Глава 2

Архитектура системы команд

Системой команд вычислительной машины называют полный перечень команд, которые способна выполнять данная ВМ. В свою очередь, под *архитектурой системы команд* (АСК) принято определять те средства вычислительной машины, которые видны и доступны программисту. АСК можно рассматривать как линию согласования нужд разработчиков программного обеспечения с возможностями создателей аппаратуры вычислительной машины (рис. 2.1).

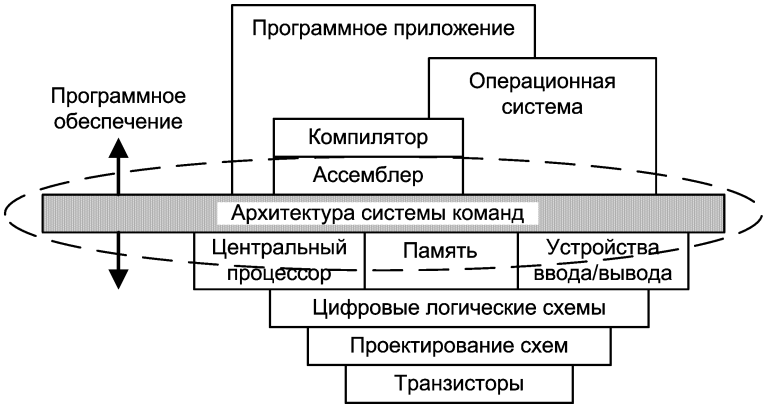

Рис. 2.1. Архитектура системы команд как интерфейс между программным и аппаратным обеспечением

В конечном итоге цель тех и других — реализация вычислений наиболее эффективным образом, то есть за минимальное время, и здесь важнейшую роль играет правильный выбор архитектуры системы команд.

В упрощенной трактовке время выполнения программы ($T_{\text{выч}}$) можно определить через число команд в программе ($N_{\text{ком}}$), среднее количество тактов процессора, приходящихся на одну команду (CPI), и длительность тактового периода $\tau_{\text{пр}}$:

$$T_{\text{выч}} = N_{\text{ком}} \times CPI \times \tau_{\text{пр}}.$$

Каждая из составляющих выражения зависит от одних аспектов архитектуры системы команд и, в свою очередь, влияет на другие (рис. 2.2), что свидетельствует о необходимости чрезвычайно ответственного подхода к выбору АСК.

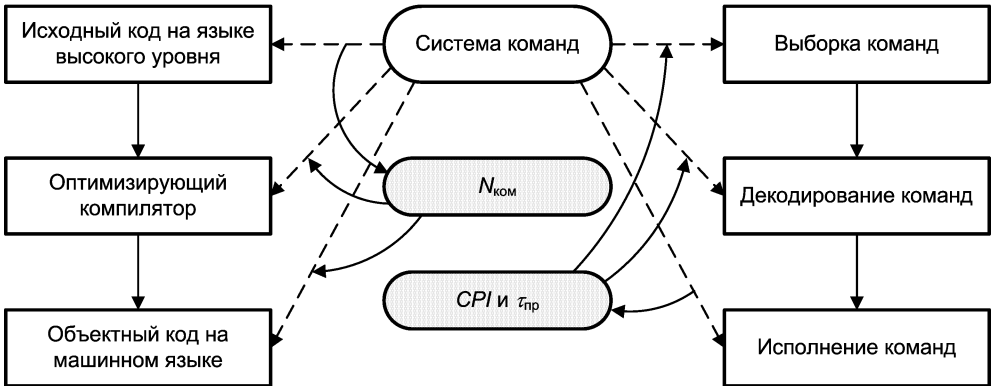

Рис. 2.2. Взаимосвязь между системой команд и факторами, определяющими эффективность вычислений

Общая характеристика архитектуры системы команд вычислительной машины складывается из ответов на следующие вопросы:

1. Какого вида данные будут представлены в вычислительной машине и в какой форме?
2. Где эти данные могут храниться помимо основной памяти?
3. Каким образом будет осуществляться доступ к данным?
4. Какие операции могут быть выполнены над данными?
5. Сколько операндов может присутствовать в команде?
6. Как будет определяться адрес очередной команды?
7. Каким образом будут закодированы команды?

