

## Глава 4. Форматы команд и данных

*Компьютеры – это не только усилители человеческого разума. В той же мере они усиливают и человеческую глупость.*

*Ю.И. Манин*

В этой главе вводятся основные определения и рассматриваются важные базовые понятия, которые будут необходимы для дальнейшего изучения архитектуры ЭВМ. С этой целью будут построены ещё несколько различных учебных машин, они как бы кратко повторяют историю начального этапа развития вычислительной техники.

### 4.1. Адресность ЭВМ

*Программировать – значит понимать.*

*Кристин Нюгард*

Как уже упоминалось, число адресов в машинной команде называется *адресностью* ЭВМ. Разнообразие архитектур ЭВМ предполагает, в частности, и разную адресность их систем команд. Рассмотрим схему выполнения команд на компьютерах с различным числом адресов операндов. По сравнению с изученной ранее ЭВМ УМ-3 несколько увеличим аппаратные возможности (объём памяти и число возможных кодов операций), чтобы приблизить эти параметры к характеристикам "настоящих" машин второго поколения.

Будем предполагать, что для хранения кода операции в команде учебной машины отводится один байт (8 бит, что достаточно для представления 256 различных кодов операций), а для хранения каждого из адресов – по 3 байта, что обеспечивает доступ к объёму адресуемой памяти в  $2^{24}$  (примерно 16 миллионов) байт. Ниже приведены форматы команд для ЭВМ различной адресности и схемы выполнения этих команд для случая бинарных (двуместных) операций (у таких операций два операнда и один результат, это, например, привычные для нас арифметические операции сложения, вычитания, умножения и деления). Как и ранее, для обобщённого обозначения любой бинарной операции будет использоваться символ  $\otimes$ .

#### • Трёхадресная машина

Трёхадресные команды являются наиболее естественными для человека при составлении программы на машинном языке, так как большинство элементарных шагов в вычислительных алгоритмах являются именно бинарными операциями (например, операции сложения, вычитания, умножения и т.д.). Пусть команда в новой трёхадресной учебной машине имеет такой формат:

КОП	A1	A2	A3
8 разрядов	24 разряда	24 разряда	24 разряда

 = 10 байт

Это модификация уже знакомой Вам трёхадресной ЭВМ УМ-3, однако, теперь длина каждой команды увеличена до 10 байт. Длина целых и вещественных чисел по-прежнему будет 32 бита или 4 байта. Итак, здесь команды и числа имеют разную длину, что типично для современных ЭВМ. Так как теперь команды и числа имеют разную длину, то предполагается, что память состоит из коротких ячеек длиной в один байт, при этом *адресом* команды или числа будет адрес её первой ячейки (байта).

Схема выполнения команд такой машины её процессором нам уже известна, она такая же, как и на старой учебной ЭВМ УМ-3:<sup>1</sup>

$R1 := \langle A2 \rangle; R2 := \langle A3 \rangle; S := R1 \otimes R2;$   
 $\langle A1 \rangle := S; \{ \otimes - \text{любая бинарная операция} \}$

В сокращённом виде эту схему можно записать как  $\langle A1 \rangle := \langle A2 \rangle \otimes \langle A3 \rangle$ .

#### • Двухадресная машина

Сократим теперь число адресов в машинной команде с трёх до двух и мысленно построим двухадресную учебную машину (по аналогии с предыдущей учебной машиной назовём её УМ-2). Длина каждой команды для двухадресной ЭВМ будет равна 7 байт:

<sup>1</sup> В схемах выполнения команд разной адресности пока не будет учитываться, что некоторые команды могут также вырабатывать признак результата (для УМ-3 этот признак помещался в регистр  $\omega$ ).

<b>КОП</b>	<b>A1</b>	<b>A2</b>
8 разрядов	24 разряда	24 разряда

= 7 байт

Схема выполнения бинарных двухадресных команд в процессоре нашей машины УМ-2 будет такой:

$R1 := \langle A1 \rangle; R2 := \langle A2 \rangle; S := R1 \otimes R2; \langle A1 \rangle := S$

Заметим, что теперь, так как адресов у нас только два, для выполнения бинарной операции первый и второй операнды задаются в команде *явно* в качестве адресов, а местоположение результата задаётся *неявно* или, как говорят, *по умолчанию*. В рассмотренном выше случае двухадресных команд результат бинарной операции по умолчанию помещается на место первого операнда, уничтожая его старое значение (заметим, что в большинстве современных машин, в том числе и в наиболее распространённых персональных ЭВМ, двухадресные команды выполняются именно по такой схеме). В сокращённом виде эту схему можно записать как  $\langle A1 \rangle := \langle A1 \rangle \otimes \langle A2 \rangle$ .

- **Одноадресная машина**

При дальнейшем сокращении числа адресов в машинной команде получится уже одноадресная учебная машина УМ-1, команды которой имеют такую структуру:

<b>КОП</b>	<b>A1</b>
8 разрядов	24 разряда

= 4 байта

Длина каждой команды этой машины равна 4 байта. Схема выполнения одноадресных команд будет такой:

$R2 := \langle A1 \rangle; S := S \otimes R2$

Как видим, при выполнении бинарных операций в одноадресной ЭВМ уже только один *второй* операнд задаётся в команде явно, а первый операнд и результат операции задаются неявно – это регистр сумматора  $S$  в арифметико-логическом устройстве. В сокращённом виде эту схему можно записать как  $S := S \otimes \langle A1 \rangle$ .

