Московский государственный университет
им. М.В. Ломоносова

Факультет вычислительной математики и
кибернетики

Е.А. Бордаченкова

Модельные ЭВМ

Москва

2012

УДК 004.2(075.8)
ББК 32.973-02я73
Б82 …

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

Рецензенты:
В.Г. Баула – доцент кафедры АСВК ВМК МГУ
Е.А. Кузьменкова – доцент кафедры СП ВМК МГУ;

Бордаченкова Е.А.

Б82 Модельные ЭВМ: Учебное пособие для студентов 1 курса. – М.: Издательский отдел факультета ВМиК МГУ имени М.В.Ломоносова (лицензия ИД №05899 от 24.09.2001 г.); МАКС Пресс, 2012. – 60 с.
ISBN 978-5-89407-501-3
ISBN 978-5-317-04524-1

Учебное пособие предназначено для студентов первого курса факультета ВМК МГУ имени М.В. Ломоносова, изучающих курс "Архитектура ЭВМ и язык ассемблера".

Ключевые слова: ЭВМ, Фон-Неймановская архитектура, центральный процессор (ЦПУ), устройство управления, арифметико-логическое устройство, счётчик адреса, регистр команды, оперативная память, адрес, ячейка, регистр, стек, индексный регистр, машинная операция, система команд, машинный язык, машинная команда, код операции, операнды, 16-ричная запись.

УДК 004.2(075.8)
ББК 32.973-02я73

Bordachenkova E.A. Computer Architectures in Simplified Models.

A textbook for first-year students of Moscow State University Computational Mathematics and Cybernetics faculty attended the course "Computer architecture and assembly language".

Keywords: computer, Von Neumann computer, processor, central processing unit (CPU), the control unit, the arithmetic & logic unit (ALU), program counter (PC), instruction register (IR), memory, address, cell, register, stack, index register, machine instruction, instruction set, machine language, operator (opcode), operands, HEX notation.

ISBN 978-5-89407-501-3

Введение

С момента своего возникновения до настоящего времени электронные вычислительные машины прошли большой путь развития. В 40-50 годы XX века вычислительные машины представляли собой гигантские научно-инженерные сооружения. "Машина весила около 30 т, имела примерно 2,4 м в высоту, 30,5 м в длину и 0,9 м в глубину и занимала зал площадью 167 кв. м. Она … потребляла около 160 кВт электроэнергии – мощность, достаточная для работы небольшого завода", – вот характеристики машины ENIAC [1]. Каждая машина была уникальна – строился один экземпляр. С развитием электронной техники и накоплением опыта в создании ЭВМ вычислительные машины становились компактнее, надёжней и дешевле, появилась возможность серийного производства компьютеров. Выделилось несколько направлений развития вычислительной техники: небольшие вычислительные машины для инженерных и коммерческих задач, мощные суперкомпьютеры для научных расчётов, специализированные системы для работы в режиме реального времени.

С появлением микросхем стали возникать всё новые и новые классы вычислительных машин. Огромное разнообразие существующих сейчас компьютеров поражает: быстродействующие многопроцессорные суперкомпьютеры, занимающие специальные залы с контролируемым микроклиматом; мощные графические рабочие станции, применяемые в киноиндустрии; широко распространённые ноутбуки и планшетные компьютеры, электронные книги, смартфоны; встраиваемые системы, которые используются в пылесосах, стиральных машинах и кухонных плитах, управляют видеокамерами, фотоаппаратами и телевизорами, находятся в пластиковых и транспортных картах, и даже в защитных наклейках, которые пищат, если вам плохо размагнитили книгу в магазине.

Несмотря на разнообразие видов электронных вычислительных устройств, их работа базируется на общих принципах. Изучение этих принципов и будет нашей задачей, а именно, мы будем изучать различные архитектуры вычислительных машин. Под ***архитектурой ЭВМ*** будем понимать набор устройств, из которых состоит ЭВМ, назначение и особенности функционирования этих устройств и взаимосвязи между ними.

Принципы организации и работы ЭВМ удобно изучать, рассматривая модели, а не реальные компьютеры. Дело в том, что на конструкцию любой реальной вычислительной машины помимо основных проектных идей, влияло множество факторов, даже случайностей, усложняющих и запутывающих устройство машины. А модель может демонстрировать идею в чистом виде.

Будем использовать термины "электронная вычислительная машина", "ЭВМ", "вычислительная машина" и "компьютер" как синонимы.

На развитие вычислительной техники большое влияние оказала работа Джона фон Неймана "Набросок отчёта о машине EDVAC" ("First Draft of a Report on the EDVAC"). Проанализировав опыт построения электронной машины EDVAC, в 1948 г. Джон фон Нейман опубликовал проект логической организации электронной вычислительной машины. Высказанные им идеи получили название "принципов фон Неймана". Все существующие компьютеры в той или иной мере соответствуют этим принципам; фактически, принципы фон Неймана стали точкой отсчёта при анализе и сравнении различных компьютеров. Мы будем рассматривать машины, отвечающие принципам фон Неймана, машины так называемой фон-Неймановской архитектуры.

§1. Структура ЭВМ

В параграфе рассмотрены общие принципы устройства и функционирования вычислительных машин. Описано, из каких элементов состоит компьютер, назначение этих элементов. Обсуждается алгоритм работы компьютера, процесс выполнения компьютером программы.

П1. Основные компоненты ЭВМ

Основными элементами любой вычислительной машины являются центральный процессор, оперативная память и внешние устройства. Схематично компьютер изображён на рисунке 1.

Центральный процессор (ЦП) выполняет программу пользователя и управляет всеми устройствами ЭВМ. *Оперативная память* (ОП) во время работы компьютера хранит данные и выполняемую программу. *Внешние устройства* служат для связи вычислительной машины с внешним миром: для ввода и вывода данных, для долговременного хранения данных и программ.

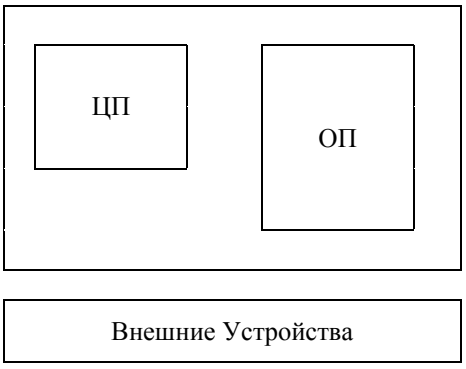


Рис.1. Схема ЭВМ

При обсуждении различных архитектур ЭВМ будем подробно рассматривать главные компоненты компьютера – процессор и оперативную память. Многообразие внешних устройств и организацию взаимодействия между устройствами ЭВМ оставим за рамками нашего рассмотрения.

Для начала обсудим назначение элементов компьютера и их внутреннюю организацию.

1. Центральный процессор

Центральный процессор – основная компонента вычислительной машины. Именно процессор выполняет программу пользователя. Кроме этого, процессор организует взаимодействие между всеми частями вычислительной машины, так что они образуют единое целое – компьютер.

Элементами центрального процессора являются устройства и регистры. Регистры – это запоминающие элементы (ячейки), расположенные внутри процессора; они предназначены для хранения информации. В настоящем параграфе нас будут интересовать следующие элементы: устройство управления (УУ), арифметико-логическое устройство (АЛУ), регистр счётчик адреса (СА) и регистр команды (РК) (см. рис. 2).

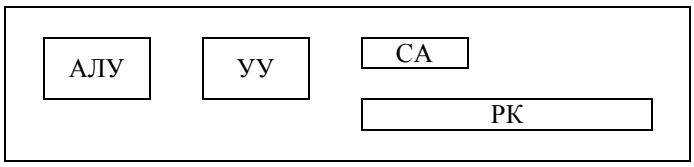


Рис. 2. Схема центрального процессора

Устройство управления управляет работой процессора в целом, координирует работу всех остальных устройств, организует процесс выполнения программы. *Арифметико-логическое устройство* выполняет арифметические и логическое операции, собственно, производит вычисления. Регистр *счётчик адреса* содержит адрес очередной команды программы. *Регистр команды* содержит команду, которая выполняется процессором в текущий момент времени.

2. Оперативная память

Оперативная память хранит программу, которую выполняет вычислительная машина, и данные для программы. Оперативная память может хранить данные только во время работы машины. Когда машина выключается, содержимое оперативной памяти теряется.

Оперативная память состоит из элементов – ячеек. Количество ячеек называется ***объёмом оперативной*** памяти. Ячейки перенумерованы, номер ячейки называется ***адресом ячейки***. Схематически оперативная память изображена на рисунке 3.

Ячейка состоит из разрядов. Количество разрядов во всех ячейках памяти одинаково, это число называется ***разрядностью*** оперативной памяти. Каждый разряд может хранить 0 или 1, т.е. одну двоичную цифру. Содержимое ячейки (число или команда) называется ***машинным словом***.

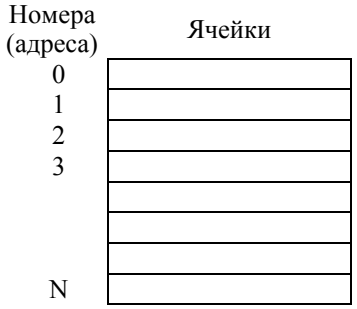


Рис. 3. Оперативная память

Основные операции, выполняемые с оперативной памятью – записать в ячейку число или команду (из процессора или с внешнего устройства) и переслать содержимое ячейки из памяти в процессор (или на внешнее устройство). Доступ к ячейке осуществляется по её адресу, номеру. Работать с частью ячейки нельзя, поэтому часто понятие ячейка определяют так: ***ячейка*** – это минимальная адресуемая единица памяти.

В любой момент времени можно обратиться к любой ячейке, независимо от того, с какой ячейкой работали раньше. Время доступа к ячейке не зависит от расположения ячейки в памяти. Оба эти обстоятельства имеют в виду, говоря, что оперативная память – устройство прямого (или произвольного) доступа.

Следует иметь в виду, что оперативная память намного более медленное устройство, чем центральный процессор.

3. Внешние устройства

Внешние устройства предназначены для связи вычислительной машины с внешним миром (записи программы в оперативную память, ввода данных и вывода результата счёта) и для долговременного хранения программ и данных. В соответствии со своим назначением внешние устройства делятся на два класса: устройства ввода-вывода и внешние запоминающие устройства.

К устройствам ввода-вывода относятся клавиатура, монитор, мышь, принтер, сканер и т.п. Устройства ввода-вывода работают намного медленнее процессора и оперативной памяти. Примерами внешних запоминающих устройств являются жесткий диск (винчестер), флэш-накопитель, магнитофон (стример). По сравнению с оперативной памятью внешняя память имеет намного больший объём, работает существенно медленнее. Время доступа к данным может зависеть от их расположения на носителе.

Важным фактором является то, что существует большое разнообразие внешних устройств, каждое из них обладает своими физическими характеристиками и особенностями функционирования, которые требуется

учитывать при организации взаимодействия вычислительной машины и внешних устройств.

П2. Система команд

Каждый процессор имеет набор встроенных, так называемых машинных, операций. ***Машинная операция*** – это элементарное действие, выполняемое аппаратным путём.[2] Машинные операции реализованы в виде электронных схем в процессоре. Примеры машинных операций: пересылка содержимого ячейки из оперативной памяти в процессор, сложение, вычитание. Любое вычисление, которое нужно произвести на вычислительной машине, необходимо разложить на элементарные действия, представить в виде последовательности машинных операций.

Машинные операции могут быть довольно сложными, такими, как извлечение корня или вычисление значения функции в точке. Конкретный набор машинных операций процессора зависит от специфики класса задач, для решения которых предполагается использовать вычислительную машину, и от экономических факторов. Действия, часто встречающиеся при решении задач, целесообразно реализовывать аппаратно, редкие операции можно реализовать программно, выразив их через машинные операции. (Например, умножение можно реализовать сложением.) Чем шире набор машинных операций, тем, вообще говоря, проще программировать для процессора, тем процессор сложнее и дороже. Чем уже набор машинных операций, тем проще процессор.

Приказ выполнить машинную операцию для указанных данных называется ***машинной командой***. Правила записи машинной команды называется ***форматом команды***. Список машинных операций вместе с форматами команд называется ***системой команд*** процессора. Машинная программа представляет собой алгоритм решения задачи, реализованный в виде последовательности машинных команд.

Стоит заметить, что часто слово "команда" используют как синоним словосочетания "машинная операция". Однако эта многозначность слова "команда" путаницы не вызывает, так как из контекста всегда понятно, о чём идёт речь – об аппаратно реализованной операции ("в процессоре имеется команда сложения") или о выполнении действия с конкретными данными.

Как правило, в набор машинных операций входят следующие группы команд.

– *Пересылки*: предназначены для обмена данными между процессором и оперативной памятью.

– *Арифметические команды*: сложение, вычитание, умножение, деление и другие операции.

– *Переходы*: команды передачи управления, позволяющие программировать условные операторы и циклы.

П3. Схема работы процессора

Рассмотрим, как функционирует вычислительная машина. Поскольку операции ввода-вывода не входят в тему обсуждения, договоримся считать, что в начале работы вычислительной машины в её оперативную память каким-то образом загружена программа. Исходные данные находятся в ячейках памяти с известными адресами. Результаты также нужно записать в известные ячейки оперативной памяти.

Работа вычислительной машины состоит из шагов – тактов. ***Тактом работы процессора*** называется выполнение одной машинной команды.

Работа происходит следующим образом.

При включении компьютера прежде всего в регистр счётчик адреса (СА) записывается некоторое число. Это действие выполняется аппаратно, независимо от программиста, причём при каждом запуске процессора в СА записывается одно и то же число. Будем считать, что это число 0.

После загрузки начального значения в регистр СА, процессор начинает последовательно, шаг за шагом, выполнять команды программы, пока не встретит команду останова.

На каждом такте процессор выполняет действия:

1. В регистр команды (РК) записывается содержимое ячейки, адрес которой находится в регистре СА.
2. Значение СА увеличивается на 1, так что теперь СА указывает на следующую команду программы.
3. Устройство управления расшифровывает команду, находящуюся в РК, и организует её выполнение. Как именно выполняется команда, зависит от вида самой команды. (При анализе конкретной системы команд мы рассмотрим более подробно процесс выполнения команды.)

Далее работа повторяется с первого шага.

Процесс заканчивается, когда выполнилась команда остановки процессора.