Предметом данной главы является обзор наиболее распространенных архитектур системы команд, как в описательном плане, так и с позиций эффективности.

Классификация архитектур системы команд

В истории развития вычислительной техники как в зеркале отражаются изменения, происходившие во взглядах разработчиков на перспективность той или иной архитектуры системы команд. Сложившуюся на настоящий момент ситуацию в области АСК иллюстрирует рис. 2.3.

Среди мотивов, чаще всего предопределяющих переход к новому типу АСК, остановимся на двух наиболее существенных. Первый — это состав операций, выполняемых вычислительной машиной, и их сложность. Второй — место хранения операндов, что влияет на количество и длину адресов, указываемых в адресной части команд обработки данных. Именно эти характеристики взяты в качестве показателей классификации архитектур системы команд.

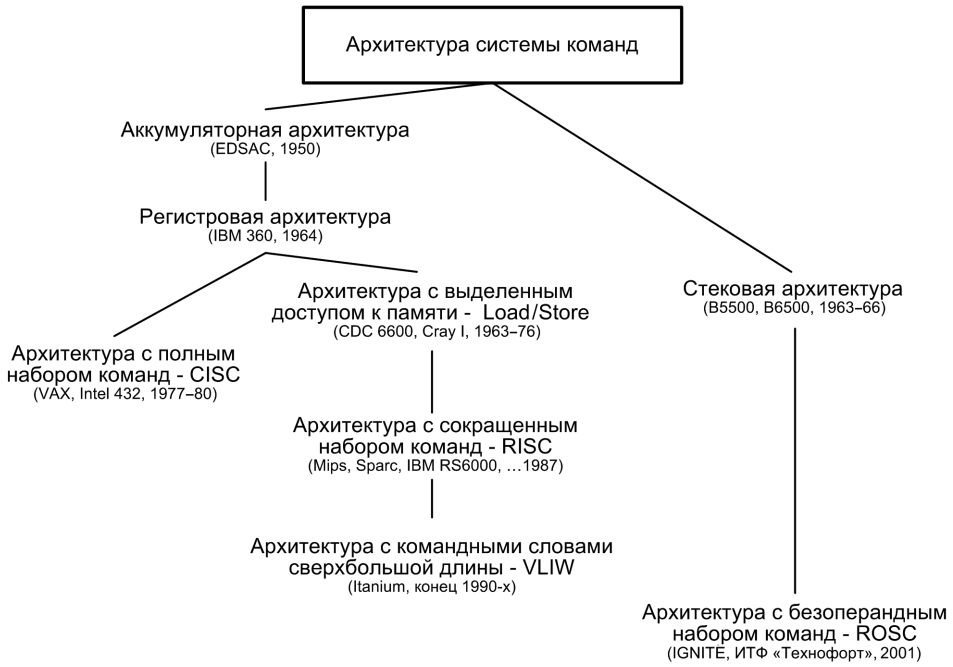

Рис. 2.3. Хронология развития архитектур системы команд

Классификация по составу и сложности команд

Современная технология программирования ориентирована на языки высокого уровня (ЯВУ), главная цель которых — облегчить процесс программирования. Но переход к ЯВУ породил серьезную проблему: сложные операторы, характерные для ЯВУ, существенно отличаются от простых машинных операций, реализуемых в большинстве вычислительных машин. Следствием такого несоответствия становится недостаточно эффективное выполнение программ на ВМ. Проблема получила название *семантического разрыва*, а для ее разрешения разработчики вычислительных машин в настоящее время выбирают один из трех подходов и, соответственно, один из трех типов АСК:

- архитектуру с полным набором команд: CISC (Complex Instruction Set Computer);
- архитектуру с сокращенным набором команд: RISC (Reduced Instruction Set Computer);
- архитектуру с командными словами сверхбольшой длины: VLIW (Very Long Instruction Word).