Для работы в одноадресной машине необходимы ещё две специальные команды, которые имеют один операнд и один результат. Эти команды реализуют унарные (одноместные) операции. У нас это команда чтения операнда из памяти на регистр сумматора:

СЧ A1

Она выполняется по схеме

$S := \langle A1 \rangle$

и команда записи значения из сумматора в память:

ЗП A1

Она выполняется по схеме

$\langle A1 \rangle := S$

Заметим, что в некотором смысле одноадресная архитектура самая простая для реализации. Например, здесь уже не нужен регистр  $R1$  для первого операнда и каждая команда обращается в память только один раз. Поэтому не случайно многие первые ЭВМ, построенные в соответствии с принципами фон Неймана, были именно одноадресными машинами. Например, это машина EDSAC, построенная в 1949 году при участии А. Тьюринга.

- **Безадресная машина**

Логическим завершением процесса уменьшения числа адресов в машинной команде является построение нулядресной (или безадресной) учебной ЭВМ, назовём её УМ-0. Почти все команды этой машины состоят только из кода операции и имеют длину один байт:

<b>КОП</b>
8 разрядов

= 1 байт

В отличие от других рассмотренных выше учебных машин, новая безадресная машина кроме основной памяти, в которой, как обычно, хранится программа и данные, использует при своей работе также аппаратно реализованный в компьютере *стек* для хранения чисел (операндов). Для обмена данными между основной памятью и стеком в язык машины вводятся две дополнительные *одноадресные* команды длиной по 4 байта. Это команда записи (добавления) машинного слова в стек из любой ячейки памяти с адресом  $A1$

ВСТЕК A1

которая выполняется по схеме

$R1 := \langle A1 \rangle; \text{ВСТЕК}(R1)$

и команда чтения машинного слова из вершины стека в ячейку основной памяти (как обычно, при чтении машинное слово удаляется из стека)

$\text{ИЗСТЕКА } A1$

которая выполняется по схеме

$R1 := \text{ИЗСТЕКА}; \langle A1 \rangle := R1$

Таким образом, в этой архитектуре команды языка машины имеют разную длину. Вообще говоря, кроме указанных выше одноадресных команд записи в стек и чтения из стека, для удобства программирования в безадресной ЭВМ могут добавляться и некоторые другие одноадресные команды (например, команды безусловного и условного переходов, команда вызова процедуры и другие). В отличие от команд записи из памяти в стек и чтения из стека в память, остальные одноадресные команды являются *избыточными*. Так, например, одноадресную команду перехода на ячейку с адресом  $A1$  можно заменить двумя последовательными командами: одноадресной командой записи в стек значения адреса перехода  $A1$  (в виде целого числа) и безадресной командой перехода по адресу, записанному в вершине стека.

Таким образом, за исключением этих двух одноадресных команд, все остальные команды являются безадресными, имеют длину 1 байт и выполняются по схеме:

$R2 := \text{ИЗСТЕКА}; R1 := \text{ИЗСТЕКА}; S := R1 \otimes R2; \text{ВСТЕК}(S)$

Как видно, для безадресных команд при выполнении бинарных операций уже все аргументы (два операнда и результат) задаются неявно и располагаются в стеке. Отсюда понятно, почему часто машины этой архитектуры называются *стековыми* ЭВМ.<sup>1</sup> На первый взгляд может показаться, что в стековых ЭВМ нарушается принцип фон Неймана однородности памяти, так как в стеке возможен доступ только к его вершине. Однако это не так: сам стек обычно является частью основной памяти, поэтому к любой ячейке стека возможен и прямой доступ, т.е. команды  $\text{ВСТЕК } A1$  и  $\text{ИЗСТЕКА } A1$  в качестве своих операндов  $A1$  могут указывать адрес произвольной ячейки памяти, в том числе и в любом месте стека.

Кроме рассмотренных выше видов машин, существовали и архитектуры ЭВМ с другим числом адресов. В качестве примера упомянем **четырёхадресные** машины, в четвёртом адресе которых дополнительно хранился ещё и адрес *следующей* выполняемой команды (для таких ЭВМ вообще не нужны команды переходов). Собственно, адресов может быть и больше, с помощью таких команд можно, например, реализовать уже не бинарные операции, а функции от 3-х и более переменных.

Далее, существуют архитектуры ЭВМ, которые различаются не только количеством *адресов* в машинной команде, но и наличием в такой команде нескольких *кодов операций*. Такие ЭВМ обычно называются машинами с *очень длинным командным словом* (VLIW – Very Large Instruction Word). Заметим, что, несмотря на необычность такой архитектуры, она может удовлетворять всем принципам фон Неймана. В таких компьютерах, например, некоторые команды могут реализовывать операторы присваивания вида  $x := a * b + c$  по схеме:<sup>2</sup>

$R1 := \langle a \rangle; R2 := \langle b \rangle; S := R1 * R2;$

$R1 := \langle c \rangle; S := S + R1; \langle x \rangle := S$

В ЭВМ с такой архитектурой команда, содержащая два кода операции и четыре адреса аргументов, в наших предыдущих предположениях о размере адреса и кода операции, имеет длину 14 байт и, например, такой формат:

КОП1	КОП2	A1	A2	A3	A4
------	------	----	----	----	----

Легко понять, что такие команды выполняются по схеме:

$R1 := \langle A2 \rangle; R2 := \langle A3 \rangle; S := R1 \text{ КОП1 } R2;$

$R1 := \langle A4 \rangle; S := S \text{ КОП2 } R1; \langle A1 \rangle := S$

Можно сказать, что компьютеры этой архитектуры как бы связывают несколько последователь-

<sup>1</sup> Надо отметить, что семантика таких популярных языков высокого уровня, как Java и Forth близка именно к стековой архитектуре исполнителя. По этой же схеме работали и первые программируемые калькуляторы.