Заметим, что, поскольку при включении машины в СА записывается число 0 и далее именно содержимое ячейки с адресом 0 помещается в ЦП, расшифровывается и выполняется, в нулевой ячейке должна находиться первая исполняемая команда программы. Итак, все программы должны начинаться по нулевому адресу, то есть с ячейки с адресом 0.

Подведём итог: регистр счётчик адреса хранит адрес команды, которая будет выполняться на следующем шаге работы процессора; регистр команды содержит команду, выполняемую на текущем шаге.

Далее займёмся обсуждением различных архитектур вычислительных машин на примере моделей – учебных машин. Договоримся при записи машинных программ для сокращения текста и повышения наглядности вместо выписывания последовательности двоичных цифр (содержимого ячейки), записывать шестнадцатиричные цифры. Каждая шестнадцатиричная цифра изображает четыре двоичные цифры.

Начнём обсуждение с рассмотрения трёхадресной модели, наиболее удобной с точки зрения программирования вычислений и обладающей самой логичной организацией функционирования.

§2. Трёхадресная учебная машина (УМ-3)

П1. Устройство трёхадресной учебной машины

1. Процессор УМ-3

Общая схема центрального процессора была разобрана в §1. Рассмотрим подробнее часть процессора, отвечающую за выполнение арифметических операций, представленную на рисунке 4.

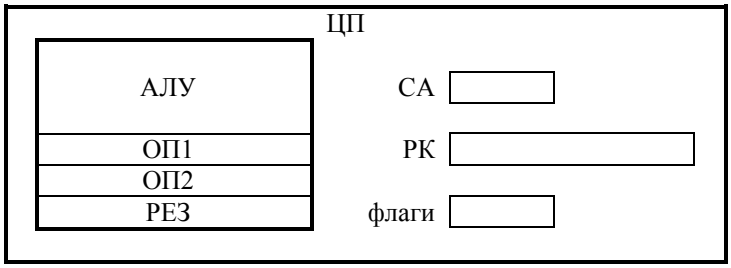


Рис. 4. Схема процессора УМ-3

С арифметико-логическим устройством связаны регистры первого и второго операндов (ОП1 и ОП2) и регистр результата (РЕЗ). Регистр результата также называют *сумматором*. При выполнении арифметической операции в регистры ОП1 и ОП2 записываются первый и второй операнды соответственно, а в регистр РЕЗ арифметико-логическое устройство помещает полученный результат.

2. Оперативная память.

Оперативная память машины УМ-3 состоит из 65536 (2^{16}, или 16^4) ячеек, которые имеют адреса от 0000_{16} до $FFFF_{16}$. Каждая ячейка состоит из 14 шестнадцатиричных разрядов. [1]

Ячейка может содержать число или команду.

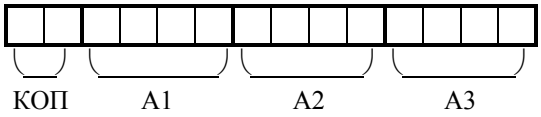
[1] Напомним, что для сокращения записи используем шестнадцатиричную систему вместо двоичной. Читателю предоставляется возможность самостоятельно определить размер ячейки в двоичных разрядах.

В машине УМ-3 используются числа без знака и числа со знаком. Числа со знаком представляются в дополнительном коде. Приведём примеры чисел в УМ-3:

числу 19 соответствует машинное слово 00 0000 0000 0013;

числу -19 соответствует слово FF FFFF FFFF FFED.

Если ячейка содержит команду, разряды ячейки группируются следующим образом:



Здесь КОП – код операции, он указывает, какую машинную операцию надо выполнить; A1, A2, A3 – адреса первого, второго и третьего операндов. Разрядность поля адреса определяется объёмом оперативной памяти.

3. Система команд

Команды машины УМ-3 можно разделить на следующие группы: арифметические команды, переходы, команды пересылки и останова. Рассмотрим команды этих групп.

Арифметические команды. В эту группу входят команды сложения, вычитания, умножения и деления. Рассмотрим сначала команду сложения. Сложение чисел без знака и сложение чисел со знаком выполняются по единому алгоритму, так как числа со знаком представляются в дополнительном коде. В АЛУ процессора есть одна электронная схема, реализующая этот алгоритм, и в УМ-3 существует одна машинная команда сложения, работающая как с числами со знаком, так и с числами без знака. В результате выполнения операции вычисляется сумма и формируются флаги. По значениям флагов можно отследить особые ситуации, например, переполнение при сложении чисел без знака (по флагу CF) и переполнение при работе с числами со знаком (по значению флага OF). Вычитание аналогично сложению: один и тот же алгоритм подходит для обработки чисел без знака и для обработки чисел со знаком, следовательно, требуется одна машинная команда вычитания.

С умножением ситуация сложнее. Числа без знака и числа со знаком обрабатываются по разным алгоритмам, поэтому в АЛУ есть две электронные схемы, реализующие эти алгоритмы, следовательно, существует две машинные команды умножения – умножение чисел без знака и умножение чисел со знаком. То же с делением. Есть машинная команда деления чисел без знака и машинная команда деления чисел со знаком.

Деление имеет ещё одну особенность: одна команда деления вырабатывает два результата, частное и остаток.[2]

Команды имеют следующие коды

Операция	**КОП**	**Операция**	**КОП**
сложение	01	умножение (без знака)	13
вычитание	02	деление (со знаком)	04
умножение (со знаком)	03	деление (без знака)	14

Рассмотрим арифметическую команду КОП A1 A2 A3. При выполнении этой команды процессор должен выполнить следующую последовательность шагов:

1. загрузить содержимое ячейки оперативной памяти с адресом A1 в регистр ОП1,
2. загрузить содержимое ячейки оперативной памяти с адресом A2 в регистр ОП2,
3. запустить в АЛУ электронную схему, реализующую операцию, задаваемую КОП,
4. записать результат из регистра РЕЗ в ячейку оперативной памяти с адресом A3.

Если выполняется операция деления, в оперативную память записываются два результата: частное – в ячейку с адресом A3, остаток – в следующую ячейку, по адресу (A3+1) mod 16^4.

Команды перехода. Действие команд перехода заключается в изменении содержимого регистра счётчик адреса (СА). Таким образом, на следующем такте работы процессора будет выполнена не та команда, которая бы выполнилась при нормальном порядке выполнении команд, а другая, чей адрес теперь записан в СА. В машине УМ-3 есть безусловный и условные переходы.

Код операции ***безусловного перехода*** 80, формат команды 80 A1 A2 A3, здесь A3 – адрес перехода, т.е. адрес команды, на которую нужно передать управление; операнды A1 и A2 не используются. Действие команды: запись в регистр СА адреса A3. Подчеркнём, что в машинной команде нужно обязательно указывать адреса первого и второго операндов, несмотря на то, что они не используются. Дело в том, что все разряды ячейки должны быть заполнены. Записывая команду, мы выписываем содержимое разрядов ячейки памяти. А разряд - это физическое устройство, которое обязательно находится в некотором состоянии (0 или 1), разряд не может быть пустым.

[2] Применяя школьный алгоритм деления в столбик, вы так же получите частное и остаток за одно деление.

Рассмотрим теперь команду ***условного перехода*** КОП A1 A2 A3. Код операции КОП задаёт условие перехода, адреса A1 и A2 определяют сравниваемые операнды, A3 – адрес перехода. Ниже приведён список кодов операций.

Условие перехода	**КОП**	**Условие перехода**	**КОП**
=	81	≠	82
числа со знаком		числа без знака	
<	83	<	93
≥	84	≥	94
≤	85	≤	95
>	86	>	96

Все команды условного перехода выполняются так: проверяется, удовлетворяют ли числа, находящиеся в ячейках памяти по адресам A1 и A2, условию перехода. Если условие оказывается истинным, в регистр СА записывается число A3, в противном случае содержимое СА не меняется. Обратим внимание, что для всех условий, кроме "равно" и "не равно" есть пара команд перехода – для чисел со знаком и для чисел без знака. Дело в том, что проверка условий для чисел со знаком и для чисел без знака выполняется по-разному. Выбор, какой вариант команды использовать в программе, зависит от того, с какими данными работает программа.

Команда пересылки. Формат команды: 00 A1 A2 A3. Действие команды заключается в пересылке содержимого ячейки памяти с адресом A1 в ячейку с адресом A3. Адрес A2 несущественен, однако должен быть указан в команде.

Команда останова. Формат команды 99 A1 A2 A3. По этой команде процессор завершает выполнение программы, останавливается.

Напомним, что при включении процессора в регистр СА автоматически (это делает аппаратура) записывается некоторое число, всегда одно и то же для конкретной ЭВМ. Затем начинает выполняться последовательность команд, начиная с команды, адрес которой находится в СА. Договоримся считать, что в машине УМ-3 при включении в регистр СА записывается число ноль. Это значит, что первая команда программы должна быть записана в ячейку с адресом 0000.

П2. Приёмы программирования для УМ-3

Машина УМ-3 работает следующим образом. Перед началом работы УМ-3 в оперативную память загружается программа и необходимые константы. Команды программы и константы образуют один блок подряд

расположенных ячеек ОП. Затем в ячейки, отведённые программистом для переменных, вводятся данные (команды ввода-вывода мы опускаем из рассмотрения). Наконец, включается ЦП и начинается выполнение программы. Результаты работы записываются в специально отведённые программистом ячейки.

1. Вычисление по формулам

Напишем программу для решения задачи: по известным a, b вычислить значение $x = (a*(-21) \bmod 50 - b)^2$.

Прежде всего решим, с какими числами будет работать программа. Поскольку в выражении встречается отрицательное число и есть операция вычитания, будем считать, что числа со знаком. Следовательно, константы будем представлять как числа со знаком и используем машинные операции для знаковых чисел.

Затем нужно составить алгоритм вычислений. Команды машины УМ-3 довольно простые, каждая команда может выполнить только одну арифметическую операцию, поэтому алгоритм будет состоять из мелких действий.

Алгоритм.

$x := a * (-21)$;
$x := x \bmod 50$;
$x := x - b$;
$x := x * x$

Теперь нужно заняться распределением памяти. В алгоритме используются три переменные: a и b – исходные данные и x – результат. Пусть переменные располагаются в ячейках со следующими адресами:

a	—	0100
b	—	0101
рабочая (для div)		0102
x	—	0103

Машинная программа выглядит как последовательность шестнадцатиричных чисел, т.е. как содержимое подряд расположенных ячеек памяти. Однако мы будем записывать программу в виде таблицы из трёх колонок: адрес ячейки, содержимое ячейки, комментарий. Собственно программой является средняя колонка, колонки "адрес" и "комментарий" служат пояснениями. Колонка "адрес" помогает отслеживать адреса ячеек. Подсчёт адресов команд и констант чрезвычайно важное дело, требующее аккуратности. Ошибки в указании адресов операндов команд трудно обнаруживать. Символ "$\rightarrow$" в комментариях имеет смысл "вычислить выражение в левой части и записать полученное значение в правую часть".

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	03 0100 <u>0005</u> 0103	a*(-21) $\rightarrow x$
0001	04 0103 <u>0006</u> 0102	x div 50 $\rightarrow$ [0102], x mod 50 $\rightarrow x$
0002	02 0103 0101 0103	$x - b \rightarrow x$
0003	03 0103 0103 0103	$x*x \rightarrow x$
0004	99 0000 0000 0000	стоп
0005	FF FFFF FFFF FFEB	$(-21)_{10}$
0006	00 0000 0000 0032	50_{10}

Поле адреса второго операнда в первых двух командах подчёркнуты. При написании команд эти поля остаются незаполненными, потому что адреса ячеек, где будут храниться константы, неизвестны до тех пор, пока программа не будет написана целиком. Тогда определится адрес последней команды программы и станут известны адреса констант. Заполним ячейки машинным представлением чисел -21 и 50 и впишем соответствующие адреса в команды.

Напомним, что деление даёт два результата, поэтому команда по адресу 0001 записывает в ячейку с адресом 0102 значение x div 50 (это значение в дальнейшем не используется), а в ячейку 0103 (переменную x) – нужное нам значение a mod 50. Запись [0102] в комментарии обозначает ячейку с адресом 0102.

Далее мы рассмотрим, как в машинных программах реализуются основные управляющие конструкции языков программирования: условные операторы и циклы.

2. Программирование условных операторов

Для программирования условных операторов используются команды условного и безусловного переходов. Схема реализации полного условного оператора выглядит следующим образом.

Язык высокого уровня	**Машинная программа**
if β	вычисление β
	if not β goto **Else**
then S1	then-часть S1
	goto **End**
else S2	**Else**: S2
* * *	**End**: * * *

Как видно из схемы, машинная программа состоит из трёх частей, расположенных последовательно. Части соответствуют конструкции условного оператора: вычисление условия β, then-часть и else-часть. Между вычислением условия и then-частью стоит команда передачи управления на else-часть. Подчеркнём, что переход на else-часть должен произойти при нарушении условия β, поэтому необходимо использовать команду перехода по отрицанию условия β. Отметим ещё один тонкий момент. Семантика полного условного оператора такова, что всегда выполняется только одна из ветвей оператора; после завершения выполнения then-части, начинает выполняться оператор, следующий за условным оператором. В машинной программе за командами, реализующими then-часть, обязательно должна быть команда безусловного перехода на конец фрагмента для того, чтобы избежать случайного выполнения else-части.

Рассмотрим, как приведённая схема реализуется в решении следующей задачи: по известным значениям *a* и *b* требуется вычислить значения *S1* = *max*(*a,b*) * 20, *S2* = *min*(*a,b*) *div* 3.

Будем считать, что работаем с числами без знака. Вычисления организуем так:

```
begin  if a < b then begin S1 := b; S2 := a end
                     else  begin S1 := a; S2 := b end;
       S1 := S1 * 20;
       S2 := S2 div 3
end.
```

Пусть переменные находятся в ячейках с адресами:

a — 0100
b — 0101
S1 — 0102
S2 — 0103

Реализация алгоритма для УМ-3:

Адрес	Содержимое ячейки	Комментарий
0000	94 0100 0101 0004	if $a \geq b$, goto ***else***
0001	00 0101 0000 0102	$b \rightarrow S1$
0002	00 0100 0000 0103	$a \rightarrow S2$
0003	80 0000 0000 0006	goto ***endif***
0004	00 0100 0000 0102	***else***: $a \rightarrow S1$
0005	00 0101 0000 0103	$b \rightarrow S2$
0006	13 0102 0009 0102	***endif***: *S1* * 20 $\rightarrow$ *S1*
0007	14 0103 000A 0103	*S2 div* 3$\rightarrow$ *S2*
0008	99 0000 0000 0000	стоп
0009	00 0000 0000 0014	20_{10}

000A | 00 0000 0000 0003 | 3_{10}

Подчёркнутые поля заполняются по мере определения соответствующих адресов.

