычислений (т. е. на каждом шаге движения курсора, $i \to i + 1$) имеется определенная вероятность рассеяния импульса курсора, скажем, p, так что в конце концов его движение хаотизируется (при этом $1/p$ — средняя длина свободного пробега курсора). Мы будем считать, что p достаточно мало. Тогда при очень длинных вычислениях потребуется очень много времени, чтобы волна могла пройти весь компьютер до конца, из-за многократного рассеяния. Однако в этом случае можно будет перемещать курсор вдоль программы с помощью внешней силы. Если курсором является, например, электрон, перемещающийся от одного свободного узла к другому, это будет что-то вроде электрического поля, перемещающего электрон вдоль проводника, сопротивление которого является конечным из-за наличия дефектов (т. е. из-за конечной вероятности рассеяния). При этом мы можем рассчитать, сколько энергии будет затрачено этой внешней силой.

Этот анализ может быть выполнен очень легко, это почти классический анализ движения электрона с некоторой средней длиной свободного пробега. Каждый раз, когда курсор рассеивается, мы будем полагать, что его рассеяние вперед или назад совершенно случайно. Для того чтобы машина могла функционировать, вероятность рассеяния вперед должна быть больше вероятности рассеяния назад. Следовательно, когда происходит рассеяние, потеря энтропии равна логарифму вероятности движения курсора вперед, деленной на вероятность движения курсора назад, или приблизительно —

отношению разности вероятностей движения вперед и назад к их сумме. Это есть потеря энтропии за один акт рассеяния. Более интересным является выражение для потери энтропии на одном шаге вычислений, которая, очевидно, в p раз больше. Мы можем записать ее в виде

$$W = \frac{pv_д}{v_г},$$

где $v_д$ — скорость направленного движения (дрейфа) курсора, а $v_г$ — скорость его случайного движения. Или, если хотите, это есть взятое p раз минимальное время, за которое вычисления могут быть выполнены (т. е. время в случае, когда все шаги выполняются только вперед), деленное на реальное возможное время. Тогда потери свободной энергии представляют собой величину pkT, умноженную на минимальное время вычислений и деленную на реальное время, которое мы выделяем на эти вычисления. Это выражение впервые было получено Беннетом. Фактор p здесь можно назвать фактором замедления, поскольку он описывает ситуацию, в которой данный узел рассеивает курсор не всегда, а лишь случайным образом с малой вероятностью p. Следует обратить внимание на тот важный факт, что потеря энергии на один шаг не равна kT; эта величина должна быть поделена на два фактора. Один $(1/p)$ является мерой того, насколько совершенной мы можем создать нашу машину, другой пропорционален времени, затраченному на выполнение вычислений. Это похоже на машину Карно, в которой для достижения обратимости следует действовать очень медленно. Для идеальной машины, у которой $p = 0$ или в которой допустимо бесконечно большое время вычислений, средние потери энергии равны нулю.

Надо отметить, что принцип неопределенности, который обычно связывает неопределенности энергии и времени, к прямым ограничениям здесь не приводит. Наш компьютер есть устройство для произведения вычислений, но время прибытия курсора на другой конец ЭВМ и время выполнения измерения данных выходного регистра (другими словами, полное время, которое необходимо для завершения вычислений) не является определенным временем. Это случайная величина, так что имеется известная неопределенность времени, за которое вычисления могут быть выполнены. Потерь, связанных с неопределенностью энергии курсора, нет, по крайней мере нет потерь, зависящих от числа шагов вычислений. Разумеется, если мы хотим производить баллистические вычисления на совершенной машине, некоторую энергию необходимо вложить в исходную волну, но эта энергия, конечно, может быть извлечена из конечной волны, когда эта последняя выйдет с другого конца машины. Все вопросы, связанные с неопределенностью операторов и необратимостью измерений, связаны с входными и выходными функциями. По существу, никаких дополнительных ограничений, обусловленных квантовой природой самого компьютера, нет; нет и ограничений, которые зависели бы от числа шагов программы.