<sup>2</sup> Такой набор команд под названием FMA для работы с *вещественными* числами реализован в процессорах Intel и AMD, начиная с 2013 года.

ных команд программы в одну длинную команду, в современных ЭВМ такая связка, иногда называемая пучком (bundle) команд, может содержать до семи кодов операций и соответствующее число операндов, а длина команды может достигать до 64 и более байт. Обычно построение таких длинных машинных команд производит компилятор с языка программирования высокого уровня для эффективного вычисления выражений. Главная идея VLIW архитектуры заключается в том, что на языке машины задаётся явный параллелизм: все операции над данными в такой длинной команде можно выполнять параллельно (в разных частях АЛУ), так как это *одна* команда.

Можно понять, что компьютеры с архитектурой VLIW будут наиболее эффективны при проведении научных расчётов в таких областях, как линейная алгебра, газовая динамика, физика твёрдого тела и т.д. Первый проект VLIW-компьютера был разработан А. Тьюрингом ещё в 1946 году (он так и не был реализован). Среди наиболее известных построенных в своё время компьютеров этой архитектуры для любознательных читателей можно назвать ЭВМ Multiflow и Cydra-5 (длина команды 16 байт), CDC6600, Эльбрус-3 и специализированный процессор TriVedia.

Далее архитектура VLIW получила своё дальнейшее развитие в компьютерах так называемой EPIC архитектуры (Explicitly Parallel Instruction Computing – вычисление с явно заданным параллелизмом команд). К сожалению, сейчас архитектура EPIC почти не используется, т.к. такие процессоры сложны в изготовлении, кроме того, для этой архитектуры очень трудно создать компиляторы с языков высокого уровня, а программировать на языке Ассемблера практически невозможно.

## 4.2. Сравнительный анализ ЭВМ различной адресности

*Человек без адреса подозрителен, человек с двумя адресами – тем более.*

*Бернард Шоу*

При изучении ЭВМ с разным количеством адресов естественно встаёт вопрос, какая архитектура лучше, например, даёт программы, занимающие меньше места в памяти. Заметим, что этот критерий для первых ЭВМ с их маленькой памятью был более важным, чем, например, удобство программирования на языке машины или скорость выполнения операций. Исследуем этот вопрос, составив небольшой фрагмент программы для рассмотренных выше учебных ЭВМ с различной адресностью. В качестве примера рассмотрим оператор присваивания, который содержит типичный набор арифметических операций:

$$x := a/(a+b)^2$$

В наших примерах будем использовать мнемонические коды операций и мнемонические имена для номеров ячеек памяти, в которых хранятся переменные (т.е. не будем производить явного распределения памяти, так как это несущественно для данного исследования). Кроме того, не будем конкретизировать тип используемых величин (целые или вещественные), предположим, что это тоже не влияет на размер программы. Вам необходимо внимательно просмотреть текст этих небольших фрагментов программ, и понять, как они работают.

### • Трёхадресная машина

СЛ	x	a	b	x := a+b
УМН	x	x	x	x := (a+b) <sup>2</sup>
ДЕЛ	x	a	x	x := a/(a+b) <sup>2</sup>

Длина этой программы в байтах: (3 команды)\*10 байт = 30 байт.

### • Двухадресная машина

ПЕР	R	a	R := a; копия a в рабочую переменную R
СЛ	R	b	R := a+b
УМН	R	R	R := (a+b) <sup>2</sup>
ПЕР	x	a	x := a;
ДЕЛ	x	R	x := a/(a+b) <sup>2</sup>

Длина этой программы: (5 команд)\*7 байт = 35 байт (плюс одна рабочая переменная R).

### • Одноадресная машина

СЧ	a	S := a
СЛ	b	S := a+b
ЗП	x	x := a+b
УМН	x	S := (a+b) <sup>2</sup>
ЗП	x	x := (a+b) <sup>2</sup>

СЧ	a	$S := a$
ДЕЛ	x	$S := a/(a+b)^2$
ЭП	x	$x := a/(a+b)^2$

Длина этой программы: (8 команд)\*4 байта = 32 байта. Как видим, длина программы для одноадресной ЭВМ получилась примерно такая же, как и для трёх и двухадресных машин. В то же время легко понять, что процессор одноадресной ЭВМ будет устроен проще, так как схема выполнения каждой команды требует меньше служебных регистров и реже обращается в память. Далее, простота процессора позволяет, при тех же затратах, увеличить скорость его работы за счёт реализации на более дорогих интегральных схемах. Отсюда понятна привлекательность одноадресной архитектуры для разработчиков ЭВМ первых поколений. Например, одноадресная отечественная ЭВМ БЭСМ-6 в конце 60-х годов прошлого века была одной из самых быстродействующих машин в мире [3].