В *CISC-архитектуре* семантический разрыв преодолевается за счет расширения системы команд, дополнения ее сложными командами, семантически аналогичными операторам ЯВУ. Основоположником CISC-архитектуры считается компания IBM, которая начала применять данный подход с семейства машин IBM 360

и продолжает его в своих мощных современных универсальных ВМ (мэйнфреймах). Аналогичный подход характерен и для компании Intel в ее микропроцессорах серии x86. Для CISC-архитектуры типичны:

- наличие в процессоре сравнительно небольшого числа регистров общего назначения;
- большое количество машинных команд, часть из которых аппаратно реализуют сложные операторы ЯВУ;
- разнообразие способов адресации операндов;
- множество форматов команд различной разрядности;
- наличие команд, где обработка совмещается с обращением к памяти.

К типу CISC можно отнести практически все ВМ, выпускавшиеся до середины 1980-х годов, и значительную часть производящихся в настоящее время. Рассмотренный способ решения проблемы семантического разрыва вместе с тем ведет к усложнению аппаратуры ВМ, главным образом, устройства управления, что, в свою очередь, негативно сказывается на производительности ВМ в целом. Это обстоятельство побудило более внимательно проанализировать программы, получаемые после компиляции с ЯВУ. Был предпринят комплекс исследований [99, 115, 128, 151], в результате которых обнаружилось, что доля дополнительных команд, эквивалентных операторам ЯВУ, в общем объеме программ не превышает 10–20%, а для некоторых наиболее сложных команд даже 0,2%. В то же время объем аппаратных средств, требуемых для реализации таких дополнительных команд, возрастает весьма существенно. Так, емкость микропрограммной памяти, хранящей микропрограммы выполнения всех команд ВМ, из-за введения сложных команд может увеличиваться на 60%.

Детальный анализ результатов упомянутых исследований привел к серьезному пересмотру традиционных решений, следствием чего стало появление *RISC-архитектуры*. Термин RISC впервые был использован Д. Паттерсоном и Д. Дитцелем в 1980 году [128]. Идея заключается в ограничении списка команд ВМ наиболее часто используемыми простейшими командами, оперирующими данными, размещенными только в регистрах процессоров. Обращение к памяти допускается лишь с помощью специальных команд чтения и записи. Резко уменьшено количество форматов команд и способов указания адресов операндов. Эти меры позволили существенно упростить аппаратные средства ВМ и повысить их быстродействие. RISC-архитектура разрабатывалась таким образом, чтобы уменьшить $T_{\text{выч}}$ за счет сокращения *CPI* и $\tau_{\text{пр}}$. Оказалось, что реализация сложных команд за счет последовательности из простых, но быстрых RISC-команд не менее эффективна, чем аппаратный вариант сложных команд в CISC-архитектуре.

Элементы RISC-архитектуры впервые появились в вычислительных машинах CDC 6600 и суперЭВМ компании Cray Research. Достаточно успешно реализуется RISC-архитектура и в современных ВМ.

Отметим, что в последнее время в микропроцессорах компаний Intel и AMD широко используются идеи, свойственные RISC-архитектуре, так что многие различия между CISC и RISC постепенно стираются.

Помимо CISC- и RISC-архитектур, в общей классификации был упомянут еще один тип АСК — архитектура с командными словами сверхбольшой длины (VLIW). Концепция VLIW базируется на RISC-архитектуре, но в ней несколько простых RISC-команд объединяются в одну сверхдлинную команду и выполняются параллельно. В плане АСК архитектура VLIW сравнительно мало отличается от RISC. Появился лишь дополнительный уровень параллелизма вычислений, в силу чего архитектуру VLIW логичнее адресовать не к вычислительным машинам, а к вычислительным системам.