В подобной машине имеется множество других проблем, обусловленных неидеальностью. Например, в регистрах для хранения данных возникает проблема перекрестных «наводок», т. е. взаимодействия между различными атомами такого регистра, или проблема взаимодействия атомов регистра непосредственно с какими-то процессами, происходящими в программе, о которых мы можем и не подозревать. Другими словами, в гамильтониане могут быть малые члены помимо тех, что записаны. До тех пор, пока мы не конкретизируем конструкцию придуманной нами ЭВМ, эти проблемы очень трудно анализировать. По крайней мере некоторые из этих проблем могут быть разрешены применением обычных методов, таких, как корректирующие коды и т. п., хорошо известных в теории обычных компьютеров. Но до тех пор, пока мы не определили конкретную конструкцию нашей ЭВМ, мы не будем знать, как анализировать такие эффекты. На практике, однако, они могут оказаться чрезвычайно существенными. Судя по всему, наш компьютер должен быть

крайне деликатным устройством, и такая неидеальность его может внести весьма существенный беспорядок в работу.

Время, необходимое для выполнения вычислительного шага, зависит от силы (энергии) взаимодействий, входящих в гамильтониан. Если предположить, что все члены гамильтониана имеют порядок 0,1 эВ, то получится, что время для одного шага курсора, выполненного в баллистическом режиме, будет порядка $6 \cdot 10^{-15}$ с. Это не столь уж гигантское улучшение; возможно, всего лишь порядка на четыре лучше, чем время переключения в современных транзисторах, и ненамного меньше, чем те весьма короткие промежутки времени, которые, в принципе, могут быть получены в различных оптических системах.

БОЛЕЕ ПРОСТАЯ РЕАЛИЗАЦИЯ

Итак, мы выполнили поставленную перед собой задачу — нашли квантовомеханический гамильтониан системы, которая способна производить вычисления, но и только. Теперь представляет интерес рассмотреть вопросы, связанные с упрощением реализации такой системы. Записанный нами гамильтониан содержит члены, которые могут описывать определенное взаимодействие пяти атомов. Например, три из этих атомов могут составлять регистр для блока ДВАЖДЫ УПРАВЛЯЕМОЕ НЕ, а два оставшихся могут играть роль соответствующих соседних узлов программного счетчика. Организовать это довольно трудно. Вопрос: можно ли это сделать с помощью более простых элементов? Оказывается, можно. Можно сделать так, что в каждом взаимодействии будут участвовать только три атома. Мы собираемся начать с новых первичных элементов вместо тех, с которых начинали ранее. Разумеется, мы используем оператор НЕ, но в дополнение к нему используем простой оператор ПЕРЕКЛЮЧАТЕЛЬ (см. также [5]).

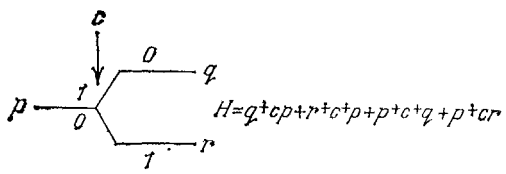


Рис. 7. ПЕРЕКЛЮЧАТЕЛЬ.

Если $c = 1$, идти от p к q и положить $c = 0$; если $c = 0$, идти от p к r и положить $c = 1$; если $c = 1$, идти от r к p и положить $c = 0$; если $c = 0$, идти от q к p и положить $c = 1$

Допустим, что в гамильтониане есть член $q^+cp + r^+c^+p$ + к. с. (во всех случаях ниже будем использовать буквы из первой половины алфавита для обозначения атомов регистра и из второй половины алфавита для обозначения узлов программы). Схема оператора ПЕРЕКЛЮЧАТЕЛЬ приведена на рис. 7. Это переключатель в том смысле, что если изначально c находится в состоянии $|1\rangle$, то курсор с линии p переходит на линию q, в то время как если c находится в состоянии $|0\rangle$, то курсор переходит с p на r. В течение операции управляющий атом c меняет свое состояние. (Можно также записать выражение, при котором управляющий атом не меняет своего состояния, например $q^+c^+cp + r^+cc^+p$ + к. с., но в этом нет ни особых достоинств, ни особых недостатков, и мы выберем наиболее простое выражение.) Комплексное сопряжение обеспечивает обратимость этого выражения. Если, однако, курсор находится на q, а c находится в состоянии $|1\rangle$ (или курсор на r, а c — в состоянии $|0\rangle$), то гамильтониан H дает нуль и курсор «отражается» обратно. Построим все наши цепи и выберем все начальные состояния так, чтобы такое положение не возникало при нормальном функционировании, в результате чего и реализуется идеальный баллистический режим.