#### • Безадресная машина

ВСТЕК	a	Поместить a в стек: a
ВСТЕК		Дублировать вершину стека: a, a
ВСТЕК	b	Теперь в стеке 3 числа: b, a, a
СЛ		В стеке два числа: a+b, a
ВСТЕК		Дублировать вершину стека, в стеке a+b, a+b, a
УМН		В стеке два числа: $(a+b)^2$ , a
ДЕЛ		В стеке одно число: $a/(a+b)^2$
ИЗСТЕКА	x	Запись результата из стека в x

Как видно, в безадресной ЭВМ есть и свои специфические безадресные "стековые" команды (дублировать вершину стека, поменять местами два верхних числа стека и др.). В данной программе использовались команды разной длины: 3 одноадресные для обмена со стеком и 5 безадресных команд. Длина всей программы: (3 команды)\*4 байта + (5 команд)\*1 байт = 17 байт.

Заметим, что безадресный способ выполнения операций нам хорошо знаком, именно так работают обычные калькуляторы. Например, сначала мы набираем в окне калькулятора (это вершина стека!) первое слагаемое, затем нажимаем клавишу "плюс", при этом первое слагаемое записывается в стек, а в окне мы набираем второе слагаемое. Затем, нажав клавишу "равно", мы заставляем калькулятор складывать два верхних числа из его стека, оставляя на вершине стека (в окне) сумму.

Итак, с уменьшением количества адресов в команде увеличивается число команд программы, зато каждая команда становится более короткой, занимает меньше места в памяти. Наше небольшое исследование показало, что архитектура ЭВМ с безадресными командами даёт более компактные программы, кроме того, процессор у них устроен проще, чем у двух и трёхадресных ЭВМ. Именно поэтому в начале развития вычислительной техники такие компьютеры были весьма распространены, их, в частности, выпускала известная фирма Барроуз (Burroughs) [3]. Однако в дальнейшем были предложены ЭВМ с другой архитектурой, которая позволила писать не менее компактные машинные программы, и при этом обладала дополнительными достоинствами, поэтому в настоящее время полностью стековые ЭВМ практически не используются.<sup>1</sup>

### 4.3. Архитектура с адресуемыми регистрами

*Главный враг знания не невежество, а иллюзия знания.*

*Стивен Хокинг*

Далее будет рассмотрена архитектура ЭВМ с адресуемыми регистрами. Эти компьютеры дают возможность писать такие же компактные программы, как и ЭВМ с безадресной системой команд, но при этом обладать рядом дополнительных достоинств.

Компьютеры с адресуемыми регистрами нарушают принцип фон Неймана линейности и однородности памяти. В этих компьютерах память, к которой может непосредственно (по адресам) обращаться процессор за *операндами* команд ("числами"), состоит из двух частей, каждая со своей независимой нумерацией ячеек (это и есть нарушение линейности памяти). Одна из этих частей называ-

<sup>1</sup> Кроме того, выпускались компьютеры и смешанной адресации. Так, уже упоминавшаяся ранее отечественная одноадресная ЭВМ БЭСМ-6, кроме одноадресных арифметических операций могла выполнять и аналогичные безадресные (стековые) операции. Стековую организацию имеет и регистровая память для работы с вещественными числами в широко распространённых ЭВМ фирмы Intel.

ется *адресуемой регистровой памятью* и имеет небольшой объём (порядка десятков, редко сотен ячеек), а другая называется *основной (оперативной) памятью* достаточно большого объёма. Ячейка каждого из видов памяти имеет свой адрес, но в случае с маленькой регистровой памятью этот адрес имеет размер в битах в несколько раз меньший, чем адрес ячейки большой основной памяти. Кроме того, ячейки регистровой и основной памяти могут иметь разную длину.

Например, построим учебную двухадресную ЭВМ этой новой архитектуры, которую назовём УМ-Р (Учебная Машина с адресуемыми Регистрами). Пусть регистровая память этой учебной ЭВМ состоит из 16 ячеек по 4 байта каждая. В этом случае адрес каждого регистра лежит в диапазоне  $0 \div 15$ , и будет помещаться в 4 бита, а основная память пусть содержит  $2^{20}$  ячеек, тогда адрес каждой ячейки занимает 20 двоичных разрядов. В такой ЭВМ в качестве адресов операндов каждой команды могут быть или адреса двух регистров, или один адрес регистра и один адрес ячейки основной памяти (заметим, что *команды* устройство управления по-прежнему может читать только из основной памяти). Адреса регистров на схемах команд будут обозначаться  $r1$  и  $r2$ , а адреса основной памяти, как и раньше,  $A1$  или  $A2$ . Первый вид команд будем называть командами формата *регистр-регистр* (обозначается RR), а второй – формата *регистр-память* (обозначается RX).<sup>1</sup> В этом случае для одного кода операции (например, сложения) получаются команды двух форматов длины 2 и 4 байта соответственно:

коп	$r1$	$r2$
8 бит	4 бита	4 бита

= 2 байта

коп	$r1$	$A2$
8 бит	4 бита	20 бит

= 4 байта

В качестве преимущества этой архитектуры нужно отметить, что ячейки регистровой памяти, так как их немного, можно разместить *внутри* процессора, они будут иметь статус *регистров*. Это позволяет производить на них арифметические и логические операции (что, как Вы помните, в основной памяти по принципам фон Неймана это невозможно). Кроме того, такое расположение обеспечивает быстрый доступ к хранимым на регистрах данным (не требуется делать обмен с расположенной отдельно от процессора основной памятью). Как правило, время доступа к регистру у ЭВМ на порядок меньше, чем к ячейке основной памяти (это и есть нарушение принципа однородности памяти). Всё это позволяет существенно повысить быстродействие такого компьютера.