3. Программирование циклов

Рассмотрим реализацию на машинном языке цикла с предусловием и цикла с постусловием. Начнём с цикла с предусловием как более универсального. Фрагмент машинной программы, реализующий цикл с предусловием, состоит из следующих групп команд: последовательность команд, вычисляющих предусловие; команда перехода по отрицанию условия на конец цикла; последовательность команд, реализующих тело цикла; команда безусловного перехода на вычисление предусловия. Структура машинной реализации изображена на схеме:

Язык высокого уровня	**Машинная программа**
while β ***do***	***Wh***: вычисление β
	if not β goto ***End***
{тело цикла} S	S
{конец цикла}	goto ***Wh***
* * *	***End***: * * *

Обратим внимание на то, что в команде условного перехода, следующей за вычислением предусловия цикла, осуществляется переход по отрицанию предусловия (с целью обойти выполнение команд тела цикла).

Рассмотрим теперь цикл с постусловием. Соответствующий циклу фрагмент на машинном языке состоит из следующих команд: последовательность команд, реализующих тело цикла; последовательность команд, вычисляющих постусловие; команда перехода по отрицанию постусловия на начало цикла:

Язык высокого уровня	**Машинная программа**
repeat	***Rpt***:
S	S
until β	вычисление β
	if not β goto ***Rpt***
* * *	* * *

Реализация цикла с постусловием на машинном языке на одну команду перехода короче, чем реализация цикла с предусловием, поэтому, если задача позволяет, имеет смысл использовать в алгоритме именно цикл с постусловием.

Рассмотрим теперь два примера реализации циклов.

Пример 1. Написать программу решения следующей задачи. Если натуральное *n* является степенью числа 3, записать в переменную *k* показатель степени, в противном случае записать в *k* число -1.

Алгоритм:

k:= 0*;*
while *n* mod 3 = 0 ***do*** ***begin*** *k:*= *k*+1*; n:*= *n* div 3 ***end;***
if $n \neq 1$ ***then*** *k*:= -1

Исходя из условия задачи, естественно представить *n* как число без знака. Обратим внимание на следующую деталь: при вычислении предусловия требуется получить остаток от деления *n* на 3. Но команда деления вычислит, кроме остатка, ещё и частное, которое можно использовать в теле цикла. Учтём это при распределении ячеек памяти:

n – 0100
k – 0101
частное – 0102
остаток – 0103

Программа:

Адрес	Содержимое ячейки	Комментарий
0000	00 0009 0000 0101	$0 \to k$
0001	14 0100 000A 0102	***Wh***: *n* mod 3 $\to$ [0103]
0002	82 0103 0009 0007	if *n* mod 3≠0, goto ***endWh***
0003	01 0101 000B 0101	$k + 1 \to k$
0005	00 0102 0000 0100	*n* div 3$\to$ *n*
0006	80 0000 0000 0001	goto ***Wh***
0007	81 0100 000B 0008	***endWh***: if *n* =1, goto ***end***
0008	00 000C 0000 0101	$-1 \to k$
0008	99 0000 0000 0000	***end***: стоп
0009	00 0000 0000 0000	0_{10}
000A	00 0000 0000 0003	3_{10}
000B	00 0000 0000 0001	1_{10}
000C	FF FFFF FFFF FFFF	$(-1)_{10}$

В программе использовано следующее обозначение: запись [0103] заменяет словосочетание "ячейка с адресом 0103".

Заметим, что деление в команде 0001 получает частное и остаток; остаток используется для проверки предусловия цикла в команде 0002, частное используется в теле цикла, в команде 0005, без повторного выполнения деления. Для хранения результатов деления использованы две вспомогательные рабочие ячейки.

Пример 2. Написать программу решения следующей задачи. Дано целое неотрицательное число *n*. Записать в переменную *md* наименьшую из значащих цифр десятичной записи числа *n*.

Поскольку в любом числе есть хотя бы одна значащая цифра, для решения задачи использован цикл с постусловием. В качестве начального значения для минимума взято число 10, превосходящее любую цифру в десятичной системе.

Будем работать с числами без знака.

Алгоритм.

md:= 10;
repeat *d*:= *n* mod 10;
    if *d* < *md* ***then*** *md*:= *d*;
    n:= *n* div 10
until *n* = 0

Распределение памяти. Разместим сначала основные переменные – данное *n* и ответ *md*, затем вспомогательные переменные, нужные для решения задачи:

n – 0100
md – 0101
n1 – 0102
d – 0103

Программа:

Адрес	Содержимое ячейки	Комментарий
0000	00 <u>0007</u> 0000 0101	10 → *md*
0001	00 0100 0000 0102	*n* → *n1*
0002	14 0102 <u>0007</u> 0102	***Rpt***: *n1* mod 10 → *d, n1* div 10 → *n1*
0003	94 0103 0101 <u>0005</u>	if *d* ≥ *md*, goto ***Endif***
0004	00 0103 0000 0101	*d* → *md*
0005	82 0102 <u>0008</u> 0002	***Endif*** **:** if *n1* ≠ 0, goto ***Rpt***
0006	99 0000 0000 0000	стоп
0007	00 0000 0000 000A	10_{10}
0008	00 0000 0000 0000	0_{10}

Подчёркнутые поля команд были заполнены после того, как стали известны соответствующие адреса.

П3. Заключение

Мы начали рассмотрение архитектур ЭВМ с трёхадресной архитектуры, поскольку она наиболее логически ясная. Алгоритм работы процессора такой архитектуры интуитивно понятен. В процессоре легко реализуется последовательное выполнение команд – поскольку одна команда занимает одну ячейку памяти, для перехода к очередной команде нужно увеличить значение регистра счётчика адреса на единицу. Выполнение отдельной команды также логически прозрачно – требуется записать в процессор два операнда, выполнить операцию, записать полученный результат в память.

Трёхадресная архитектура имеет достоинства не только с точки зрения реализации, она наилучшим образом отвечает организации вычислений. В самом деле, любое элементарное математическое действие (например, сложение) имеет два операнда и результат; по сути, математическое действие является трёхадресным. Программировать, оперируя привычными понятиями, удобно, логика программ соответствует логике вычислений. Поэтому применение трёхадресных команд упрощает процесс программирования, способствует уменьшению количества ошибок, то есть делает программирование более надёжным.

Многие успешные отечественные вычислительные машины имели трёхадресную архитектуру.

В качестве примера можно назвать машину М-2, построенную в 1953 г. под руководством И. С. Брука. В создании программного обеспечения М-2 принимал участие Г. М. Адельсон-Вельский (известный читателям как один из авторов АВЛ-деревьев). На машине М-2 проводились расчеты в области теоретической и экспериментальной физики, атомной энергетики, теплотехники, авиации и артиллерии, был произведён расчет прочности плотин Куйбышевской и Волжской гидроэлектростанций. Машина победила в первом международном матче шахматных программ (авторы программы А. С. Кронрод, В. Л. Арлазаров) [3].

Приведём ещё примеры трёхадресных машин: БЭСМ-1 (1953 г., главный конструктор С. А. Лебедев; являлась самой быстродействующей машиной в Европе и одной из самых быстродействующих ЭВМ в мире); «Стрела» (1953 г., главный конструктор Ю. Я. Базилевский; первая серийная ЭВМ в Советском Союзе); М-20 (1958 г., главный конструктор С. А. Лебедев; заместители главного конструктора – М. К. Сулим и М. Р. Шура-Бура; родоначальница серии ЭВМ: М-20, М-220, М-222с).

В современных процессорах также используются трёхадресные команды, как правило, это арифметические и логические команды. В качестве примеров можно привести процессор Ultra SPARC II компании SUN [4] и серию процессоров ARM, применяемых во встроенных системах и мобильных телефонах [5].

§3. Двухадресная учебная машина (УМ-2)

Если посмотреть внимательно на команды машинных программ, разобранных в предыдущем параграфе, можно обратить внимание, что далеко не во всех командах все три адреса операндов различны. Оказывается, это типичная ситуация – при анализе реальных программ было установлено, что только в 25% команд существенно используются три адреса. В остальных же 75% команд либо два адреса совпадают, либо некоторые из адресов не используются. Таким образом, большинство трёхадресных команд содержат избыточную информацию, т.е. память расходуется не экономно. Попробуем избавиться от этого недостатка, убрав один адрес из команды. Мы приходим к идее двухадресной архитектуры.

П1. Удаление одного адреса из машинных команд

Обратимся к основным группам команд процессора. Команда пересылки существенно использует два адреса (источник и получатель); команда безусловного перехода работает с одним адресом операнда (адресом перехода); команда останова вовсе не использует операндов. Удаление одного адреса операнда не повлияет на эти команды. Три операнда требуются арифметическим командам и командам условного перехода. Рассмотрим, как избавиться от одного адреса в этих командах.

Арифметические команды. Довольно часто в алгоритмах требуется изменить значение переменной оператором вида $v := v \circ x$, где $\circ$ обозначает знак операции. Примем правило – во всех арифметических командах записывать результат на место одного из операндов, а именно, на место первого операнда. Таким образом, формат арифметических команд принимает вид КОП A1 A2. Действие команды заключается в следующем: поместить в регистр первого операнда содержимое ячейки с адресом A1, поместить в регистр второго операнда содержимое ячейки памяти с адресом A2, выполнить операцию, записать результат в ячейку с адресом A1.

Команды условного перехода. Напомним, как выполняются эти команды в УМ-3: проводится проверка, удовлетворяют ли условию перехода содержимое ячеек с адресами A1 и A2; если условие оказалось истинным, в регистр счётчик адреса записывется адрес перехода A3. Все три адреса команды необходимы для её работы. Чтобы использовать команды с двумя адресами, разделим действие команды на два этапа – сравнение операндов и сам переход по условию. Вопрос в том, что такое сравнение и как команда перехода узнаёт, истинно ли условие перехода или нет.

Как вы помните, арифметические команды, помимо получения результата, устанавливают значения арифметических флагов CF, OF, SF и ZF. В системе команд модельной ЭВМ имеются переходы по условиям "если первый операнд меньше второго", "…меньше либо равен…" и т.п.

Обозначим *ОП1* и *ОП2* – первый и второй операнды соответственно. Математически неравенство *ОП1* Δ *ОП2* (здесь Δ обозначает знак отношения из набора {=, ≠, <, ≤, >, ≥}) эквивалентно неравенству *ОП1* - *ОП2* Δ 0. Значит, сравнение *ОП1* и *ОП2* можно заменить сравнением их разности с нулём, а истинность условия перехода проверить по значениям флагов. Таким образом, команда сравнения будет работать как вычитание с установкой флагов, только без записи результата в память. Далее команда условного перехода, анализируя значения флагов, определяет, выполняется ли условие перехода и, если условие истинно, меняет значение счётчика адреса.

Проверка истинности условия перехода осуществляется следующим образом.

= Равенство *ОП1* - *ОП2* = 0 выполняется тогда и только тогда, когда ZF = 1; команде перехода достаточно проверить значение флага ZF.

Отношение "≠" проверяется аналогично.

Проверка истинности других отношений зависит от того, являются ли операнды числами со знаком или числами без знака. Рассмотрим сначала случай, когда операнды – числа без знака.

< Неравенство ОП1 - ОП2 < 0 означает, что при выполнении вычитания чисел без знака получился отрицательный результат, т.е. произошло переполнение. Это возможно тогда и только тогда, когда CF=1.

≥ Отношение ОП1 - ОП2 ≥ 0 есть отрицание предыдущего отношения:
(ОП1 - ОП2 ≥ 0) ≡ not (ОП1 - ОП2 < 0).
Значит, оно эквивалентно отрицанию условия not (CF=1), а именно, условию CF=0.

≤ Условие ОП1 - ОП2 ≤ 0 ("меньше либо равно") выражается комбинацией условия меньше и условия равно:
(ОП1 - ОП2 < 0) or (ОП1 - ОП2 = 0),
которая эквивалентно выражению (CF=1) or (ZF=1).

> Отношение ОП1 - ОП2 > 0 эквивалентно отрицанию предыдущего условия следовательно, оно реализуется проверкой (CF=0) and (ZF=0).

Пусть теперь операнды – числа со знаком.

Разберём проверку условия "меньше". Условие ОП1 - ОП2 < 0 означает, что результат вычитания ОП1 - ОП2 отрицательный. Возможно два случая.

Во-первых, математический результат попадает в диапазон представимых чисел со знаком, машинное вычитание выполнено корректно, и при этом результат отрицателен. Эта ситуация описывается выражением (OF=0) and (SF=1).

Во-вторых, математический результат вычитания может выйти из диапазона чисел, представимых как числа со знаком, т.е. машинный результат не совпадает с математическим, произошло переполнение. В этой ситуации значение флага SF противоречит знаку математического результата. Второй случай описывается выражением (OF=1) and (SF=0).

Формально, нужно объединить два полученных условия – либо первая ситуация, либо вторая. Но можно записать соответствующее условие короче: OF≠SF.

Исходя из соотношения ОП1 - ОП2 < 0 ⇔ OF≠SF, повторяя цепочку рассуждений, проделанную выше, получим условия на флаги для всех команд перехода для чисел со знаком. Условия сведены в таблицу:

Условие перехода	**Проверяемое условие**	**Условие перехода**	**Проверяемое условие**
=	ZF=1		
≠	ZF=0		
числа без знака			
<	CF=1	≤	(CF=1) ***or*** (ZF=1)
≥	CF=0	>	(CF=0) ***and*** (ZF=0)
числа со знаком			
<	OF≠SF	≤	(OF≠SF) ***or*** (ZF=1)
≥	OF=SF	>	(OF=SF) ***and*** (ZF=0)

Таким образом, мы выяснили, что переход от трёхадресных команд к двухадресным принципиально осуществим. Рассмотрим, как этот переход реализуется в конкретной архитектуре и насколько он влияет на сложность программирования.