С помощью этого переключателя можно выполнять большое количество операций. Например, можно получить оператор УПРАВЛЯЕМОЕ НЕ, как это показано на рис. 8. Переключатель a управляет операцией 0 (т. е. дает на выходе нуль, если $a = 1$). Пусть вначале курсор находится на линии s. Если $a = 1$, то курсор переносится по верхней линии на рис. 8, в то время

как если $a = 0$, он переносится по нижней линии, в любом из этих случаев оказываясь в конце на программном узле t. На этих диаграммах горизонтальные и вертикальные линии представляют атомы программы. Переключатели представлены наклонными линиями, а в прямоугольники (блоки) заключены другие матрицы, действующие на регистры, такие, как НЕ b. Для примера, гамильтониан этой небольшой части оператора УПРАВЛЯЕМОЕ НЕ (начальное положение s, конечное — t) имеет вид

$$H_c(s, t) = s_M^+ as + t^+ a^+ t_M + t_M^+ (b + b^+) s_M + s_N^+ a^+ s + t^+ a t_N + t_N^+ s_N + \text{к. с.}$$

Хотя на первый взгляд кажется, что здесь имеются два возможных пути, что может привести к усложнениям, характерным для квантовой механики, на самом деле это не так. Если в начале расчетов атом a находился в определенном состоянии, то к моменту, когда курсор достигнет атома s, атом a все еще будет находиться в некотором определенном состоянии (хотя, возможно, отличном от начального вследствие предыдущих вычислительных операций на нем). Следовательно, выбирается только один из двух путей.

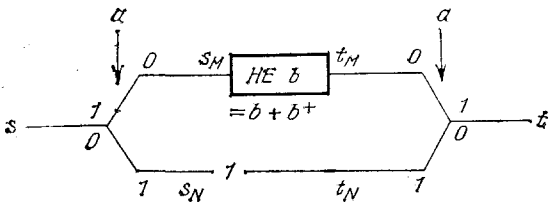


Рис. 8. Оператор УПРАВЛЯЕМОЕ НЕ, построенный из переключателей

Приведенное выше выражение можно упростить, если опустить член $s_N^+ t_N$ и положить $t_N = s_N$. В этом случае нет нужды беспокоиться о том, что один путь длиннее (курсор проходит два узла), чем другой (курсор проходит один узел), поскольку интерференция по-прежнему отсутствует. В любом случае не возникает никакого рассеяния, если в цепь связанных узлов внедряется дополнительный кусок цепи, состоящий из произвольного числа узлов с теми же самыми взаимными связями между узлами (аналогично согласованию импедансов передающих длинных линий).

Рис. 9. Блок M. s_M — начальный узел программы, t_M — конечный узел программы

Далее, рассмотрим самую общую комбинацию различных элементов. Элемент M (рис. 9) может быть представлен в виде логического блока взаимодействующих частей, из которых мы изобразим только первый входной узел курсора s_M и последний на другом конце узел t_M. Все остальные узлы программы, находящиеся между s_M и t_M, рассматриваются как внутренние части M; M также содержит все необходимые регистры. Только s_M и t_M являются узлами, которые могут иметь внешние связи. Гамильтониан этой подсистемы обозначим через H_M, а названия входного и выходного узлов программы идентифицируются записью $H_M(s_M, t_M)$. Так что, следовательно, H_M есть часть гамильтониана, представляющая все атомы в прямоугольнике и их внешние начальные и конечные узлы.

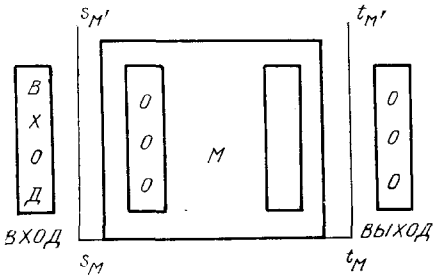
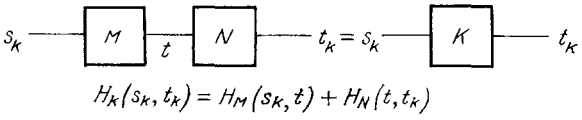


Рис. 10. Блок с внешними входом и выходом

Рассмотрим особенно важный и интересный случай, когда входные данные (находящиеся на атомах регистра) приходят с одного логического блока и нужно передать их другому (рис. 10). Допустим, что в блоке M на входном регистре находятся нули и на выходном (который может быть тем же самым регистром) — также нули. Тогда можно использовать это следующим образом. Можно просто построить такую программу, скажем, начинаю-