Заметим, что, кроме *адресуемых* регистров, номера которых явно указываются в команде, в арифметико-логическом устройстве по-прежнему могут быть и не адресуемые регистры, например, уже знакомые нам регистры первого и второго операнда  $R1$  и  $R2$ , а также регистр сумматора  $S$ . Таким образом, команда формата регистр-память 

коп	$r1, A2$
-----	----------

 должна выполняться процессором, например, по схеме

$R2 := \langle A2 \rangle; S := r1 \otimes R2; r1 := S$

Отметим, что содержимое регистров, в отличие от содержимого ячеек основной памяти, принято записывать без угловых скобок, так что, например, запись  $r1$  в приведённом выше примере обозначает содержимое регистра  $r1$ , а не номер (адрес) этого регистра.

Из сказанного выше можно сделать вывод, что при программировании на таких ЭВМ желательно как можно чаще использовать регистровую память и как можно реже обращаться к большой основной памяти, этого правила мы и будем придерживаться.<sup>2</sup> Теперь составим для нашей машины УМ-Р фрагмент программы, который реализует, как и в предыдущих примерах, оператор присваивания  $x := a / (a + b)$ .<sup>2</sup> Мнемонические коды операций задают арифметические операции с обычным смыслом, а  $r1$  и  $r2$  обозначают номера адресуемых регистров. Операнды в команде будем разделять запятой, а точка с запятой, как это принято в языке Ассемблера, задаёт начало *комментария* к команде:

СЧ  $r1, a; r1 := a$

<sup>1</sup> Вскоре выяснится, что третий логически допустимый формат команд (память-память), когда оба адреса принадлежат основной памяти, не даёт при программировании в этой архитектуре никаких преимуществ. Исходя из этого, такой формат в этой учебной машине не реализован.

<sup>2</sup> В некоторых ЭВМ так называемой RISC архитектуры этот принцип возведён в абсолют: все действия над данными производятся *только* командами формата регистр-регистр, оставляя для формата регистр-память *только* операции обмена между регистрами и основной памятью.

```

СЧ  r2,b;  r2 := b
СЛ  r2,r1; r2 := b+a=a+b
УМН r2,r2; r2 := (a+b)2
ДЕЛ r1,r2; r1 := a/(a+b)2
ЗП  x,r1;  x  := r1=a/(a+b)2

```

Длина этого фрагмента программы равна

(3 команды)\*4 байта + (3 команды)\*2 байта = 18 байт.

Как видим, данная архитектура, обладая отмеченными выше преимуществами, не уступает стекковой (безадресной) архитектуре по компактности получаемых программ.

Рассмотрим теперь главный недостаток ЭВМ с адресуемыми регистрами. Заметим, что если ранее для каждой арифметической операции было необходимо реализовать по одной команде (для целых и вещественных чисел), то теперь число этих команд возросло вдвое из-за необходимости реализовывать эти команды как в формате RR, так и в формате RX. Это приводит к усложнению процессора, который отныне должен поддерживать значительно большее количество операций. Однако преимущества архитектуры с адресуемыми регистрами настолько очевидны, что её имеют большинство современных ЭВМ.

В этой архитектуре встречаются команды разного формата (и, соответственно, разной длины). Как говорится, современные ЭВМ обладают *многообразием форматов команд*. Например, на широко распространённых персональных компьютерах фирмы Intel реализовано около десяти форматов команд, а длина команд составляет от 1 до 15 байт.

Разумеется, с развитием вычислительной техники в компьютерах появилось и много новых особенностей, некоторые из которых будут рассмотрены далее в этой книге.

#### 4.4. Способы адресации

*Если тебя посылают, то уточни, на всякий случай, адрес.*

Перейдём теперь к изучению другого важного понятия в архитектуре ЭВМ. Введём следующее определение. *Способ адресации* – это способ задания операндов внутри машинной команды. Другими словами это правила, по которым заданные в команде (двоичные) числа определяют местонахождение и значение *операндов* для данной команды. Как правило, способ адресации операндов определяется только кодом операции команды.

Как Вам уже известно, часть операндов может вообще задаваться неявно, т.е. в самой команде нет полей, которые определяют месторасположение таких операндов. Теперь остаётся только разобратся со способами адресации *явных* операндов.

Для лучшего усвоения этого очень важного понятия модифицируем описанную нами ранее одноадресную учебную ЭВМ УМ-1, введя в её язык команды сложения целых чисел с разными способами адресации. Мнемоника кодов операций сложения будет указывать на способ адресации.

- **Прямой способ адресации**

СЛ	2	S := S + <2>
----	---	--------------

При этом способе адресации число на месте операнда задаёт *адрес* ячейки основной памяти, в котором и содержится необходимый в команде операнд. Будем, как обычно в угловых скобках обозначать *содержимое* ячейки основной памяти с данным адресом. Так, в приведённом выше примере <2> обозначает содержимое ячейки с адресом 2. Важно понять, что в этой ячейке, конечно же, скорее всего, хранится *не* число 2.