Таблица 2.1. Сравнительная оценка CISC-, RISC- и VLIW-архитектур

Характеристика	CISC	RISC	VLIW
Длина команды	Варьируется	Единая	Единая
Расположение полей в команде	Варьируется	Неизменное	Неизменное
Количество регистров	Несколько (часто специализированных)	Много регистров общего назначения	Много регистров общего назначения
Доступ к памяти	Может выполняться как часть команд различных типов	Выполняется только специальными командами	Выполняется только специальными командами

Таблица 2.1 позволяет оценить наиболее существенные различия в архитектурах типа CISC, RISC и VLIW.

Классификация по месту хранения операндов

Количество команд и их сложность, безусловно, являются важнейшими факторами, однако не меньшую роль при выборе АСК играет ответ на вопрос о том, где могут храниться операнды и каким образом к ним осуществляется доступ. С этих позиций различают следующие виды архитектур системы команд:

- стековую;
- аккумуляторную;
- регистровую;
- с выделенным доступом к памяти.

Выбор той или иной архитектуры влияет на принципиальные моменты: сколько адресов будет содержать адресная часть команд, какова будет длина этих адресов, насколько просто будет происходить доступ к операндам и какой, в конечном итоге, будет общая длина команд.

Стековая архитектура

Стеком называется память, по своей структурной организации отличная от основной памяти ВМ. Принципы построения стековой памяти детально рассматриваются позже, здесь же выделим только те аспекты, которые требуются для пояснения особенностей АСК на базе стека.

Стек образует множество логически взаимосвязанных ячеек (рис. 2.4), взаимодействующих по принципу «последним вошел, первым вышел» (LIFO, Last In First Out).

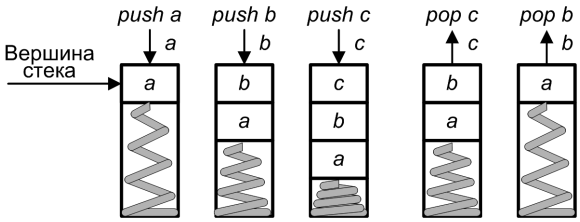

Рис. 2.4. Принцип действия стековой памяти

Верхнюю ячейку называют *вершиной стека*. Для работы со стеком предусмотрены две операции: *push* (проталкивание данных в стек) и *pop* (выталкивание данных из стека). Запись возможна только в верхнюю ячейку стека, при этом вся хранящаяся в стеке информация предварительно проталкивается на одну позицию вниз. Чтение допустимо также только из вершины стека. Извлеченная информация удаляется из стека, а оставшееся его содержимое продвигается вверх. В вычислительных машинах, где реализована АСК на базе стека (их обычно называют стековыми), операнды перед обработкой помещаются в две верхних ячейки стековой памяти. Результат операции заносится в стек. Принцип действия стековой машины поясним на примере вычисления выражения $(a + b) * (c + d) - e$.

При описании вычислений с использованием стека обычно используется иная форма записи математических выражений, известная как обратная польская запись (обратная польская нотация), которую предложил польский математик Я. Лукашевич. Особенность ее в том, что в выражении отсутствуют скобки, а знак операции располагается не между операндами, а следует за ними (постфиксная форма).

Преобразование традиционной записи выражения в постфиксную удобно выполнять с помощью вспомогательного стека. Назовем его стеком последовательности операций (СПО), чтобы не путать со стеком вычислительной машины. Для работы с СПО зададим приоритеты операций исходного (традиционного) выражения (табл. 2.2). В табл. 2.2 нулем обозначен минимальный, а четверкой — максимальный приоритет.

Таблица 2.2. Приоритеты операций для работы с СПО

Операция	**Символ операции**	**Приоритет**
Открывающая скобка	(	0
Закрывающая скобка	)	1
Сложение | вычитание	+ | –	2
Умножение | деление	* | /	3
Возведение в степень	**	4

По мере просмотра традиционной строки в СПО помещаются встречающиеся символы операций, которые в дальнейшем (по определенным правилам) переносятся в постфиксную строку. Формирование строки с выражением в обратной польской нотации осуществляется в соответствии со следующим алгоритмом:

1. Традиционная (входная) строка с выражением просматривается слева направо.
2. Если очередной символ входной строки — операнд, он сразу переписывается в постфиксную строку.
3. Левая скобка, а также символ математической операции в случае, если СПО пуст, всегда заносится в СПО.
4. Когда при просмотре встречается правая скобка, символ, находящийся на вершине СПО, выталкивается из СПО и заносится в постфиксную строку. Процедура повторяется вплоть до появления на вершине СПО левой скобки. Когда это происходит, обе скобки взаимно уничтожаются, при этом левая скобка из СПО удаляется.
5. Если СПО пуст или символ операции во входной строке имеет более высокий приоритет, чем символ на вершине СПО, символ операции из входной строки заталкивается в СПО.
6. Если приоритет символа во входной строке равен или ниже приоритета символа на вершине СПО, символ из вершины СПО выталкивается в постфиксную строку. Процедура повторяется до тех пор, пока на вершине СПО не появится символ с меньшим приоритетом, либо левая скобка, либо СПО станет пустым. После этого символ из входной строки заносится в СПО.
7. При достижении последнего символа входной строки содержимое СПО последовательно выталкивается в постфиксную строку.

Процесс получения обратной польской записи для выражения $(a + b) * (c + d) - e$ представлен в табл. 2.3.

Таблица 2.3. Формирование обратной польской записи для выражения (a + b) * (c + d) – e

Просматриваемый символ	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Входная строка	(	*a*	+	*b*	)	*	(	*c*	+	*d*	)	–	*e*	
Состояние стека последовательности операций	(	(	+ (	+ (		*	(*	(*	+ (*	+ (*	*	–	–	
Постфиксная строка		*a*		*b*	+			*c*		*d*	+	*	*e*	–

Таким образом, рассмотренное выше выражение в польской записи имеет вид: *ab + cd + *e –*. Данная форма записи однозначно определяет порядок загрузки операндов и операций в стек (рис. 2.5).

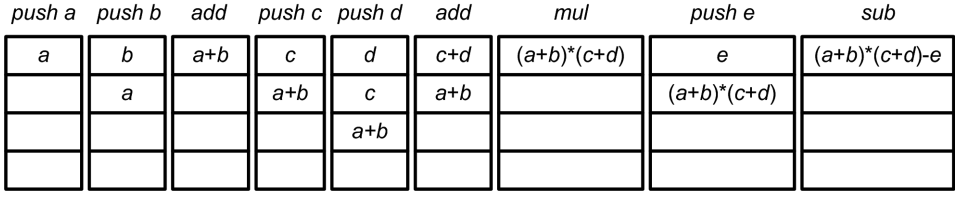

Рис. 2.5. Последовательность вычисления выражения *ab+cd+*e–* на вычислительной машине со стековой архитектурой

Основные узлы и информационные тракты одного из возможных вариантов ВМ на основе стековой АСК показаны на рис. 2.6.

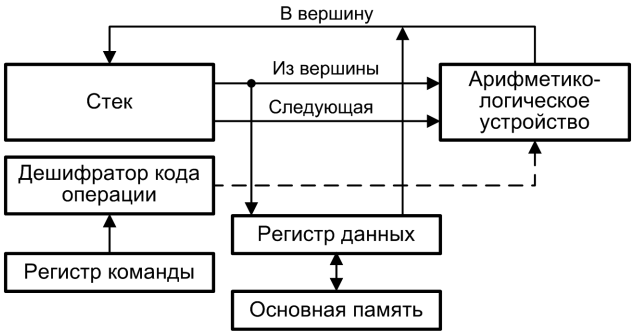

Рис. 2.6. Архитектура вычислительной машины на базе стека

Информация может быть занесена в вершину стека из памяти или из АЛУ. Для записи в стек содержимого ячейки памяти с адресом *x* выполняется команда *push x*, по которой информация считывается из ячейки памяти, заносится в регистр данных, а затем проталкивается в стек. Результат операции из АЛУ заносится в вершину стека автоматически.

Сохранение содержимого вершины стека в ячейке памяти с адресом *x* производится командой *pop x*. По этой команде содержимое верхней ячейки стека подается на шину, с которой и производится запись в ячейку *x*, после чего все содержимое стека проталкивается на одну позицию вверх.