щуюся с $s_{\overline{M}'}$, которая будет обменивать данные, находящиеся на внешнем регистре, содержащем входную информацию, со входным регистром блока M, в настоящее время содержащим нули. Тогда первым шагом в наших вычислениях, начинающихся, скажем, с $s_{M'}$, будет обмен с регистром внутри M. Это переводит нули в тот регистр, в котором была входная информация, и переводит входную информацию внутрь M, туда, куда следует. Курсор теперь находится на s_M. (Мы уже объяснили, как обмен может быть выполнен операторами УПРАВЛЯЕМОЕ НЕ.) Затем, после того как программа переходит от s_M к t_M, получаем результат внутри блока M. Затем выходной регистр M очищается, так как мы записываем результат в некоторый новый внешний регистр, специально для этого предназначенный и исходно содержащий нули. Это мы делаем, переходя от t_M до $t_{M'}$ при помощи обмена данных между пустым внешним регистром и выходным регистром M.



$$H_K(s_K, t_K) = H_M(s_K, t) + H_N(t, t_K)$$

Рис. 11. Последовательные операции

Теперь мы можем рассмотреть различные способы связи таких блоков. Например, наиболее очевидным способом является последовательное соединение. Если мы хотим сначала выполнить M, а потом N, можно связать конечную часть одного блока с входной частью другого (рис. 11). Это даст нам новый эффективный оператор K с гамильтонианом

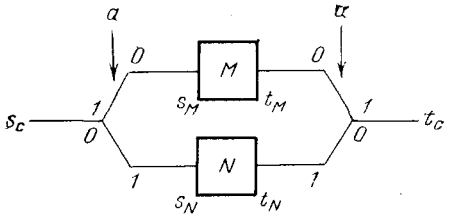


Рис. 12. Выполнение условной операции: если $a = 1$, то M, иначе N

$$H_K(s_K\ ,\ t_K) = H_M(s_K\ ,\ t)\ +\ H_N(t,\ t_K).$$

Условный переход в самом общем виде («Если $a = 1$, выполнить M, а если $a = 0$, выполнить N») может быть осуществлен согласно рис. 12. Для него

$$H_{\text{усл}}(s_c,\ t_c) = (s_M^+ a s_c + t_c^+ a^+ t_M + s_N^+ a^+ s_c + \\ + t_c^+ a t_N + \text{к.с.}) + H_M(s_M,\ t_M) + H_N(s_N,\ t_N).$$

Частным случаем этого является оператор УПРАВЛЯЕМОЕ НЕ, для которого M = НЕ b, гамильтониан

$$H_{\text{НЕ } b}(s,\ t) = s^+ (b + b^+)\ t + \text{к. с.},$$

а N — пустой оператор $s^+ t$.

В качестве другого примера рассмотрим «очиститель от мусора» (ранее показанный на рис. 6), но не содержащий две машины — прямую и обратную,— а использующий одну и ту же машину, посылая данные обратно в нее в противоположном направлении с использованием вышеописанного переключателя (рис. 13). Допустим, что в этой системе имеется специальный «флаг» f, который изначально всегда установлен на нуль. Предположим также, что мы имеем входные данные во внешнем регистре, пустой внешний регистр, способный принять выходные данные, и что все машинные регистры пусты (содержат нули). Начнем с s. Сначала скопируем (используя оператор УПРАВЛЯЕМОЕ НЕ) наш внешний вход в M. Затем действует M, и курсор переходит на верхнюю линию рис. 13. Выход M копируется во внешний выходной регистр. Теперь M содержит мусор. Затем f меняется на НЕ f, подводится к нижней линии переключателя, проходит через M в обратном направлении, уничтожая мусор, и снова освобождает вход. Когда

мы копируем данные и повторяем это копирование снова, мы заполняем один из регистров нулями — тот регистр, на который мы копировали их первый раз. После копирования данные выходят (поскольку f теперь изменилось) на другую линию, где f снова обнуляется, и выходят на линию t.