- **Непосредственный способ адресации**

СЛН	2	S := S+2
-----	---	----------

При таком способе адресации поле адреса команды содержит, как говорят, *непосредственный* (immediate) операнд. Таким образом, число 2 в нашем примере обозначает не ячейку памяти с адресом 2, а непосредственно целочисленное значение 2. Разумеется, такие непосредственные операнды могут быть только (неотрицательными) целыми числами, по величине не превышающими максимального значения, которое можно записать в поле адреса.

Использование непосредственного метода адресации позволяет не располагать (целочисленные) константы в ячейках памяти, а помещать их внутрь команд, (на место адреса операнда), что может сильно сэкономить память. Заметим, что это же позволяет лучше защитить константы от случайной

порчи при ошибочной записи в те ячейки памяти, где они расположены. Это, разумеется, повышает надёжность программирования на таких ЭВМ. Само программирование на языке машины также упрощается, так как теперь не надо производить распределение памяти под хранение таких констант.

Заметим, что прямая адресация использовалась во всех рассмотренных ранее учебных машинах, исключение составляли только команды ввода/вывода УМ-3, где второй адрес задавал не *номер* ячейки памяти, а *число* чисел в массиве для ввода или вывода. Таки образом, это был *непосредственный*, а не прямой операнд.

- **Косвенный способ адресации**

СЛК	2
-----	---

 $S := S + \langle\langle 2 \rangle\rangle$ 

Здесь число на месте операнда задаёт *адрес* ячейки памяти, содержимое которой, в свою очередь, трактуется как целое число – адрес необходимого операнда в памяти ЭВМ. Таким образом, число 2 в нашем примере является *косвенным* (indirect) адресом операнда. При таком способе адресации для доступа к операнду процессору необходимо дважды обратиться к основной памяти:

 $R1 := \langle A2 \rangle; R1 := \langle R1 \rangle; S := r1 \otimes R1$ 

В качестве примера выполним несколько команд сложения с различными способами адресации для этой учебной одноадресной ЭВМ, и рассмотрим значение регистра сумматора  $S$  после выполнения этих команд (см. рис. 4.1). В комментариях к каждой команде показаны производимые этой командой действия. Справа на этом рисунке показаны первые ячейки памяти и хранимые в них целые числа. В этом примере, как и в самой первой учебной машине УМ-3, предполагается, что длина команды совпадает с длиной числа и длиной машинного слова.

	Адрес	Значение ячейки
СЧ 2; $S := \langle 2 \rangle = 3$	000	1
СЛ 2; $S := S + \langle 2 \rangle = 6$	001	2
СЛН 2; $S := S + 2 = 8$	002	3
СЛК 2; $S := S + \langle\langle 2 \rangle\rangle = S + \langle 3 \rangle = S + 4 = 12$	003	4

Рис. 4.1. Значение регистра сумматора после выполнения одноадресных команд сложения с различными способами адресации.

## 4.5. Многообразие форматов данных

*Во многих мудрости много печали; и кто умножает познания, умножает скорбь.*

Соломон. «Екклесиаст»

Современные ЭВМ позволяют совершать операции над целыми и вещественными числами разной длины. Это вызвано чисто практическими соображениями. Например, если нужно нам целое число помещается в один байт, то неэкономно использовать под его хранение два или четыре числа байта (экономия будет особенно заметной, если необходимо хранить большой *массив* таких чисел). Во избежание такого неоправданного расхода памяти введены соответствующие *форматы данных*, отражающие представление в памяти ЭВМ чисел разной длины. Например, в зависимости от размера целого числа, оно может занимать в памяти 1, 2, 4, 8 и более байт. Приведённая ниже таблица иллюстрирует *многообразие форматов данных*, для представления целых чисел на широко распространённых персональных ЭВМ.

Размер (байт)	Название формата	Обозначение в Ассемблере
1	Короткое целое	<b>db</b>
2	Длинное целое (слово)	<b>dw</b>
4	Двойное слово	<b>dd</b>
8	Четверное слово	<b>dq</b>
16	Восьмерное слово <sup>1</sup>	<b>do</b>

Многообразие форматов данных требует усложнения архитектуры как устройства управления (резко возрастает число команд в языке машины), так и арифметико-логического устройства, в частности, регистровой памяти. Теперь регистры должны уметь хранить, а само арифметико-логическое устройство – обрабатывать данные *разной длины*.

<sup>1</sup> Только в 64-битных Ассемблерах.



## 4.6. Форматы команд

*А "язык" процессоров x86, между прочим, очень интересен. На сегодняшний день они имеют едва ли не самую сложную систему команд, дающую системным программистам безграничные возможности для самовыражения. Прикладные программисты даже не догадываются, сколько красок мира у них украли компиляторы!*

*Крис Касперски aka мыцъх*

Для операций с разными способами адресации и разными форматами данных необходимо введение различных *форматов команд*, которые по-разному задают местонахождение и количество операндов, и, естественно, имеют разную длину. Для широко распространённых сейчас двухадресных ЭВМ это такие форматы команд (в скобках указано их мнемоническое обозначение):

- регистр – регистр (RR);
- регистр – память, память – регистр (RX);
- регистр – непосредственный операнд в команде (RI);
- память – непосредственный операнд в команде (SI);
- память – память, т.е. оба операнда в основной памяти (SS).<sup>1</sup>

Многообразие форматов команд и данных позволяет писать более компактные и эффективные программы на языке машины, однако, как уже упоминалось, сильно усложняет процессор ЭВМ.