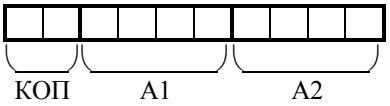
П2. Описание двухадресной учебной машины УМ-2 и программирование для этой машины

1. Устройство учебной машины УМ-2

Процессор машины УМ-2 отличается от процессора УМ-3 только разрядностью регистров. Регистр команды, регистры ОП1, ОП2 и РЕЗ состоят из десяти 16-разрядных ячеек. Регистр СА по-прежнему состоит из четырёх разрядов, позволяя адресовать 16^4 ячеек оперативной памяти.

Оперативная память состоит из 16^4ячеек, имеющих адреса от 0000_{16} до $FFFF_{16}$. Ячейка содержит десять шестнадцатиричных разрядов. Ячейка может хранить число (без знака или со знаком, представленным в дополнительном коде) или команду.

Команды машины имеют следующий формат:



Существуют следующие команды.

Арифметические команды. В эту группу входят команды, совпадающие с командами трёхадресной машины: сложение (01), вычитание (02), умножение (03 для чисел со знаком и 13 для чисел без знака) и деление (04 и 14 для чисел со знаком и без знака соответственно). Семантика команды КОП А1 А2 следующая: записать в регистр ОП1 первый операнд (содержимое ячейки памяти с адресом А1); записать в регистр ОП2 второй операнд; выполнить операцию, задаваемую кодом операции КОП; записать результат из регистра РЕЗ в ячейку с адресом А1.

Кроме перечисленных команд имеется команда сравнения 05 А1 А2. Действие этой команды состоит в следующем: записать в регистры ОП1 и ОП2 первый и второй операнды; выполнить действие ОП1 - ОП2, получив значение РЕЗ и флаги. Результат вычитания из процессора в память не записывается.

Команды перехода. Коды операций те же, что и в машине УМ-3. Команды выглядят так: КОП А1 А2; переход осуществляется по адресу А2, первый адрес А1 не используется. Команда условного перехода по значениям флагов анализирует, истинно ли условие перехода; если условие истинно, записывает адрес А2 в регистр СА, если условие ложно, содержимое СА не изменяется. Команда перехода не меняет значения флагов.

Команда пересылки. Формат команды 00 А1 А2; действие заключается в следующем: содержимое ячейки с адресом А2 записывается в ячейку с адресом А1.

Команда останова имеет формат 99 А1 А2; действие команды заключается в завершении выполнения программы. Адреса А1 и А2 не используются.

2. Примеры программирования для УМ-2

Программировать для двухадресной машины, вообще говоря, не сложней, чем для трёхадресной. Однако семантика арифметических команд – запись результата на место первого операнда – иногда требует включения в программу подготовительных команд пересылки. Кроме того, нужно не забывать использовать команду сравнения перед командами условного перехода.

Тот факт, что результат арифметических команд записывается на место первого операнда, будем отражать в столбце комментария, используя нотацию оператора присваивания.

Пример 1. Вычислить значение $x = (a*(-21) \bmod 50 - b)^2$ по известным a, b.

Алгоритм.

x:= a;
x:= x *(-21);
t:= x;
x:= t mod 50;{ t:= t div 50}
x:= x-b;
x:= x*x

Распределение памяти.

a – 0100
b – 0101
t – 0102
x – 0103

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	00 0103 0100	x:= a
0001	03 0103 0007	x:= x *(-21)
0002	00 0102 0103	t:= x
0003	04 0102 0008	x:= t mod 50;{ t:= t div 50}
0004	02 0103 0101	x:= x-b
0005	03 0103 0103	$x*x \to x$
0006	99 0000 0000	стоп
0007	FF FFFF FFEB	$(-21)_{10}$
0008	00 0000 0032	50_{10}

Пример 2. Имеется целое неотрицательное число n. Записать в переменную *md* наименьшую из значащих цифр десятичной записи числа n.

Напомним алгоритм решения задачи:

md:= 10;
repeat *d*:= *n* mod 10;
  if *d* < *md* ***then*** *md*:= *d*;
  n:= *n* div 10
until *n* = 0

Распределение памяти.

n – 0100
md – 0101
n1 – 0102
d – 0103

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	00 0101 0009	*md*:= 10
0001	00 0102 0100	*n1*:=*n*
0002	14 0102 0009	***Rpt***: *d*:= *n1* mod 10, *n*1:= *n*1 div 10

0003	05 0103 0101	$d \geq md$?
0004	94 0000 0006	if $\geq$, goto ***Endif***
0005	00 0101 0103	$md := d$
0006	05 0102 000A	***Endif*** : $n1 \neq 0$?
0007	82 0000 0002	if $\neq$, goto ***Rpt***
0008	99 0000 0000	стоп
0009	00 0000 000A	10_{10}
000A	00 0000 0000	0_{10}

Напоминаем, что подчёркивание означает, что поля команд были заполнены после того, как стали известны соответствующие адреса.

Если посчитать длину полученной программы в разрядах (без учёта констант), и сравнить с длиной соответствующей программы для УМ-3, увидим, что, несмотря на дополнительные команды сравнения, программа получилась короче.

П3. Заключение

Сравним двухадресную и трёхадресную архитектуры. В двухадресной машине эффективнее используется память за счёт того, что в командах не хранится избыточная информация. Устройство процессора в двухадресной архитектуре не сложнее, чем в трёхадресной, поскольку основные команды – арифметические – выполняются по одним и тем же алгоритмам в обеих архитектурах и, следовательно, реализуются одинаковыми по сложности электронными схемами. Быстродействие машин УМ-3 и УМ-2 примерно одинаково, так как выполнение арифметических команд в обеих архитектурах требует трёх обращений к ОП (самых медленных операций компьютера). Программировать в двухадресной архитектуре не сложнее, чем в трёхадресной. Экономичность и вместе с тем удобство двухадресных команд определили их успешность в истории вычислительной техники.

Как пример ЭВМ с двухадресной системой команд можно привести советскую машину Минск-32. Машина производилась с 1968 г. по 1975 г. и стала самой массовой машиной того времени в СССР. ЭВМ Минск-32 применялась для проведения инженерных и научных расчётов, решения планово-экономических задач [4].

Двухадресные арифметические команды имела машина Z-3 немецкого инженера Конрада Цузе, построенная в 1941 году [8]. Двухадресные команды были в знаменитых машинах IBM 360, есть они в процессорах Intel, используемых в современных персональных компьютерах [6].

§4. Учебная машина с переменным форматом команд (УМ-П)

Для экономного использования оперативной памяти мы избавились от хранения избыточной информации в ячейке, удалив третий адрес из машинных команд, перейдя к двухадресной архитектуре. Однако, если проанализировать набор машинных операций, можно заметить, что не для всех операций нужны два адреса. Операции останова вообще не требуются операнды, операции переходов используют один операнд – адрес перехода. Попытаемся полностью избавиться от избыточности, оставив в машинной команде логически необходимое для выполнения соответствующей операции количество адресов. При этом появятся машинные команды разных форматов. Такую архитектуру принято называть архитектурой с переменным форматом команд.

В данном параграфе мы рассмотрим модельную машину с переменным форматом команд УМ-П. Возьмём за основу двухадресную архитектуру машины УМ-2.

П1. Описание учебной машины УМ-П

1. Форматы команд

Рассмотрим основные группы команд с точки зрения необходимого количества операндов.

Арифметические команды. Арифметическим операциям требуются два операнда, поэтому команды этой группы в УМ-П будут двухадресными: КОП A1 A2. Длина команды составляет десять разрядов: два разряда хранят код операции, следующие четыре разряда содержат адрес первого операнда, последние четыре – адрес второго операнда.

Команда пересылки. Для выполнения пересылки нужно знать, что пересылать и куда пересылать. Операция требует двух операндов, следовательно, машинная команда пересылки двухадресная. Формат и длина команды такие же, как у арифметических команд.

Команды перехода. При выполнении *безусловного перехода* нужно знать адрес команды, на которую требуется передать управление (адрес перехода). Командам *условного перехода* кроме адреса перехода требуется информация об истинности условия перехода. Эта информация получается путём анализа значений флагов, расположенных в ЦП. Само условие перехода кодируется в КОП. Таким образом, командам условного перехода, так же как и команде безусловного перехода, требуется один операнд – адрес перехода. Мы приходим к одноадресному формату команд

перехода: КОП А1. Длина команды составляет шесть разрядов, два разряда для кода операции, четыре для хранения адреса перехода.

Команда останова. Этой команде не требуются операнды, она состоит только из кода операции, формат команды: КОП. Длина команды равна двум разрядам.

Итак, мы получили три возможные значения длины команды, а именно, два, шесть и десять разрядов. Как хранить в ячейках памяти эти команды? Все ячейки состоят из одинакового количества разрядов, поэтому привычное соответствие, когда одна ячейка содержала одну команду, в новой ситуации невозможно. Есть два логически возможных варианта.

Первый вариант – сделать размер ячейки равным размеру самой длинной команды, по десять разрядов. В этом случае команды длины шесть разрядов не заполнят ячейку целиком, останутся свободные разряды, т.е. разряды ячейки используются неэкономно. Можно заполнять свободные разряды ячейки короткими командами. Получим, что ячейка содержит несколько команд, например, короткие команды длины два можно хранить по пять команд в ячейке. При этом возникает следующая сложность. Поскольку, в силу своей природы, вся ячейка имеет один адрес (ячейка – минимально адресуемая единица памяти), всем командам, хранящимся в ячейке, соответствует один адрес. В таком случае переходы можно осуществлять не на любую команду программы, а только на те команды, которые записаны в первых разрядах ячейки. Процесс программирования в этом случае сильно усложняется. При записи машинных команд в программе приходится учитывать не только порядок действий в алгоритме, но то, как команды будут располагаться внутри ячейки, при необходимости переупорядочивать команды программы. Внесение изменений в программу чрезвычайно неудобно и чревато появлением большого количества ошибок.

Второй вариант – сделать размер ячейки равным размеру самой короткой команды, по два разряда. При этом команда длины шесть разрядов будет размещаться в трёх последовательных ячейках памяти, а команда длины десять разрядов – в пяти. Проблем, затрудняющих программирование, в этом случае не возникает. Но есть трудность иного рода. Выполнение каждой машинной команды становится более сложным, чем в машинах УМ-3, УМ-2. Сначала процессор должен считать из памяти КОП. Затем определить по внутренним таблицам размер команды, считать из ОП необходимое количество ячеек (окончание команды) в регистр команды и увеличить на соответствующее число значение регистра СА. Соответственно, сам процессор в архитектуре с переменным форматом команды становится более сложным и, следовательно, более дорогим.

2. Оперативная память

Оперативная память состоит из 16^4 ячеек. Адреса изменяются от 0000_{16} до $FFFF_{16}$. Каждая ячейка содержит два шестнадцатиричных разряда. Ячейки хранят числа или команды.

Размер числа – десять разрядов. Одно число занимает пять ячеек, расположенных подряд. Числа со знаком представляются в дополнительном коде (дополнение до 16^{10}).

Команды могут занимать одну, три или пять ячеек; длина команды зависит от её формата. Таблица форматов команд приведена в следующем пункте, посвящённом процессору.

3. Процессор машины УМ-П

Процессор машины с переменным форматом команд отличается от процессора двухадресной машины разрядностью регистров. Регистр счётчик адреса, по-прежнему, состоит из четырёх разрядов. Регистр команды состоит из десяти разрядов, чтобы длинная команда уместилась в нём целиком. Регистры первого и второго операндов и регистр результата предназначены для хранения чисел, поэтому их разрядность равна длине числа – десяти разрядам.

Набор машинных операций тот же, что в УМ-2. Семантика команд та же, что в УМ-2, форматы машинных команд УМ-П представлены ниже.

Название	**КОП**	**Формат**	**Длина команды**
останов	99	КОП	1
пересылка	00	КОП A1 A2	5
сложение	01	КОП A1 A2	5
вычитание	02	КОП A1 A2	5
умножение			
со знаком (без знака)	03 (13)	КОП A1 A2	5
деление			
со знаком (без знака)	04 (14)	КОП A1 A2	5
сравнение	05	КОП A1 A2	5
безусловный переход	80	КОП A1	3
переход по =	81	КОП A1	3
по ≠	82	КОП A1	3
по <	83 (93)	КОП A1	3
по ≥	84 (94)	КОП A1	3
по >	85 (95)	КОП A1	3
по ≤	86 (96)	КОП A1	3

П2. Приёмы программирования для машины УМ-П

Договоримся при записи машинной программы на листе для большей наглядности располагать одну команду в строке. Таким образом, часто в строке будет выписано содержимое не одной, а нескольких ячеек, составляющих команду. Это обстоятельство будем учитывать в колонке адресов, указывая адрес первой ячейки строки.

При распределении памяти необходимо помнить, что теперь число хранится не в одной ячейке, а в пяти, так что каждая переменная занимает пять последовательных ячеек.

Пример 1. Вычислить значение $x = (a*(-21) \bmod 50 - b)^2$ по известным a, b.

Алгоритм.

$x := a$;
$x := x*(-21)$;
$t := x$;
$x := t \bmod 50$;{ $t := t \operatorname{div} 50$}
$x := x-b$;
$x := x*x$

Распределение памяти.

a – 0100
b – 0105
t – 010A
x – 010F

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	00 010F 0100	$x := a$
0005	03 010F 001F	$x := x*(-21)$
000A	00 010A 0103	$t := x$
000F	04 010A 0024	$x := t \bmod 50$;{ $t := t \operatorname{div} 50$}
0014	02 010F 0105	$x := x-b$
0019	03 010F 010F	$x*x \to x$
001E	99	стоп
001F	FF FFFF FFEB	$(-21)_{10}$
0024	00 0000 0032	50_{10}

Программа получилась практически такая же, что и для машины УМ-2. Этого следовало ожидать, ведь форматы арифметических команд в обеих машинах совпадают. Различие заключается только в адресах переменных и команд.

Пример 2. Имеется целое неотрицательное число n. Записать в переменную *md* наименьшую из значащих цифр десятичной записи числа n.

Алгоритм.

md:= 10;
repeat *d*:= *n* mod 10;
    if *d* < *md* ***then*** *md*:= *d*;
    n:= *n* div 10
until *n* = 0

Распределение памяти.

n – 0100
md – 0105
n1 – 010A
d – 010F

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	00 0105 0009	*md*:= 10
0005	00 010A 0100	*n1*:=*n*
000A	14 010A 0025	***Rpt***: *n1* mod 10 → *d, n1* div 10 → *n1*
000F	05 010F 0105	*d* ≥ *md*?
0014	94 001C	if ≥, goto ***endif***
0017	00 0105 010F	*md*:= *d*
001C	05 010A 002A	***endif*** **:** *n1* ≠ 0?
0021	82 000A	if ≠, goto ***Rpt***
0024	99	стоп
0025	00 0000 000A	10_{10}
002A	00 0000 0000	0_{10}

Полученная программа короче (если измерять длину в количестве разрядов) аналогичной программы для УМ-2: длина команд перехода и останова сократилась, при этом лишние команды, связанные с изменением архитектуры машины, не появились.