Итак, между s и t имеем новую машину, обладающую следующими свойствами. Когда она начинает работу, в регистре, названном ВХОД, имеются

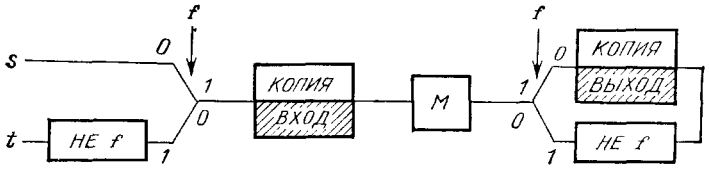


Рис. 13. Очиститель от мусора

входные данные. Во внешнем регистре, названном ВЫХОД, мы имеем нули. Имеются также внутренний флаг, поставленный на нуль, и блок M, свободный от всех данных. По окончании работы, т. е. после перехода от s к t, входной регистр по-прежнему содержит входные данные, а выходной регистр содержит результат действия оператора M. Сам блок M, однако, по-прежнему пуст и флаг f снова установлен на нуль.

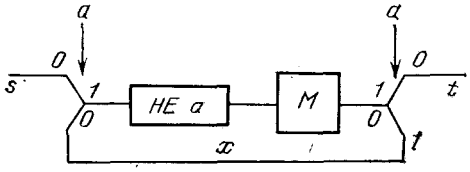


Рис. 14. Выполнение операции M дважды

В компьютерных программах также весьма важно иметь возможность использовать одну и ту же подпрограмму несколько раз. Разумеется, с логической точки зрения это можно сделать, записывая нужную часть программы снова и снова каждый раз, когда в этом возникнет необходимость, но на практике гораздо лучше, если бы мы могли однажды создать часть компьютера, выполняющую некоторую операцию, и затем снова и снова использовать ее. Для того чтобы продемонстрировать это, сначала просто предположим, что некоторую операцию мы хотим выполнить дважды последовательно (рис. 14). Начнем с линии s, установив флаг a на нуль, и, следовательно, пройдем вдоль верхней линии рис. 14. и первое, что произойдет,— это сменится значение a. Затем выполнится оператор M. Поскольку теперь a имеет другое значение, то вместо того, чтобы выйти на верхнюю линию второго переключателя, аналогичную той линии, через которую вошли, мы попадем на нижнюю линию, которая возвратит нас обратно ко входу подпрограммы, после чего подпрограмма повторится и a вновь изменится. Теперь, после выхода из M мы через второй переключатель попадем на его верхнюю линию и, следовательно, в точку выхода t. Гамильтониан этого процесса имеет вид

$$H_{MM}(s,\ t) = s_N^{\dagger} a^{+} s + s_M^{\dagger}(a^{+} + a)\, s_N + \\ + x^{+} a^{+} t_M + s_N^{\dagger} a x + t^{+} a t_M + \text{к. с.} + H_M(s_M,\ t_M).$$

Используя такую переключающую цепь многократно, мы можем многократно повторять необходимую операцию. Например, повторяя то же самое последовательно три раза (гнездовая, или вложенная последовательность операторов), можно выполнить требуемую операцию восемь (2^3) раз с помощью устройства, изображенного на рис. 15. Для этого нам понадобятся три флага. При повторном выполнении операций необходимость иметь флаги обусловлена тем, что нужно следить, сколько раз операция выполнена и в каком месте программы мы находимся, иначе мы никогда не добьемся обратимости. В нормальном компьютере подпрограмма может быть многократно использована повторно без какой-либо регистрации того, что было сделано.

Но в нашей ЭВМ мы должны вести регистрацию, и делаем это с помощью флагов, отмечая, в который раз вызвана подпрограмма. Если подпрограмма вызывается из определенного места, а выход должен быть сделан в какое-то другое место и потом она вызывается еще раз, то, поскольку ее адреса возврата различны, необходимо следить за этим и знать, откуда она была вызвана

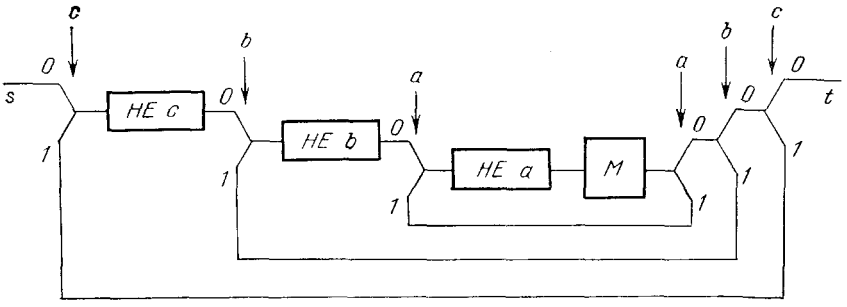


Рис. 15. Выполнение операции M восемь раз

и куда должно быть возвращено управление в каждом отдельном случае, так что требуется хранение еще большего количества данных. Многократное использование подпрограммы в обратимой машине лишь немногим сложнее, чем в обычном компьютере. Все вышеизложенное рассматривалось в работах [1-4].