Возвращаясь к учебной ЭВМ УМ-3 теперь можно, используя новые изученные понятия, сказать, что это трёхадресная машина с прямым способом адресации (за исключением команд ввода/вывода), одним форматом команд и одним форматом данных.

## 4.7. Базирование адресов

*Записал я длинный адрес на бумажном лоскутке.*

*Арсений Тарковский*

Для дальнейшего уменьшения объёма программы современные ЭВМ могут использовать новый способ адресации, основанный на принципе *базирования адресов*. Изучение этого важного и нового понятия проведём на следующем примере. Пусть в программе на учебной машине УМ-1 необходимо реализовать арифметический оператор присваивания  $Z := (X+Y)$ <sup>2</sup>. Ниже приведена эта часть программы с соответствующими комментариями (напомним, что S – это регистр сумматора одноадресной ЭВМ):

```
СЧ X; S:=X
СЛ Y; S:=X+Y
ЗП R; R:=X+Y – запись в рабочую переменную
УМ R; S:=(X+Y)2
ЗП Z; Z:=(X+Y)2
```

В этой одноадресной учебной ЭВМ имеется  $2^{24}$  (примерно 16 миллионов) ячеек памяти по 32 бит в ячейке, в каждой ячейке помещается одна команда или одно число. Будем, не теряя общности, считать, что наш фрагмент программы располагается где-то примерно в середине памяти. Пусть, например, наши переменные при распределении памяти оказались помещены соответственно в следующих ячейках памяти (как и ранее, адреса для удобства даны в десятичной системе счисления):

```
X – в ячейке с адресом 10 000 000
Y – в ячейке с адресом 10 000 001
Z – в ячейке с адресом 10 000 002
R – в ячейке с адресом 10 000 003
```

---

<sup>1</sup> Использование для мнемонического обозначения операнда в памяти сразу двух букв X (RX) и S (SS) связано с особенностями выполнения команд этих форматов (X – явный, а S – неявный операнд, т.е. заданный по умолчанию), надо не задумываться над этим, а просто запомнить.

Тогда приведённый выше фрагмент программы после замены мнемонических обозначений переменных их адресами будут выглядеть следующим образом:

```

СЧ 10 000 000; S:=X
СЛ 10 000 001; S:=X+Y
ЗП 10 000 003; R:=X+Y
УМ 10 000 003; S:=(X+Y)2
ЗП 10 000 002; Z:=(X+Y)2

```

Из этого примера видно, что большинство адресов в нашей программе можно представить в виде выражения  $B+\Delta$ , где число  $B$  назовём *базовым адресом* программы или просто *базой* (в нашем случае  $B=10\ 000\ 000$ ), а число  $\Delta$  – *смещением* адреса относительно этой базы. Здесь налицо существенная *избыточность* информации в программе. Очевидно, что в каждой команде можно указывать только короткое *смещение*  $\Delta$ , а базу хранить отдельно (обычно на каком-то специальном *базовом* регистре процессора). Исходя из этих соображений, предусмотрим в машинном языке нашей одноадресной ЭВМ команду *загрузки базы* (длина этой команды 4 байта):

ЗГБ	A1
8 бит	24 бита

Тогда наш фрагмент программы будет иметь такой вид:

```

ЗГБ 10000000
...
СЧ 000; S:=X
СЛ 001; S:=X+Y
ЗП 003; R:=X+Y
УМ 003; S:=(X+Y)2
ЗП 002; Z:=(X+Y)2

```

Как видно, в большинстве команд можно теперь вместо длинного *адреса* ячейки памяти указывать только короткое *смещение*  $\Delta$  этой ячейки относительно базы. Это позволит значительно уменьшить размер программы. Заметим, однако, что теперь при выполнении *каждого* обращения за операндом в основную память, процессор должен *вычислять* значение адреса этого операнда по формуле  $A=B+\Delta$ . Это вычисление производится в устройстве управления и, естественно, усложняет его, не говоря уже о том, что и выполнение всей команды может несколько замедлиться. Например, адрес переменной  $X$  вычисляется как  $\text{Адрес}(X) = B+\Delta=10^7+1=10000001$ .

Осталось выбрать оптимальную длину максимального смещения  $\Delta$  в поле адреса команды. Для этого вернёмся к рассмотрению учебной ЭВМ с адресуемыми регистрами, для которой теперь будет реализовано базирование адресов основной памяти (не регистровой, там это не нужно, так как адреса регистров и так маленькие). Например, пусть под запись смещения в команде выделено поле длиной в 12 бит. Будем, как и раньше, обозначать операнд в памяти  $A1$  или  $A2$ , но помним, что теперь это только *смещение*  $\Delta$  относительно базы. Тогда все команды, которые обращаются за операндом в основную память, будут в нашей ЭВМ с адресуемыми регистрами более короткими:<sup>1</sup>

КОП	r1	A2
8 бит	4 бита	12 бит

Рассмотрим схему выполнения такой команды для формата регистр-память ( $\otimes$  как обычно задаёт какой-то код бинарной операции):

$r1 := r1 \otimes \langle B+A2 \rangle$

или для формата память-регистр:

$\langle B+A2 \rangle := \langle B+A2 \rangle \otimes r1$

Область, в которой находятся вычисляемые относительно базы ячейки основной памяти, называется *сегментом* – это сплошной участок памяти, начало которого определяется некоторым *сегментным* регистром. Разбиения памяти на такие участки называется *сегментированием* памяти.