П3. Заключение

Сравнивая машину с переменным форматом команд с двухадресной и трёхадресной машинами, можно сделать следующие выводы. Очевидным преимуществом переменного формата команд является экономное использование памяти. Процесс выполнения арифметических команд включает в себя те же три обращения к памяти, что и в двух- и трёхадресной машинах (получение первого и второго операндов и запись результата). Однако сам алгоритм обработки процессором одной машинной команды стал сложнее и медленнее: требуется прочитать из памяти код операции, определить размер команды, прочитать оставшуюся часть команды (дополнительные обращения к памяти!) Усложнение алгоритма выполнения команды вызывает усложнение реализующих этот алгоритм электронных схем и, в конечном счёте, усложнение и удорожание самого процессора. Относительно программирования в архитектуре с переменным форматом команд можно сказать, что программировать достаточно удобно, команды

имеют естественное количество операндов, поэтому процесс программирования не сложнее, чем для двухадресной архитектуры.

Экономное использование памяти и удобство программирования обеспечили популярность этой архитектуры в истории вычислительной техники. В качестве примеров можно привести следующие процессоры, имеющие переменный формат команд: процессоры компании Intel, используемые в персональных компьютерах, процессор Ultra SPARC II (компания SUN) [6], процессоры ARM, используемые во встраиваемых системах и в телефонах [7], семейство процессоров Motorola 68000.

§5. Одноадресная учебная машина (УМ-1)

При переходе от трёхадресной системы команд к двухадресной длина команды сократилась. Посмотрим, что получится, если мы продолжим двигаться по пути сокращения длины команды. Пусть команда содержат только одно поле адреса. Рассмотрим, какую архитектуру получим в этом случае.

П1. Обсуждение одноадресной архитектуры

Начнём рассмотрение, напомнив основные команды наших модельных вычислительных машин: арифметические команды (включая сравнение), команда пересылки, команды переходов и команда останова. Команды последних двух групп проблем не представляют, поскольку командам пересылки нужен один адрес (адрес перехода), а команде останова совсем не нужен операнд.

Какова будет семантика арифметических команд, если в команде хранится только один адрес операнда? Откуда процессору брать другой операнд? Вспомним, что в процессоре имеются служебные регистры, связанные с арифметическим устройством, это регистры первого и второго операнда и сумматор. Поступим просто – договоримся, что первый операнд берётся из регистра-сумматора и результат выполнения команды записывается в этот же регистр. Обозначим сумматор S.

Опишем архитектуру одноадресной модельной машины УМ-1 более точно.

Формат команд имеет вид КОП А. Код операции занимает два шестнадцатиричных разряда, адрес – четыре разряда. Ниже описано действие команд.

Арифметические команды: коды операций 01 (сложение), 02 (вычитание), 03 и 13 (умножение для чисел со знаком и для чисел без знака), 04 и 14 (деление для чисел со знаком и для чисел без знака). Выполнение команды происходит по алгоритму: записать содержимое ячейки с адресом А в

регистр второго операнда ОП2; выполнить операцию S○ОП2 (здесь ○ – знак операции, задаваемой КОП); записать результат в сумматор S. Команда деления вырабатывает два результата, частное и остаток. Частное записывается в регистр S, остаток – в дополнительный регистр S1. Обратим внимание, что теперь при выполнении арифметической команды требуется сделать только одно обращение к оперативной памяти, а не три, как в рассмотренных ранее моделях. Таким образом, арифметические команды выполняются почти в три раза быстрее, чем в УМ-3, УМ-2.

Команда сравнение: 05 A. Работает следующим образом: операнд с адресом A записывается в регистр ОП2; выполняется вычитание S-ОП2 с установлением значений флагов; результат вычитания не записывается в S. Таким образом, значение регистра сумматора не изменяется в результате выполнения сравнения.

Команды пересылок. Поскольку одним из операндов арифметических команд всегда является сумматор, мы должны иметь возможность записать в сумматор нужное значение (из ячейки памяти), подготовив его к вычислениям, и выполнить обратное действие, сохранить полученный результат в нужную ячейку памяти (из сумматора). Кроме того, иногда требуется поменять местами содержимое сумматоров S и S1. Таким образом, требуется три команды пересылки – пересылка содержимого ячейки памяти в сумматор, обратная пересылка содержимого сумматора в ячейку памяти и обмен S ↔ S1.

Команда	**Действие**	**Команда**	**Действие**
00 A	S:= [A1]	20 A	S ↔ S1, значение A не используется
10 A	S → [A1]		

Команды перехода и ***команда останова***: коды операций и семантика те же, что в машине УМ-2.

Ячейки оперативной памяти машины УМ-1 состоят из 6 разрядов, размер ячейки совпадает с длиной команды. Объём оперативной памяти – 16^4 ячеек. Как обычно, ячейка может содержать команду или число. Используются числа со знаком (в дополнительном коде) и числа без знака.

П2. Примеры программирования для машины УМ-1

Заметим, что в модели УМ-1 нам всё время придётся следить за содержимым сумматора. Посмотрим, как будут выглядеть программы разобранных ранее задач для машины УМ-1.

Пример 1. Вычислить значение $x = (a*(-21) \bmod 50 - b)^2$ по известным a, b.

Алгоритм.

S:= *a*;
S:= *S* *(-21);
S1:= *S* mod 50;
S ↔ *S1*;
S:= *S*-*b*;
S → *t*;
S:= *S***t*;
S → *x*

Распределение памяти.

a – 0100
b – 0101
x – 0102
t – 0103

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	00 0100	*S*:= *a*
0001	03 0009	*S*:= *S* *(-21)
0002	04 000A	*S1*:= *S* mod 50
0003	20 0000	*S* ↔ *S1*
0004	02 0101	*S*:= *S*-*b*
0005	10 0103	*S* → *t*
0006	03 0103	*S*:= *S***t*
0007	10 0102	*S* → *x*
0008	99 0000	стоп
0009	FF FFEB	$(-21)_{10}$
000A	00 0032	50_{10}

В команде 0003 (обмен значений основного и вспомогательного сумматоров) адрес не требуется, но все разряды ячейки нужно заполнить, поэтому в поле адреса стоит 0000, так же, как и в команде останова 0008.

Пример 2. Имеется целое неотрицательное число *n*. Записать в переменную *md* наименьшую из значащих цифр десятичной записи числа *n*.

Алгоритм.

md:= 10;
repeat *d*:= *n* mod 10;
    if *d* < *md* ***then*** *md*:= *d*;
    n:= *n* div 10
until *n* = 0

Реализуем следующую модификацию алгоритма, указав явно использование сумматора:

S:=10; *S* → *md*; {*md*= 10}; *S*:= *n*;
repeat *S1*:= *S* mod 10;{*d*} {*S*:=*S* div 10, *n*}
S ↔ *S*1; {*S*=*d*, *S*1=*n*}
if *S* < *md* ***then*** *md*:= *S*;
S ↔ *S*1 {*S*=*n*, *S*1=*d*}
until *S* = 0

Распределение памяти.

n – 0100
md – 0101

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	00 000C	*S*:= 10
0001	10 0101	*md*:= *S*
0002	00 0100	*S*:=*n*
0003	14 000C	***Rpt***: *S* mod 10 → *S1, S* div 10 → *S*
0004	20 0000	*S* ↔ *S1*
0005	05 0101	*S* ≥ *md*?
0006	94 0009	if ≥, goto ***endif***
0007	10 0101	*md*:= *S*
0008	20 0000	*S* ↔ *S1*
0009	05 000D	***endif*** **:** *S* ≠ 0?
000A	82 0003	if ≠, goto ***Rpt***
000B	99 0000	стоп
000C	00 000A	10_{10}
000D	00 0000	0_{10}

Пример показывает, что программировать для УМ-1 сложнее, чем для УМ-П: перед написанием машинной программы нам потребовалось преобразовать алгоритм – переписать его в терминах сумматора и дополнительного сумматора, а не в терминах переменных задачи, пришлось выписать команды обмена сумматоров, не относящиеся к алгоритму решения задачи.

Программа получилась короче, чем программа для УМ-П, и тем более короче, чем для УМ-2, несмотря на использование фиктивных адресов в командах, где не требуются операнды и несмотря на лишние команды подготовки значения сумматора. Экономия получилась за счёт того, что основные команды – арифметические – очень короткие.

Следует напомнить, что программа на УМ-1 работает много быстрее, чем программа решения той же задачи на УМ-2.

П3. Заключение

Наиболее медленные операции в компьютерах связаны с обменом между памятью и процессором. Конечно, одноадресная архитектура, позволяющая уменьшить количество таких обменов, использовалась в реальных вычислительных машинах, несмотря на относительное неудобство программирования.

Приведём в качестве примеров две машины, построенные в Америке. Прежде всего, это машина Atlas, построенная в 1950 г., третья из работающих машин с хранимой программой (Д. Хилл, Ф. Маллени, А. Коэн). Один экземпляр машины работал для ВМФ США, второй – в Агентстве национальной безопасности. В работе по модификации Atlas принимал участие Сеймур Крей [9].

Вторая машина, которую хотелось бы упомянуть, это Whirlwind (“Вихрь”). Первая в Америке машина, решавшая задачи в режиме реального времени, предназначенная для обработки данных радиолокационных наблюдений, была построена в 1951 г. (Джей Форрестер, Роберт Эверетт). Интерактивное общение с машиной обеспечивалось “световой пушкой” (light gun), подобной тем, что применялись в световых тирах. Световая пушка позволяла оператору выделить объект на экране дисплея, ЭВМ извлекала из внешней памяти и выдавала на дисплей данные о воздушной цели, скорости и направлении ее полета, типе вооружения и т. д. Операторы, опираясь на эту информацию, могли принимать необходимые решения (в частности, передавать координаты и курс целей самолетам-перехватчикам) [10].

Перейдём теперь к отечественной вычислительной технике. Одноадресной была первая советская серийная ЭВМ Урал-1, разработанная в 1954-1955 годах под руководством Б.И.Рамеева и производившаяся с 1956 по 1961 гг. Было выпущено 183 экземпляра машины. Машина была относительно недорогой и использовалась для инженерно-технических и экономических расчётов на производствах, в вычислительных центрах НИИ, конструкторских бюро. Одна из машин использовалась на космодроме «Байконур» для расчёта полёта ракет [5, 11].

Одноадресной была выдающаяся по быстродействию и архитектурным решениям машина БЭСМ-6. Машина разрабатывалась под руководством С.А.Лебедева в Институте точной механики и вычислительной техники (ИТМ и ВТ) АН СССР и на Московском заводе счетно-аналитических машин (САМ); в создании машины принимали участие В.А.Мельников, Л.Н.Королев, В.С.Петров, Л.А.Теплицкий, А.Н.Томилин, ведущие разработчики программного обеспечения – В. П. Иванников, А. Н. Томилин, Д. Б. Подшивалов, В. Ф. Тюрин, Э. З. Любимский, Ю. М. Баяковский. ЭВМ БЭСМ-6, выпускалась с 1968 по 1987; было построено 355 машин. В БЭСМ-6 были реализованы многие оригинальные решения, определившие перспективу дальнейшего развития ЭВМ общего назначения: механизм

модификации адресов, конвейерный центральный процессор (с отдельными конвейерами для устройства управления и арифметического устройства), расслоение памяти, КЭШ-память, виртуальная организация памяти, система прерываний, привилегированный и пользовательский режим работы. Согласно [12], "В 1975 году, в ходе космического полёта «Союз-Аполлон», управление осуществлялось комплексом, в состав которого входила БЭСМ-6. Эта система позволяла обрабатывать данные по траектории полёта за 1 минуту, в то время как на американской стороне такой расчёт занимал 30 минут."

§6. Стековая учебная машина (УМ-С)

Пойдём по пути дальнейшего сокращения длины команды. Рассмотрим, какую модель машины при этом получим.

П1. Безадресная система команд

Есть одна логическая возможность сократить одноадресную команду – убрать из неё поле адреса. При этом команда будет состоять только из кода операции. Однако нужно решить, откуда брать операнды? И куда записывать результат? Договоримся операнды и результат хранить в специальной структуре – стеке.

1. Организация стека

В программировании стек – это структура данных с дисциплиной обслуживания LIFO (Last In First Out, последний пришёл, первый ушёл). В нашем случае стек будет располагаться в оперативной памяти, а дисциплину обслуживания будут обеспечивать специальный регистр, указывающий на вершину стека, и семантика работы машинных команд.

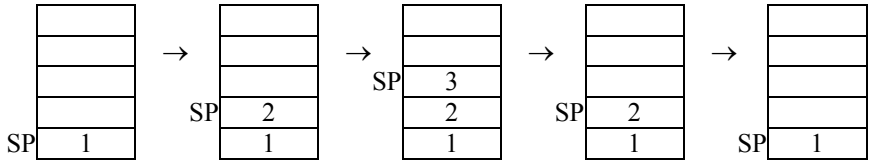


Рис. 5. Работа со стеком.

Пусть стек располагается в области памяти со старшими адресами. При записи чисел в стек заполняются сначала последние ячейки памяти, затем ячейки с меньшими адресами и т.д. Вершина стека поднимается. При выборке чисел из стека сначала берутся числа с меньшими адресами, затем – с большими, вершина стека опускается, как показано на рисунке 5.

Адрес последнего записанного в стек числа хранится в специальном регистре процессора – указателе стека SP (stack pointer).

2. Выполнение арифметических команд

Итак, теперь полагаем, что арифметические команды работают только со стеком и не имеют дела с обычными ячейками памяти. Рассмотрим, как выполняется безадресная команда вычитания. Основные шаги:

1. Взять из стека второй операнд и поместить в регистр ОП2.
2. Взять из стека первый операнд и поместить в регистр ОП1.
3. Выполнить вычитание ОП1-ОП2 (результат – в сумматоре S, установлены все арифметические флаги).
4. Положить в стек результат из S.

Изменение состояния стека в результате выполнения вычитания изображено на рисунке 6.