Для выполнения арифметической или логической операции на вход АЛУ подается информация, считанная из двух верхних ячеек стека (при этом содержимое стека продвигается на две позиции вверх, то есть операнды из стека удаляются). Результат операции заталкивается в вершину стека. Возможен вариант, когда результат сразу же переписывается в память с помощью автоматически выполняемой операции *pop x*.

Верхние ячейки стековой памяти, где хранятся операнды и куда заносится результат операции, как правило, делают более быстродействующими и размещают в процессоре, в то время как остальная часть стека может располагаться в основной памяти и частично даже на магнитном диске.

К достоинствам АСК на базе стека следует отнести возможность сокращения адресной части команд, поскольку все операции производятся через вершину стека, то есть адреса операндов и результата в командах арифметической и логической обработки информации указывать не нужно. Код программы получается компактным. Достаточно просто реализуется декодирование команд.

С другой стороны, стековая АСК по определению не предполагает произвольного доступа к памяти, из-за чего компилятору трудно создать эффективный программный код, хотя создание самих компиляторов упрощается. Кроме того, стек

становится «узким местом» ВМ в плане повышения производительности. В силу упомянутых причин, данный вид АСК долгое время считался неперспективным и встречался, главным образом, в вычислительных машинах 1960-х годов.

Последние события в области вычислительной техники свидетельствуют о возрождении интереса к стековой архитектуре ВМ. Связано это с популярностью языка Java и расширением сферы применения языка Forth, семантике которых наиболее близка именно стековая архитектура. Среди современных ВМ со стековой АСК особо следует отметить стековую машину IGNITE компании Patriot Scientist, которую ее авторы считают представителем нового вида АСК — *архитектурой с безоперандным набором команд*. Для обозначения таких ВМ они предлагают аббревиатуру ROSC (Removed Operand Set Computer). ROSC-архитектура заложена и в некоторые российские проекты, например разработки ИТФ «Технофорт». Строго говоря, по своей сути ROSC мало отличается от традиционной архитектуры на базе стека и выделение ее в отдельный вид представляется не вполне обоснованным.

Аккумуляторная архитектура

Архитектура на базе аккумулятора исторически возникла одной из первых. В ней для хранения одного из операндов арифметической или логической операции в процессоре имеется выделенный регистр — *аккумулятор*. В этот же регистр заносится и результат операции. Поскольку адрес одного из операндов предопределен, в командах обработки достаточно явно указать местоположение только второго операнда. Изначально оба операнда хранятся в основной памяти, и до выполнения операции один из них нужно загрузить в аккумулятор. После выполнения команды результат заносится в аккумулятор, и если этот результат не является операндом для последующей команды, то его требуется сохранить в ячейке памяти.

Типичная архитектура ВМ на базе аккумулятора показана на рис. 2.7.

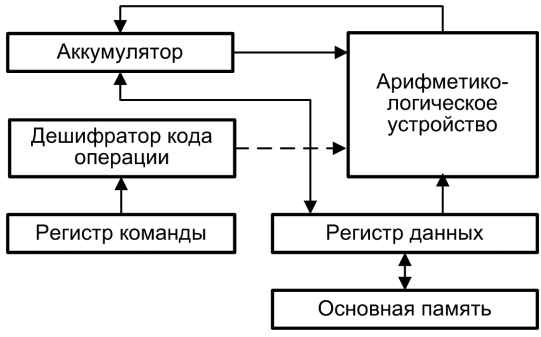

Рис. 2.7. Архитектура вычислительной машины на базе аккумулятора

Для загрузки в аккумулятор содержимого ячейки *x* предусмотрена команда загрузки *load x*. По этой команде информация считывается из ячейки памяти *x*, выход памяти подключается ко входам аккумулятора, и происходит занесение считанных данных в аккумулятор.

Запись содержимого аккумулятора в ячейку *x* осуществляется командой сохранения *store x*, при выполнении которой выходы аккумулятора подключаются к шине, после чего информация с шины записывается в память.