Сегментирование позволяет уменьшить объём памяти для хранения программ, но оно имеет и один существенный *недостаток*: теперь каждая команда может обращаться не к *любой* ячейке оперативной памяти, как это было у нас раньше, а только к тем из ячеек, до которых "дотягивается" сме-

<sup>1</sup> Таким был формат команд популярного в 60-70-х годах прошлого века семейства ЭВМ IBM-360/370 и их отечественных аналогов ряда ЭВМ Единой серии (ЕС ЭВМ).

щение относительно своей базы. В нашем предыдущем примере при длине смещения 12 бит каждая команда может обращаться к диапазону адресов от значения сегментного регистра В до  $B+2^{12}-1$ . Для доступа к другим ячейкам памяти необходимо записать в сегментный регистр новое значение (как говорят, *перезагрузить* сегментный регистр). Этот недостаток также существенно затрудняет в таких компьютерах работу с массивами длиной больше, чем  $2^{12}$  ячеек. Однако, несмотря на указанный недостаток, ЭВМ первых поколений в целях уменьшения объёма программы часто позволяли производить сегментирование памяти и базирование адресов команд.

Заметим также, что отмеченный выше недостаток исправляется путём реализации нескольких сегментных регистров, а также *переменной* длины смещения (например, разрешается смещение длиной в 1, 2, 4 или 8 байт). Необходимо, однако, понимать, что это ещё более увеличивает набор команд языка машины и усложняет процессор.

Итак, ещё раз отметим очень важное обстоятельство сегментного способа адресации. В новой архитектуре для осуществления *любого* доступа к памяти ЭВМ необходимо, чтобы ячейка, к которой осуществляется доступ, находилась в сегменте, на начало которого указывает некоторый сегментный регистр. Как уже говорилось, ЭВМ могут обеспечивать одновременную работу с несколькими сегментами памяти и, соответственно, иметь несколько сегментных регистров.

В ЭВМ используется не только сегментная организации памяти, но и для так называемое *базирование* (относительной адресации команд и данных). При относительной адресации некоторый *базовый* регистр ставится в средину участка памяти, к которому необходимо осуществлять доступ, и используется *знаковое* смещение. На рис. 4.2 показан доступ к ячейкам памяти при помощи сегментирования и базирования.

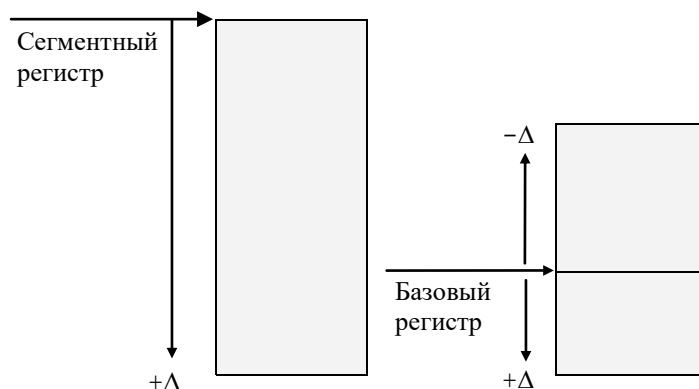


Рис. 4.2. Доступ к памяти при помощи сегментирования и базирования.

Относительная адресация широко используется для доступа к локальным переменным процедур и функций (в так называемом стековом кадре), а также для команд относительного перехода, когда в качестве базового регистра используется регистр (счётчик) адреса, который, как Вы знаете, указывает на следующую выполняемую команду. При величине  $\Delta$  в один байт получаются очень компактные программы.

**+** В некоторых архитектурах адресуемые регистры процессора являются *универсальными*, т.е. каждый из них может быть использован как сегментный или для выполнения любых операций над данными. Сложность процессора при этом существенно повышается, поэтому во многих архитектурах используются *специализация* регистров, т.е. определённые регистры являются только сегментными (на них нельзя, например, складывать числа), другие не являются сегментными, но могут производить операции над данными, третьи используются в качестве счётчиков циклов и т.д.

### Вопросы и упражнения

1. Что такое адресность ЭВМ ?
2. Почему в безадресной ЭВМ должны быть хотя бы две одноадресные команды ?
3. Какие ЭВМ называются компьютерами с адресуемыми регистрами ? В чём достоинства и недостатки такой архитектуры ?
4. Что такое способ адресации ? Чем отличается прямая адресация от косвенной ?

5. Добавьте в язык учебной машины УМ-3 новую команду *косвенной* пересылки:

ПЕРК A1, A2, A3

Эта команда использует косвенную адресацию по своему третьему адресу и выполняется по правилу:

$\langle A1 \rangle := \langle \langle A3 \rangle \rangle$

Покажите, что с помощью этой команды можно обрабатывать массивы без использования самомодифицирующихся программ. Переделайте для этого программу суммирования всех элементов массива из примера на рис. 3.4.

6. Для чего нужно многообразие форматов данных ?
7. Почему в архитектуре возникает многообразие форматов команд ?
8. Почему есть формат команд память-непосредственный операнд (SI), но нет формата команд непосредственный операнд-память (IS) ?
9. В чём преимущества и недостатки сегментной организации памяти ?
10. Что такое базирование адресов ?