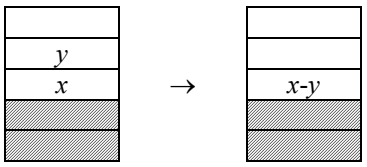


Рис.6. Операция вычитания.

Конечно, перед выполнением вычитания стек следует подготовить, положив в него первый и второй операнды. Если вспомнить, что данные хранятся в обычных ячейках памяти, становится ясно, что требуется переслать содержимое ячейки (с некоторым известным программисту адресом) в стек. Таким образом, в машине, по крайней мере, будут команды пересылки из памяти в стек и обратно, из стека в память, и в этих командах необходимо указать адрес ячейки. Так что все команды машины сделать безадресными не удастся. Однако самые важные для вычислений команды, арифметические, будут безадресными. Это обстоятельство и дало имя рассматриваемой архитектуре.

Определим теперь более точно архитектуру машины с безадресной системой команд.

П2. Описание стековой учебной машины УМ-С

1. Оперативная память

Выберем размер ячейки памяти равным длине короткой команды, т.е. два разряда, подобно тому, как мы поступали при обсуждении архитектуры с переменным форматом команды. Объём оперативной памяти оставим прежним – 16^4 ячеек с адресами от 0000_{16} до $FFFF_{16}$.

Числа будем представлять аналогично тому, как это сделано в УМ-П: одно число занимает три последовательно расположенные ячейки, используются числа без знака и числа со знаком (в дополнительном коде).

2. Система команд

Арифметические команды. Команда состоит только из кода операции. Имеются команды сложения (01), вычитания (02), умножения чисел со знаком (03) и чисел без знака (13), деления чисел со знаком (04) и деления чисел без знака (14). Кроме этого, есть команда сравнения (05). Все арифметические команды работают по приведённому выше алгоритму: вынимают из стека второй и первый операнды, выполняют вычисление и помещают результат в стек. Команда деления записывает в стек два результата, сначала частное, потом остаток. Команда сравнения ничего не записывает в стек, только устанавливает значения флагов. Одна арифметическая команда занимает одну ячейку.

Команды работы со стеком. К этой группе относятся команды, позволяющие поместить в стек операнды перед арифметической операцией и взять результат операции из стека, а также служебные операции, необходимые для программирования вычислений. Команды, которые осуществляют обмен между стеком и обычными ячейками памяти, имеют формат КОП A, где КОП занимает два разряда, адрес A – четыре разряда. Таким образом, длина таких команд составляет три ячейки. Это следующие команды:

в стек, формат 5A A, действие: содержимое ячейки с адресом A записывается в стек (при этом уменьшается значение SP); флаги не меняются.

из стека, формат 5B A, действие: в ячейку с адресом A записывается верхний элемент из стека (при этом увеличивается значение SP); флаги не меняются.

Другие команды работают только со стеком и не используют адреса, состоят только из кодов операций:
дублирование вершины стека 5C и ***обмен*** двух верхних элементов стека 5D. Действие команд изображено на рисунке 7.

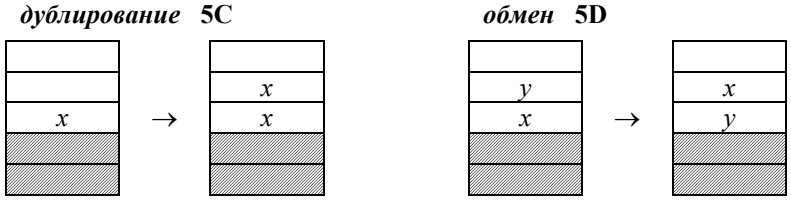


Рис. 7. Стековые операции.

Команды ***переходов*** и ***останова*** имеют те же форматы и семантику, что в УМ-П.

П3. Программирование для УМ-С

Программировать в стековой архитектуре сложней, чем в рассмотренных ранее моделях. Для написания программы недостаточно просто разбить алгоритм на мелкие шаги, соответствующие машинным операциям, необходимо сам алгоритм организовать по-другому. Приходится постоянно следить за состоянием стека, поэтому в запись программы добавим новую колонку – содержание стека. Будем записывать элементы стека в одной строке слева направо, вершина стека находится справа.

1. Вычисление по формулам

Для того, чтобы получить алгоритм вычисления по формуле, пригодный для реализации на УМ-С, нужно преобразовать формулу в постфиксную запись. Напомним, что в постфиксной записи знак операции записывают после операндов этой операции.

Пример 1. Вычислить значение $x = (a*(-21) \bmod 50 - b)^2$ по известным a, b.

Для получения постфиксной записи выражения нужно построить дерево выражения, затем обойти дерево в концевом порядке (левое поддерево – правое поддерево – корень). Для нашего выражения получим

$$a \ -21 \ * \ 50 \ \text{mod} \ b \ - \ \wedge 2$$

где знак $\wedge 2$ означает унарную операцию возведение в квадрат.

По постфиксной записи машинная программа для УМ-С строится непосредственно: читаем формулу и выписываем соответствующие машинные команды. Если видим переменную или константу, используем команду записи в стек соответствующей ячейки, если видим знак операции, выписываем арифметическую команду, выполняющую эту операцию.

Возведение в квадрат реализуем как умножение операнда на себя, при этом нам потребуется дублирование вершины стека.

Распределение памяти.

a – 0100
b – 0103
x – 0106

Программа.

Адрес	Содержимое ячейки	Комментарий	Стек
0000	5A 0100	в стек *a*	*a*
0003	5A 0015	в стек -21	*a,* -21
0006	03	*	(*a**(-21))

0007	5A 0018	в стек 50	(*a**(-21)), 50
000A	04	/	*div, mod*
000B	5A 0103	в стек *b*	*div, mod, b*
000E	02	-	*div, mod - b*
000F	5C	дублирование	*div, mod - b, mod - b*
0010	03	*	*div,* $(mod - b)^2$
0011	5B 0106	из стека *x*	*div*
0014	99	стоп	*div*
0015	FF FFEB	$(-21)_{10}$	
0018	00 0032	50_{10}	

После окончания работы программы в стеке остался один элемент – частное, которое не потребовалось для вычислений. Строго говоря, можно запрограммировать удаление этого элемента из стека, но для сохранения прозрачности машинной программы мы не включили в неё команды очистки стека.

Если сравнить размер полученной программы с программой решения той же задачи для УМ-П, обнаружим, что программа для УМ-С намного короче. Это не случайно, в вычислительных задачах используется много арифметических операций, а арифметические команды в стековой модели очень короткие.

2. Программирование сложных управляющих структур

Рассмотрим, как реализуются в УМ-С традиционные для программирования управляющие структуры на примере вычисления наибольшей цифры числа.

Пример 2. Имеется целое неотрицательное число *n*. Записать в переменную *md* наименьшую из значащих цифр десятичной записи числа *n*.

Алгоритм.

```
md:= 10;
repeat   d:= n mod 10;
      if d < md  then  md:= d;
      n:= n div 10
until   n = 0
```

Как и в примере 1, из-за особенностей стековой архитектуры напрямую записать алгоритм в виде машинной программы не удастся. Обсудим, как реализовать в УМ-С некоторые сложные моменты алгоритма.

Прежде всего, проблему представляет реализация первого же оператора присваивания, поскольку в УМ-С нет команд пересылок из одной ячейки в другую. Действие *md*:= 10 реализуется двумя командами: в стек 10, из стека *md*.

В цикле на каждом шаге происходит вычисление цифры *d* и получение нового значения *n*. Собственно переменных (ячеек памяти) для этих величин мы заводить не будем, они будут храниться как элементы стека. Поэтому перед началом работы цикла нам нужно поместить значение *n* в стек. И оператора присваивания *d*:= *n* mod 10 в машинной программе не будет, частное запишется в стек автоматически при выполнении деления. Нам придётся только следить, где именно в стеке оно находится.

Ещё один момент. Нужно не забывать, что каждая из операций – вычисление *n* mod 10, выполнение сравнений $d < md$ и $n = 0$ – требует, чтобы операнды находились на вершине стека и при необходимости записывать их в стек.

Распределение памяти.

n – 0100
md – 0103
рабочая – 0106

Программа.

Адрес	Содержимое ячейки	Комментарий	Стек
0000	5A 0021		
0003	5B 0103	*md*:= 10	
0006	5A 0100	в стек *n*	*n*
0009	5A 0021	в стек 10	*n*, 10
000C	14	***Rpt***: /	*div*, *mod*
000D	5C	дублирование	*div*, *mod*, *mod*
000E	5A 0103	в стек *md*	*div*, *mod*, *mod*, *md*
0011	05	$d \geq md$?	*div*, *mod*
0012	94 0019	if ≥, goto ***endif***	
0015	5C	дублирование	*div*, *mod*, *mod*
0016	5B 0103	из стека →*md*	*div*, *mod*
0019	5B 0106	***endif*** **:** из стека mod	*div*
001C	5C	дублирование	*div*, *div*
001D	5A 0024	в стек 0	*div*, *div*, 0
0020	05	***endif*** **:** $n1 \neq 0$?	*div*(=*n*)
0021	82 0009	if ≠, goto ***Rpt***	
0024	99	стоп	*n* (=0)
0025	00 000A	10_{10}	
0028	00 0000	0_{10}	

Обратим внимание на то, что команды сравнения забирают операнды из стека. Поэтому для того, чтобы сохранить величины для продолжения вычислений, перед командами сравнений стоят команды дублирования

вершины стека. И так же, как в примере 1, после завершения программы в стеке остаётся последнее значение *n* (число 0), которое мы не удаляем.

Несмотря на неизбежные лишние команды подготовки операндов в стеке, программа получилась короткая, меньше, чем программа для этой же задачи на УМ-2. В этом смысле сокращение длины арифметических команд оправдало себя. Однако нельзя сказать, что машинная программа легко сопоставляется исходному алгоритму. Если убрать из текста правую колонку (содержание стека), программа станет малопонятной. Непрозрачность получающихся программ – серьёзный недостаток стековой архитектуры.

П4. Заключение

Подведём итог, какие же достоинства и недостатки имеет стековая архитектура? Большим преимуществом является, конечно, компактный размер программ. С точки зрения быстродействия стековая архитектура проигрывает одноадресной (три обращения к памяти (стеку) при выполнении арифметических команд против одного обращения в УМ-1) и почти эквивалентна двухадресной архитектуре. С точки зрения удобства программирования УМ-С хуже, чем УМ-2, программировать неудобно, требуется преобразовывать алгоритмы решения задач под стековую архитектуру, постоянно приходится анализировать состояние стека, программы получаются довольно запутанные.

На практике имеем следующее. Стек, как специальная структура с аппаратной поддержкой, оказался очень удобным средством для реализации процедур и используется практически во всех современных компьютерах. Идею использовать стек для работы с процедурами предложил Алан Тьюринг в 1946 г. в работе над проектом ACE. Эта идея впервые была реализована много позже, в 1955 г., в ЭВМ DEUCE [13].

Процессоры стековой архитектуры используются во встроенных системах (пластиковые карточки, видеокамеры, охранные системы, телевизоры, телефоны). Основными факторами являются простота и дешевизна процессора, небольшой объём программ. Объём программ очень важен, поскольку программа, управляющая работой устройства, должна быть встроена в само устройство. Для хранения программы используют специальную флэш-память.

Проблема быстродействия в этом случае решается следующим образом. В процессор встраивают так называемый регистровый файл – довольно большое количество регистров. Он предназначен для хранения верхней части стека. Большую часть времени процессор работает с этим встроенным стеком; если он заполняется до конца или становится пустым, происходит обмен между встроенным стеком и стеком в оперативной памяти.

В качестве примера можно привести процессор picoJava II компании SUN, являющийся основой (ядром) большого количества встраиваемых процессоров, ориентированных на исполнение Java-программ (Fujitsu MB86799, MB92901) [4, 14].

Среди отечественных машин, имеющих стековую архитектуру, самым известным является многопроцессорный вычислительный комплекс Эльбрус-2, разработанный в конце 1970-х гг. в институте Точной Механики и Вычислительной Техники АН СССР под руководством С.А.Лебедева, С.В.Бурцева, Б.А.Бабаяна. Комплекс разрабатывался для работы в режиме реального времени и используется в радиолокационной станции дальнего обнаружения Дон-2Н системы московской ПРО [15, 16].

§7. Учебная машина с регистрами (УМ-Р)

Из всех рассмотренных ранее моделей машин наибольшим быстродействием обладала машина с одноадресной архитектурой. Напомним, что выигрыш достигался за счёт сокращения количества обращений к оперативной памяти: при выполнении арифметических команд один операнд брался не из памяти, а из регистра-сумматора, расположенного в процессоре, и результат операции сохранялся в сумматоре, а не записывался в память. Пересылки из памяти в процессор и обратно – самые медленные этапы в выполнении машинных команд, если уменьшить их количество, скорость выполнения программ существенно увеличится. Идея минимизации количества обменов между процессором и оперативной памятью реализована в архитектуре с регистрами.

П1. Описание модельной машины с регистрами УМ-Р

За основу архитектуры машины с регистрами УМ-Р взята двухадресная модель УМ-2, она имеет достаточно логичную организацию и схему работы, удобна для программирования и даёт относительно неизбыточный программный код.

1.1. Устройство процессора машины УМ-Р

Схема процессора УМ-Р приведена на рисунке 8. На схеме изображены только элементы, важные нам для понимания работы процессора.

Основные элементы (устройство управления, АЛУ, регистры СА и РК, регистр флагов) те же, что и в процессорах машин УМ-3 и УМ-2. Помимо этих устройств, в процессоре находятся так называемые регистры. Регистры подобны ячейкам памяти, в них можно хранить числа. Можно сказать, что регистр – это ячейка, расположенная в процессоре, а не в оперативной памяти. Программист может использовать регистры в ка-

честве операндов машинных команд. В процессоре машины УМ-Р есть 16 регистров, они имеют номера от 0_{16} до F_{16}, каждый регистр состоит из восьми шестнадцатиричных разрядов. Если регистр является операндом машинной команды, в команде указывается номер этого регистра. Если оба операнда команды являются регистрами, длина команды составляет четыре разряда, т.к. два разряда занимает код операции и по одному разряду нужно для указания номеров регистров. Следовательно, ячейки ОП должны состоять из четырёх разрядов, чтобы было удобно хранить команды.