Для выполнения операции в АЛУ производится считывание одного из операндов из памяти в регистр данных. Второй операнд находится в аккумуляторе. Выходы регистра данных и аккумулятора подключаются к соответствующим входам АЛУ. По окончании предписанной операции результат с выхода АЛУ заносится в аккумулятор.

Достоинствами аккумуляторной АСК можно считать короткие команды и простоту декодирования команд. Однако наличие всего одного регистра порождает многократные обращения к основной памяти.

АСК на базе аккумулятора была популярна в ранних ВМ, таких, например, как IBM 7090, DEC PDP-8.

Регистровая архитектура

В машинах данного типа процессор включает в себя массив регистров общего назначения (РОН). Разрядность регистров обычно совпадает с размером машинного слова. К любому регистру можно обратиться, указав его номер. Количество РОН в архитектурах типа CISC обычно невелико (от 8 до 32), и для представления номера конкретного регистра необходимо не более пяти разрядов, благодаря чему в адресной части команд обработки допустимо одновременно указать номера двух, а зачастую и трех регистров (двух регистров операндов и регистра результата). RISC-архитектура предполагает использование существенно большего числа РОН (до нескольких сотен), однако типичная для таких ВМ длина команды (обычно 32 разряда) позволяет определить в команде до трех регистров.

Регистровая архитектура допускает расположение операндов как в регистрах, так и в основной памяти, поэтому в рамках данной АСК выделяют три формата команд обработки:

- регистр-регистр;
- регистр-память;
- память-память.

В формате «регистр-регистр» операнды могут находиться только в регистрах. В них же засылается и результат. Формат «регистр-память» предполагает, что один из операндов размещается в регистре, а второй в основной памяти. Результат обычно замещает один из операндов. В командах формата «память-память» оба операнда хранятся в основной памяти. Результат заносится в память. Каждому из форматов соответствуют свои достоинства и недостатки (табл. 2.4).

В выражениях вида (*m*, *n*), приведенных в первом столбце таблицы, *m* означает количество операндов, хранящихся в основной памяти, а *n* — общее число операндов в команде арифметической или логической обработки.

Формат «регистр-регистр» является основным в вычислительных машинах типа RISC. Команды формата «регистр-память» характерны для CISC-машин. Наконец,

формат «память-память» считается неэффективным, хотя и остается в наиболее сложных моделях машин класса CISC.

Таблица 2.4. Сравнительная оценка вариантов размещения операндов

Вариант	Достоинства	Недостатки
Регистр-регистр (0, 3)	Простота реализации, фиксированная длина команд, простая модель формирования объектного кода при компиляции программ, возможность выполнения всех команд за одинаковое количество тактов	Большая длина объектного кода, из-за фиксированной длины команд часть разрядов в коротких командах не используется
Регистр-память (1, 2)	Данные могут быть доступны без загрузки в регистры процессора, простота кодирования команд, объектный код получается достаточно компактным	Потеря одного из операндов при записи результата, длинное поле адреса памяти в коде команды сокращает место под номер регистра, что ограничивает общее число РОН. CPI зависит от места размещения операнда
Память-память (3, 3)	Компактность объектного кода, малая потребность в регистрах для хранения промежуточных данных	Разнообразие форматов команд и времени их исполнения, низкое быстродействие из-за обращения к памяти

Возможную структуру и информационные тракты вычислительной машины с регистровой архитектурой системы команд иллюстрирует рис. 2.8.