Ясно, что использование таких переключателей, в частности, последовательное их соединение в структуру типа дерева (разветвляющиеся структуры), позволит нам направить данные в любую точку памяти. Память

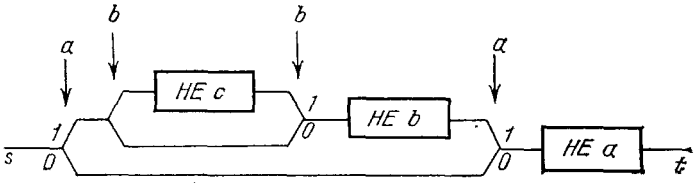


Рис. 16. Дискретный трехбитовый счетчик

тогда будет просто местом, где находятся регистры, на которые мы можем скопировать данные, и затем вернуться к программе. Курсор обязан следовать за данными, так что необходимо предположить, что существует другое дерево переключателей, установленное в противоположном направлении, чтобы обеспечить возврат курсора после копирования данных, так, чтобы система осталась обратимой.

На рис. 16 показан дискретный двоичный счетчик (на три бита a, b, c, где c — старший бит), который следит за тем, сколько в конечном счете раз курсор пройдет от s до t. Эти несколько примеров достаточно ясно показывают, что действительно мы можем построить все вычислительные функции с помощью операторов ПЕРЕКЛЮЧАТЕЛЬ и НЕ. В подробности я вдаваться не буду.

ЗАКЛЮЧЕНИЕ

Из приведенных примеров ясно, что квантовомеханический компьютер на самом деле не использует многие специфические свойства дифференциальных уравнений квантовой механики.

Мы только попытались имитировать, насколько это возможно, цифровую машину с обычной последовательной архитектурой. Это аналогично использованию транзисторов в обычной ЭВМ, где мы фактически не используем весь

аналоговый континуум свойств транзисторов, а лишь пытаемся работать с ними как с цифровыми устройствами с двумя состояниями, используя их только в режиме насыщения, так что логический анализ поведения таких систем значительно проще. Далее, наша система является абсолютно последовательной, например даже при сравнении (ИСКЛЮЧАЮЩЕЕ ИЛИ) двух чисел, содержащих k битов, нужно обрабатывать каждый бит последовательно. В обратимых квантовых системах операции могут выполняться параллельно с соответствующим выигрышем в скорости вычислений, но здесь это не рассматривалось.

По причинам, представляющим теоретический и академический интерес, я рассматривал полные и обратимые системы, но если такие крошечные машины окажутся полезными, нет никаких причин, по которым необратимые, увеличивающие энтропию взаимодействия, не могут быть использованы при построении такой машины. Например, возможно, что в длительных вычислениях будет разумным для большей уверенности запретить курсору возвращаться назад после достижения определенной точки. Или может оказаться удобным связать необратимую память (для редко используемых данных) с необратимой логикой или кратковременными обратимыми регистрами памяти и т. д. Опять же нет никаких причин, по которым мы должны использовать цепь взаимодействующих узлов для далеких связей, где проводная или световая связь может оказаться проще и быстрее.

Во всяком случае, очень похоже, что законы физики не ставят предела миниатюризации компьютеров до тех пор, пока плотность записи не превышает 1 бит/атом и справедливы законы квантовой механики.

Автор благодарен Т. Тоффоли за помощь в составлении списка литературы.

СПИСОК ЛИТЕРАТУРЫ

1. Bennett C.H.//IBM J. Res. a. Devel. **1979**. V. 6. P. 525.
2. Fredkin E., Toffoli T.//Intern. J. Theor. Phys. **1982**. V. 21. P. 219.
3. Bennett C. H.//Intern. J. Theor. Phys. **1982**. V. 21. P. 905.
4. Toffoli T.//Math. Syst. Theory. **1981**. V. 14. P. 13.
5. Priese L.//J. Cybernetics. **1976**. V. 6. P. 101.