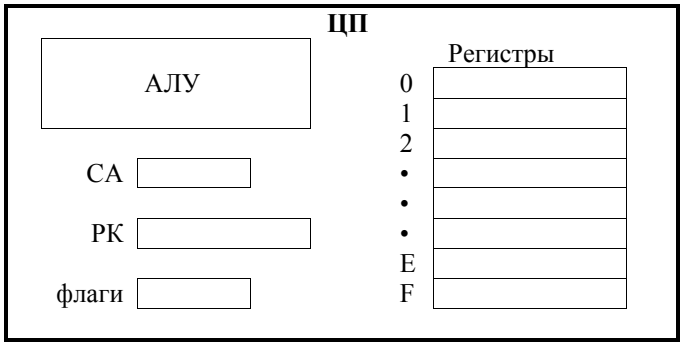


Рис. 8. Процессор машины УМ-Р.

1.2. Оперативная память машины УМ-Р

Память состоит из 16^4 ячеек, перенумерованных от 0000_{16} до $FFFF_{16}$. Размер ячейки – четыре 16-ричных разряда. Ячейки могут хранить числа или команды.

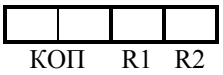
В четырёхразрядную ячейку можно записать относительно маленькое число, не больше, чем 16^4. Но желательно иметь возможность работать и с большими числами, поэтому договоримся считать, что число занимает две соседние ячейки, т.е. восемь разрядов. Можно использовать как числа без знака, так и числа со знаком. Числа со знаком представляются в дополнительном коде.

1.3. Система команд

Как упоминалось выше, за основу взята двухадресная система команд. Каждая команда состоит из кода операции и двух операндов. Первый операнд обязан быть регистром. Второй операнд может быть регистром или адресом ячейки памяти. Результат операции записывается в первый операнд.

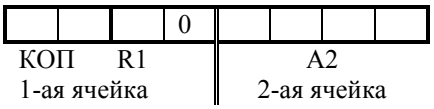
В УМ-Р есть следующие форматы команд:

1) Команда ***регистр-регистр***: оба операнда – регистры



Команда регистр-регистр занимает одну ячейку, является короткой. Выполнение такой команды не требует обращения к памяти, значит, происходит очень быстро. Поэтому везде, где это возможно, следует использовать команды данного типа.

2) Команда ***регистр-память***: первый операнд – регистр, второй операнд – адрес ячейки памяти



Команда регистр-память занимает две ячейки. В первой ячейке хранится код операции КОП и регистр, являющийся первым операндом R1. Последний разряд ячейки не используется, мы будем записывать в него число 0. Вторая ячейка содержит адрес второго операнда A2. При выполнении арифметической команды формата регистр-память происходит одно обращение к оперативной памяти для чтения второго операнда.

Все арифметические команды УМ-Р имеют две формы – короткую и длинную. Семантика арифметических команд традиционна: два операнда помещаются в регистры ОП1 и ОП2, связанные с АЛУ, выполняется операция с установлением значений флагов, результат вычислений записывается в первый операнд. Результаты команды деления записываются в два соседних регистра: частное – в первый операнд, остаток – в регистр, следующий за ним (в списке регистров ЦП). Команды переходов имеют только длинную форму, поскольку в команде необходимо указать адрес перехода. Коды команд перехода те же, что и в УМ-3.

Ниже приведена таблица кодов операций УМ-Р (кроме команд перехода).

Название команды	**регистр-память**	**регистр-регистр**
Останов		99
Пересылка	00	20
Обратная пересылка (R1 → A2)	10	
Сложение	01	21
Вычитание	02	22
Умножение со знаком (без знака)	03 (13)	23 (33)
Деление со знаком (без знака)	04 (14)	24 (34)
Сравнение	05	25

В системе команд УМ-Р, подобно УМ-1, есть команда обратной пересылки с кодом 10. Действие команды 10 R1 0 A2 таково: содержимое регистра R1 записывается в ячейку памяти по адресу A2. Название команды отражает тот факт, что запись происходит в направлении, противоположном обычному, не в первый операнд, а во второй.

П2. Пример программирования для машины УМ-Р

Разберём, как реализуется для машины УМ-Р алгоритм вычисления наибольшей цифры числа.

Пример. Имеется целое неотрицательное число *n*. Записать в переменную *md* наименьшую из значащих цифр десятичной записи числа *n*.

Напомним алгоритм решения задачи.

```
md:= 10;
repeat   d:= n mod 10;
      if d < md  then  md:= d;
      n:= n div 10
until   n = 0
```

Для получения наибольшего эффекта от наличия регистров в УМ-Р величины, использующиеся на каждом шаге цикла, нужно хранить в регистрах. В нашем алгоритме это переменные *d*, *n* и *md* и числа 10 и 0. Значит, для программирования цикла нам потребуются пять регистров. Договоримся, какие регистры будем использовать. Пусть число 0 хранится в регистре с номером 0 (обозначим его R0), а число 10 – в регистре с номером 10 (A_{16}, регистр RA). Переменные *n* и *d* нужно разместить в соседних регистрах, чтобы команда деления записывала в них новые значения одновременно и не понадобились бы дополнительные пересылки. Пусть переменной *n* соответствует регистр номер 6 (R6), а переменной *d* – регистр номер 7 (R7). Переменная *md* будет храниться в регистре номер 1 (R1).

Обратим внимание на следующий важный момент. Как мы договаривались в начале обсуждения принципов работы модельных ЭВМ, перед началом работы программы входные данные загружаются в известные ячейки оперативной памяти, а после завершения вычислений результат так же хранится в памяти. Мы не уточняем, как данные попадают в память и как из памяти извлекается результат. Эти функции выполняются устройствами ввода-вывода, они выходит за рамки обсуждаемой темы. А вот позаботиться о том, чтобы регистры процессора получили нужные для вычислений значения и в конце работы записать результат из регистров в ячейки памяти должна программа. Регистры недоступны для устройств ввода-вывода.

Распределение памяти.

n – 0100
md – 0102

Программа.

Адрес	Содержимое ячейки	Комментарий
0000	22 0 0	Загрузка констант *R0*:=0
0001	00 A 0 0011	*RA*:= 10
0003	20 1 A	*R*1:= 10 (*md*)
0004	00 6 0 0100	*R6*:= *n*
0006	34 6 A	***Rpt***: *d*:= *n* mod 10, *n*:= *n* div 10
0007	25 7 1	*d* ≥ *md*?
0008	94 0 0 000B	if ≥, goto ***Endif***
000A	20 1 7	*md*:= *d*
000B	25 6 0	***Endif*** **:** *n* ≠ 0?
000C	82 0 0 0006	if ≠, goto ***Rpt***
000E	10 1 0 0102	*R*1 → *md*
0010	99 0 0	стоп
0011	0000 000A	10_{10}

Программа устроена следующим образом. Перед основным циклом стоит группа команд загрузки регистров, хранящих константы. Для получения значения 0 применена команда вычитания R0-R0. Дело в том, что при включении процессора регистры содержат какие-то (какие конкретно, неизвестно) значения.[3] При вычитании из значения R0 его самого обязательно получим ноль. Используя эту команду вместо загрузки константы в регистр из памяти, мы сокращаем текст программы (не нужно запасать число 0 в ячейке, команда R0-R0 короче команды пересылки из памяти в регистр) и уменьшаем время счёта программы (вычитание R0-R0 работает быстрее, чем пересылка из памяти в регистр).

Затем идёт группа команд, которые записывают начальные значения в регистры-переменные. Далее располагаются команды, реализующие цикл.

После цикла использована команда обратной пересылки 10 1 0 0102, которая переписывает ответ из регистра *R*1 в переменную *md*, в оперативную память. Напомним, что это действие необходимо в машине УМ-Р, иначе результат вычислений будет потерян.

[3] Регистры состоят из разрядов (двоичных). Двоичный разряд – это физическое устройство, подобное электрической лампочке, оно обязательно находятся в каком-либо состоянии, то есть содержат двоичную цифру 0 или 1. Поэтому при включении ЦП регистр в целом содержит какое-то, наперёд неизвестное, число.

П3. Заключение

Какие выводы можно сделать, сравнив архитектуру УМ-Р с другими моделями и полученную программу с разобранными ранее программами? Прежде всего, устройство процессора УМ-Р сложней, чем устройство процессора, например, УМ-2. Процессор УМ-Р содержит дополнительные устройства – регистры. Кроме того, алгоритм выполнения машинной команды в модели УМ-Р сложней, чем в УМ-2, так как приходится прочитать первую ячейку команды, проанализировать тип команды (регистр-регистр или регистр-память), для команды регистр-память прочитать вторую ячейку команды и только затем начать её выполнение. Соответственно, процессор машины УМ-Р сложнее и дороже процессора УМ-2. Однако, количество регистров мало, поэтому сложность процессора УМ-Р выше сложности УМ-М не намного.

Быстродействие машины УМ-Р намного выше, чем УМ-2, за счёт сокращения количества обращений к памяти при выполнении арифметических команд. Программы для УМ-Р получаются короче, чем для УМ-2 из-за того, что наиболее используемые команды регистр-регистр короче команд УМ-2, несмотря на появление в программах лишних команд загрузки регистров и команд переписывания результатов из регистров в память. Программировать для УМ-Р не сложней, чем для УМ-2, и программы получаются довольно понятные.

Поскольку использование регистров даёт существенное улучшение быстродействия при относительно небольшом увеличении сложности (стоимости) процессора и не затрудняет программирование, регистры используются практически во всех компьютерах.

Впервые так называемые регистры общего назначения были использованы в вычислительной машине PEGASUS, разработанной в 1954-1955 гг. английской компанией Ferranti. Существенную роль в создании машины сыграл К. Стрейчи. Он выдвинул и всячески развивал подход, согласно которому вычислительная машина должна быть дешёвой, надёжной и удобной для пользователя (программиста). Машина PEGASUS такой и получилась. Она выпускалась в Англии с 1956 по 1962 гг., однако затем, к сожалению, была вытеснена с рынка американскими машинами [17].

§8. Учебная машина с модификацией адресов

П1. Сложность работы с массивами

Большой класс задач для вычислительных машин составляют задачи обработки массивов (последовательностей элементов, расположенных в оперативной памяти). Работа с массивами имеет свои особенности, которые обсудим на следующем примере. Рассмотрим задачу вычисления

суммы элементов массива в модели УМ-Р. Пусть в оперативной памяти находится массив из 20 элементов x[0..19] и пусть требуется записать в переменную S сумму элементов массива x. На языке высокого уровня задача решается с помощью использования индексных выражений. Сформулируем алгоритм следующим образом:

S:= 0; *i*:= 0;
repeat *S*:= *S* + *x*[*i*]; {1}
i:= *i* + 1
until *i* = 20

Договоримся о распределении памяти для массива x и S

S – 0100
x[0] – 0102
x[1] – 0104
. . .
x[19] – 0128

Пусть для накопления суммы в машинной программе будет использоваться регистр R5. Тогда в машинной реализации приведённого алгоритма на первом шаге цикла вместо оператора {1} нужно выполнить команду 01 5 0 0102. На втором шаге к накопленной сумме R5 следует прибавить значение x[1], то есть число по адресу 0104. Значит, нужно выполнить команду 01 5 0 0104. На третьем шаге нужно прибавлять число с адресом 0106, и т.д. На каждом шаге адрес операнда увеличивается на 2, поскольку в УМ-Р числа занимают по две ячейки. На последнем шаге цикла вторым слагаемым команды будет число с адресом 0128.

Таким образом, команда сложения в теле цикла должна меняться в процессе работы цикла. Об изменении этой команды должна позаботиться сама программа. Как это сделать? Вспомним, что команда программы хранится в ячейках памяти, так же как и любое число. Мы можем прибавить к содержимому этих ячеек (т.е. к команде) число 2, при этом команда изменится, адрес второго операнда увеличится.

Программа, которая в процессе работы изменяет свой код, называется ***самомодифицирующейся***.

Программа

Адрес	Содержимое ячейки	Комментарий
0000	00 2 0 0015	*R2*:= 2 Константы
0002	00 F 0 0017	*RF*:= последний шаг
0004	00 B 0 0008	*RB*:= копия команды [0008]
0006	20 A B	*RA*:= текущая команда [0008]
0007	22 5 5	*R5*:= 0 (*S*)
0008	01 5 0 0102	***Rpt***: *S*:= *S* + *x*[*i*];

000A	21 A 2	модификация
000B	10 A 0 0008	команды [0008]
000D	25 A F	последний шаг?
000E	82 0 0 0008	if ≠, goto ***Rpt***
0010	10 5 0 0100	$R5 \rightarrow S$
0012	10 B 0 0008	восстановление [0008]
0014	99 0 0	стоп
0015	0000 0002	2_{10}
0017	01 5 0 0130	[0008] в конце цикла

Всегда во время работы программы регистр RA и ячейка с адресом 0008 содержат команду прибавления к регистру R5 текущего элемента массива. Происходит это так. В начале программы в RA записывается содержимое ячейки 0008, команда прибавления к R5 самого первого элемента массива. На первом шаге цикла выполняется именно эта команда. Затем, при переходе к следующему шагу цикла, в RA генерируется новая команда, настроенная на очередной элемент массива, после чего полученная команда записывается непосредственно в программу, в ячейку с адресом 0008.

Проверка завершения работы реализована так. Легко вычислить, что после того, как будет обработан последний элемент массива, команда примет вид 01 5 0 0130. Это число было записано в регистр RF. На каждой итерации цикла значение RA сравнивается с RF, цикл завершается, когда достигается равенство.

Поскольку команда по адресу 0008 изменяется в процессе работы цикла, после завершения цикла она не настроена на обработку массива, начиная с первого элемента, поэтому программу нельзя запустить выполняться ещё раз. Для того, чтобы программу можно было выполнять несколько раз, после завершения вычислений нужно восстанавливать первоначальный текст программы. В примере это действие реализовано следующим образом: до начала работы цикла в регистре RB был сохранён исходный вариант команды 0008; после окончания работы цикла первоначальный вариант изменявшейся команды 0008 восстанавливается из RB.

Какие выводы можно сделать, проанализировав пример самомодифицирующейся программы?

Прежде всего, отладка таких программ намного сложнее, чем отладка обычных программ, поскольку текст программы динамически изменяется. Кроме статических ошибок, изначально имевшихся в программе, появляется класс ошибок, возникающих вследствие изменения программы во время её выполнения. Если статические ошибки можно обнаружить, тщательно проанализировав текст программы, динамические ошибки появляются в выполняемой программе, их нет в исходном тексте. Более того, ошибки могут возникать не при каждом запуске программы, а только при обсчёте некоторых из начальных данных. Чтобы поймать такую ошибку, нужно

получить как бы покадровые снимки текста программы – полный текст программы на каждой итерации цикла.