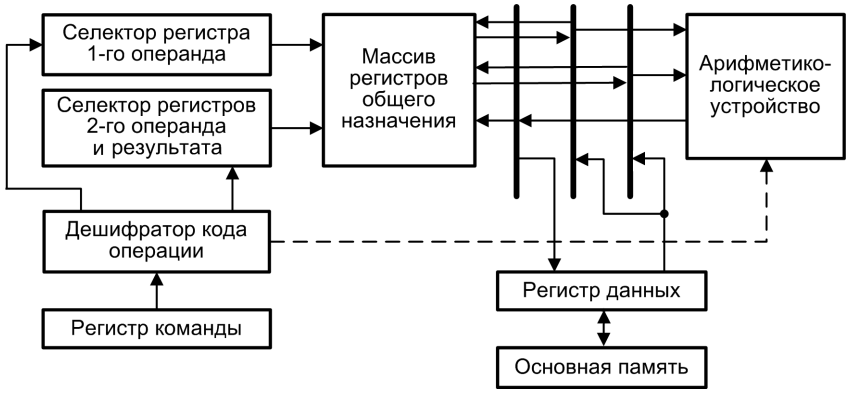

Рис. 2.8. Архитектура вычислительной машины на базе регистров общего назначения

Операции загрузки регистров из памяти и сохранения содержимого регистров в памяти идентичны таким же операциям с аккумулятором. Отличие состоит в этапе выбора нужного регистра, обеспечиваемого соответствующими селекторами.

Выполнение операции в АЛУ включает в себя:

- определение местоположения первого операнда (регистр или память);
- выбор регистра первого операнда или считывание первого операнда из памяти;

- определение местоположения второго операнда (регистр или память);
- выбор регистра второго операнда или считывание второго операнда из памяти;
- подачу на вход АЛУ операндов и выполнение операции;
- определение местоположения результата (регистр или память);
- выбор регистра результата или ячейки памяти и занесение результата операции из АЛУ.

Обратим внимание на то, что между АЛУ и регистровым файлом должны быть три шины. Две из трех шин, расположенных между массивом РОН и АЛУ, обеспечивают передачу в арифметико-логическое устройство операндов, хранящихся в регистрах общего назначения или ячейках памяти. Третья служит для занесения результата в выделенный для этого регистр или ячейку памяти. Эти же шины позволяют загрузить в регистры содержимое ячеек основной памяти и сохранить в ОП информацию, находящуюся в РОН.

К достоинствам регистровых АСК следует отнести: компактность получаемого кода, высокую скорость вычислений за счет замены обращений к основной памяти на обращения к быстрым регистрам. С другой стороны, данная архитектура требует более длинных команд по сравнению с аккумуляторной архитектурой.

В наши дни этот вид архитектуры системы команд является преобладающим, в частности такую АСК имеют современные персональные компьютеры.

Архитектура с выделенным доступом к памяти

В архитектуре с выделенным доступом к памяти обращение к основной памяти возможно только с помощью двух специальных команд: *load* и *store*. В английской транскрипции данную архитектуру называют *Load/Store architecture*. Команда *load* (загрузка) обеспечивает считывание значения из основной памяти и занесение его в регистр процессора (в команде обычно указывается адрес ячейки памяти и номер регистра). Пересылка информации в противоположном направлении производится командой *store* (сохранение). Операнды во всех командах обработки информации могут находиться только в регистрах процессора (чаще всего в регистрах общего назначения). Результат операции также заносится в регистр.

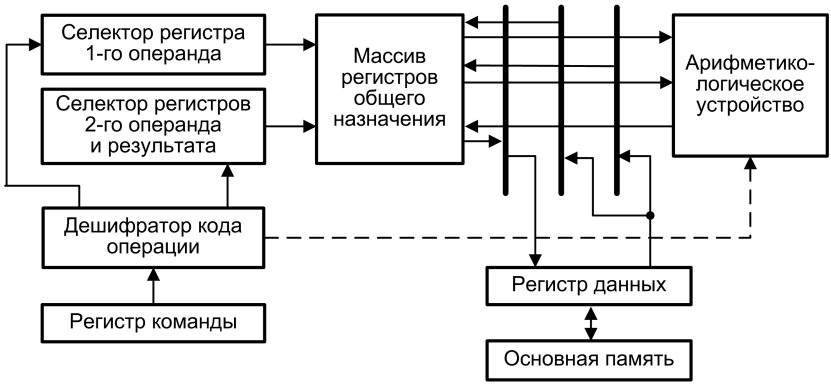

Рис. 2.9. Архитектура вычислительной машины с выделенным доступом к памяти