Второй момент связан с быстродействием. Как видно в приведённой программе, чтобы прибавить элемент массива к регистру, нужно не только взять из оперативной памяти сам элемент, но ещё изменить команду и записать её в память. Получаем два обращения к памяти на каждом шаге цикла, не считая дополнительного сложения при модификации команды. Так что работа с массивами происходит довольно медленно.

Чтобы сделать обработку массивов удобной, безопасной и быстрой, был предложен механизм аппаратной модификации адресов, который мы рассмотрим в следующей модели ЭВМ.

П2. Суть идеи модификации адресов

Основная идея заключается в том, чтобы хранить изменяемую часть команды в регистре и при выполнении команды учитывать это изменение.

В примере с суммированием массива поступим следующим образом. Адрес элемента массива разделим на две части: адрес начала массива и расстояние от начала массива до текущего элемента. Адрес начала массива укажем в команде, расстояние от начала массива до текущего элемента будем хранить в регистре-модификаторе. При переходе от обработки одного элемента массива к следующему меняется расстояние от начала массива (0, 2, 4,…). Для перехода к следующему элементу массива достаточно увеличить значение регистра-модификатора. А теперь самое важное – изменим алгоритм выполнения машинной команды. Операндом команды будет не та ячейка, чей адрес указан в команде, а ячейка, адрес которой получается прибавлением значения регистра-модификатора к адресу, указанному в команде. Таким образом, при выполнении команды для получения реального адреса операнда адрес, указанный в команде, изменяется, модифицируется путём прибавлением содержимого регистра, отсюда и происходит название архитектуры и название самого регистра, используемого для изменения адреса. Вычисленный адрес реального операнда принято называть ***исполнительным адресом***.

Выполнение команды процессором разделяется на два шага:

1. Вычислить адрес реального операнда (прибавить значение регистра-модификатора к адресу в команде) – получить исполнительный адрес.
2. Выполнить требуемую машинную операцию с ячейкой, расположенной по исполнительному адресу.

Далее рассмотрим, как реализуется описанная идея в модели вычислительной машины.

П3. Описание учебной машины с модификацией адресов УМ-М и пример программирования для неё

Машина с модификацией адресов близка к машине УМ-Р. Процессор и оперативная память в УМ-М устроены так же. Число занимает две ячейки памяти. Форматы и семантика команд вида регистр-регистр и команд перехода остались те же, что в УМ-Р.

Ключевое отличие УМ-М от УМ-Р состоит в формате и семантике команд вида регистр-память. Команда приобретает вид КОП R M A. Здесь КОП, как обычно, код операции, R и M – номера регистров (R – первый операнд, M – модификатор), A – адрес второго операнда. Команда выполняется в два шага:

1. Вычисление исполнительного адреса по формуле

$$A_{исп} = \begin{cases} (A + [M])\ mod\ 16^4, & \text{если M не } R0 \\ A, & \text{если M есть регистр } R0 \end{cases}$$

2. Выполнение операции, задаваемой КОП, причём вторым операндом является ячейка памяти, расположенная по адресу $A_{исп}$.

Запись [M] обозначает, что в формуле используется не номер регистра M, а содержимое этого регистра. Формула говорит о том, что для получения реального адреса второго операнда нужно сдвинуться от ячейки с адресом A на [M] ячеек.

Исполнительный адрес вычисляется по модулю 16^4. Это означает, что ячейки оперативной памяти как бы образуют кольцо: за ячейкой с адресом $FFFF_{16}$ идёт ячейка 0000_{16}. Например, если $R6=20_{16}$, то для команды 01 3 6 FFFF второй операнд берётся из ячейки с адресом $001F_{16}$.

Если в качестве второго регистра указан регистр с номером 0, модификации адреса не происходит. Таким образом, регистр R0 не может быть модификатором.

В машине УМ-М есть специальная команда для работы с исполнительным адресом: 11 R M A. Команда вычисляет исполнительный адрес и записывает его в первый операнд, т.е. реализует операцию $R:= A_{исп}$.

Рассмотрим, как для машины УМ-М реализуется алгоритм вычисления суммы элементов массива x[0..19].

Пример 1. Вычислить S – сумму элементов массива x[0..19].

Алгоритм

```
S:= 0; i:= 0;
repeat  S:= S + x[i];
        i:= i + 1
until   i = 20
```

Распределение памяти

```
   S  –  0100
x[0]  –  0102
x[1]  –  0104
   . . .
x[19]  –  0128
```

Для доступа к элементам массива в качестве модификатора будем использовать регистр R6. Поскольку адреса соседних элементов отличаются на 2, при переходе к очередному элементу нужно прибавлять к R6 число 2. Заметим, что значение модификатора не совпадает с номером элемента, в отличие от переменной *i*, поэтому приведённый алгоритм непосредственно не переводится в машинную программу.

Не будем заводить специальную переменную или отводить регистр для хранения номера элемента *i*. Вместо этого для управления циклом используем модификатор R6. Цикл завершается, когда R6 будет указывать за последний элемент массива, то есть значение регистра станет равным $20*2=40_{10}$.

Программа

Адрес	Содержимое ячейки	Комментарий
0000	00 2 0 0002	*R2*:= 2 Константы
0002	00 F 0 0010	*RF*:= 40
0004	22 5 5	*R5*:= 0 (*S*)
0005	22 6 6	*R5*:= 0 модификатор
0006	01 5 6 0102	***Rpt***: *S*:= *S* + *x*[*R6*];
0008	21 6 2	R6:= R6+2
0009	25 6 F	R6=40?
000A	82 0 0 0006	if ≠, goto ***Rpt***
000C	10 5 0 0100	*R5* → *S*
000E	99 0 0	стоп
000F	0000 0002	2_{10}
0010	0000 0028	40_{10}

Как и следовало ожидать, мы получили более короткую программу, чем самомодифицирующаяся программа из П.1, основная часть – цикл суммирования – занимает всего шесть ячеек и использует на каждом шаге только одно обращение к памяти (за элементом *x*[*i*]). Текст программы стал наглядным, ближе к тексту алгоритма на языке высокого уровня.

Заметим, что модифицироваться может адрес операнда любой команды, а не только адреса операндов арифметических команд. А именно, можно модифицировать адрес в командах перехода. Это обстоятельство позволяет получить интересную реализацию переключателя (оператора case языка паскаль). Рассмотрим пример.

Пример 2. Расстояние задано в виде пары чисел (k, u), где k есть величина, u – единица измерения. Единица измерения закодирована числами 0 – сантиметр, 1 – дециметр, 2 – метр, 3 – километр. Определить *l* – расстояние в сантиметрах.

Реализуем следующий алгоритм:

case *u* ***of***
  0: *l*:= *k**1;
  1: *l*:= *k**10;
  2: *l*:= *k**100;
  3: *l*:= *k**100000;
end

Распределение памяти

k – 0100
u – 0102
l – 0104

Значение *l* будем вычислять в регистре R3. Каждая ветвь case реализуется последовательностью операций: записать в R3 значение k, умножить R3 на соответствующий коэффициент, перейти на конец оператора case. Для сокращения текста программы вынесем из ветвей общие действия – запись k в R3. Поскольку машинные реализации ветвей состоят из одинакового количества машинных команд, несложно по значению u получить адрес начала соответствующей ветви (адрес = начальный адрес + u*длина ветви). Второе слагаемое вычислим в регистре R6. Для перехода на начало ветви используем команду перехода с модификацией адреса по R6.

Программа

Адрес	Содержимое ячейки	Комментарий
0000	00 3 0 0100	*l*:= *k*
0002	00 6 0 0102	*R6*:= *u*
0004	13 6 0 0019	*R6*:= *R6**4
0006	80 0 6 0008	goto ветвь ***case***
0008	13 3 0 001B	*l*:= *l**1 ***case***
000A	80 0 0 0016	goto ***End***
000C	13 3 0 001D	*l*:= *l**10
000E	80 0 0 0016	goto ***End***
0010	13 3 0 001F	*l*:= *l**100
0012	80 0 0 0016	goto ***End***
0014	13 3 0 0021	*l*:= *l**100000
0016	10 0 0 0104	***End***: *R3* → *l*
0018	99 0 0	стоп
0019	0000 0004	4 (длина ветви)
001B	0000 0001	1_{10}
001D	0000 000A	10_{10}
001F	0000 0064	100_{10}
0021	0001 86A0	100000_{10}

Можно сделать текст программы более наглядным, расположив константу-множитель в той ветви case, где она используется. Поскольку при

этом изменится длина реализации ветви, нужно изменить множитель в вычислении R6. Приходим к следующему решению:

Адрес	Содержимое ячейки	Комментарий
0000	00 3 0 0100	*l*:= *k*
0002	00 6 0 0102	*R6*:= *u*
0004	13 6 0 0023	*R6*:= *R6**6
0006	80 0 6 0008	goto ветвь ***case***
0008	13 3 0 000C	*l*:= *l**1 ***case***
000A	80 0 0 0020	goto ***End***
000C	0000 0001	1_{10}
000E	13 3 0 0012	*l*:= *l**10
0010	80 0 0 0020	goto ***End***
0012	0000 000A	10_{10}
0014	13 3 0 0018	*l*:= *l**100
0016	80 0 0 0020	goto ***End***
0018	0000 0064	100_{10}
001A	13 3 0 001E	*l*:= *l**100000
001C	80 0 0 0020	goto ***End***
001E	0001 86A0	100000_{10}
0020	10 0 0 0104	***End***: *R3* → *l*
0022	99 0 0	стоп
0023	0000 0006	6 (длина ветви)

Длина программы увеличилась на две ячейки, потому что в последней ветви добавилась команда безусловного перехода, необходимая для того, чтобы управление не попало на константу 100000.

П4. Заключение

Основное преимущество, которое даёт модификация адресов, конечно, – повышение надёжности программ. Сравнив тексты программ суммирования элементов массива для машин УМ-Р и УМ-М, видим, что текст программы второй машины более понятный, легко сопоставляется алгоритму, написанному на языке высокого уровня, к тому же вторая программа короче.

Про быстродействие модели УМ-М можно сказать следующее. Время выполнения команды регистр-память в УМ-М больше, чем в УМ-Р, поскольку в такт ЦП добавился этап вычисления исполнительного адреса. Однако посмотрим внимательней, как обрабатывается один элемент массива в УМ-Р: во-первых, требуется команда обработки самого элемента, во-вторых, нужно изменять эту команду в программе, на что тратится, по крайней мере, одно обращение к памяти. Ясно, что аппаратная мо-

дификация адреса выполняется быстрее, чем программная реализация этого же действия, при этом не требуется обращение к ОП для изменения команды. Следовательно, в модели УМ-М программы обработки массивов работают быстрее.

Однако процессор УМ-М сложнее процессора УМ-Р, так как в нём реализован более сложный алгоритм выполнения машинной команды.

Модификация адресов впервые была реализована в машине, построенной в 1949 году в Манчестерском университете под руководством Ф. Вильямса, при участии А.Тьюринга. Механизм модификации адресов применялся в упомянутых выше машинах БЭСМ-6 [12] и PEGASUS [17]. Практически во всех современных компьютерах используется аппаратная модификация адресов.

Заключение

Подведём некоторые итоги. Мы обсудили различные идеи компьютерной архитектуры на примере семи модельных вычислительных машин. Довольно подробно разобрали назначение и функционирование основных устройств ЭВМ – оперативной памяти и процессора; рассмотрели, как происходит основная работа вычислительной машины, а именно процесс выполнения программы.

Анализ и сопоставление различных модельных ЭВМ, с одной стороны, демонстрируют разнообразие архитектурных подходов, существующих в области компьютерной техники, с другой стороны, позволяют оценить достоинства и недостатки каждого из архитектурных решений. Следует заметить, что в реальных компьютерах зачастую одновременно используются несколько из рассмотренных архитектурных решений.

Разобранные примеры дают представление об особенностях программирования на машинном языке каждой архитектурной модели. Хотя в настоящее время не программируют непосредственно на машинных языках, вместо этого при необходимости используют машинно-зависимые языки, так называемые ассемблеры, при программировании на ассемблере конкретного процессора необходимо учитывать особенности его архитектуры; приёмы программирования на ассемблере вытекают из приёмов программирования на машинном языке.

Литература

1. Полунов Ю. Электронная, универсальная...
http://www.computer-museum.ru/frgnhist/universal_p.htm

2. Любимский Э.З., Мартынюк В.В., Трифонов Н.П. Программирование. М.:Наука, 1980

3. Виртуальный компьютерный музей
http://www.computer-museum.ru/histussr/4.htm (М-2)

4. Виртуальный компьютерный музей
http://www.computer-museum.ru/histussr/9.htm (Минск-32)

5. Виртуальный компьютерный музей
http://www.computer-museum.ru/histussr/ural1.htm

6. Таненбаум Э. Архитектура компьютера. СПб.:Питер, 2005

7. Хамахер К., Вранешич З., Заки С. Организация ЭВМ. СПб.:Питер, 2003

8. Рауль Рохас Архитектура вычислительных машин Конрада Цузе Z1 и Z3.
http://www.zib.de/zuse/Inhalt/Kommentare/Html/0688/index.html

9. Полунов Ю. Недолгая, но славная ERA.
http://www.computer-museum.ru/frgnhist/era_p.htm

10. Полунов Ю. ЭВМ "ВИХРЬ" и её окружение.
http://www.computer-museum.ru/frgnhist/vihr_p.htm

11. Урал-1. Материал из Википедии.
http://ru.wikipedia.org/wiki/%D0%A3%D1%80%D0%B0%D0%BB-1

12. БЭСМ-6. Материал из Википедии.
http://ru.wikipedia.org/wiki/%D0%91%D0%AD%D0%A1%D0%9C-6

13. Полунов Ю. Ни на что не похожая …
http://www.computer-museum.ru/frgnhist/turing_p.htm

14. Каршенбойм И. Стековые процессоры, или Новое – это хорошо забытое новое. //Компоненты и технологии №9 2003г.

15. Королёв Л.Н. Архитектура ЭВМ. М.: Научный мир, 2005

16. Дон-2Н. Материал из Википедии.
http://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BD-2%D0%9D

17. Полунов Ю. Пегас из конюшни Ферранти.
http://www.computer-museum.ru/frgnhist/polunov.htm

Содержание