АРХИТЕКТУРА ЭВМ. УЧЕБНЫЕ МАШИНЫ

Бордаченкова Е.А.

Пособие предназначено для студентов первого курса факультета ВМиК в поддержку курса "Архитектура ЭВМ и язык ассемблера".

В первом семестре Вы уже познакомились с понятием "алгоритм". Как Вы помните, алгоритм описывает действия, которые исполнитель произведёт над объектами для достижения некоторой цели. В этом пособии мы рассмотрим пример одного возможного исполнителя, а именно электронно-вычислительную машину (ЭВМ, или компьютер). Мы разберем, как устроен компьютер и как именно он выполняет алгоритм.

§1. Принципы фон-Неймана.

П.1. Свойства ЭВМ.

Целью создания первых вычислительных машин было облегчить, упростить громоздкие арифметические вычисления, которые приходилось выполнять при решении физических и инженерных задач. Для того чтобы производство вычислительной машины экономически оправдало себя, нужно чтобы

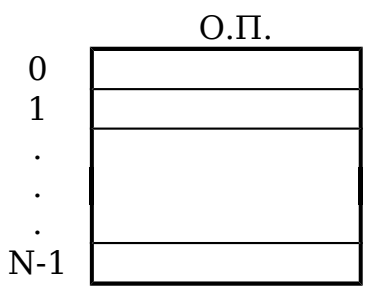
1. машина была универсальной - пригодной для решения не одной конкретной задачи, а целого класса задач.
2. машина должна обладать достаточным быстродействием. (Быстродействие - скорость вычислений). Чем выше быстродействие, тем больше задач решает машина за фиксированный отрезок времени. Тем эффективнее работа машины.
3. стоимость производства машины не должна быть очень большой.

П.2. Принципы фон-Неймана.

В 1943 г. американский математик Джон фон-Нейман описал как, по его мнению, должна быть устроена машина для вычислений. Сформулированные им принципы получили

название "принципов фон-Неймана", а машины, построенные в соответствии с ними, стали называть фон-Неймановскими. Большинство современных ЭВМ являются фон-Неймановскими.

Основными частями ЭВМ являются процессор и память. Процессор управляет работой компьютера; обеспечивает выполнение программ. Память (оперативная память) служит для хранения данных и программы во время работы компьютера.



Оперативная память состоит из элементов - ячеек. Каждая ячейка имеет свой номер, который называется адресом. Если в памяти N ячеек, они нумеруются от 0 до N-1. Ячейка состоит из разрядов. Каждый разряд содержит одну двоичную цифру (0 или 1). Количество разрядов во всех ячейках одинаково и называется разрядностью машины.

Принципы фон-Неймана.

1. Линейная организация памяти.

Ячейки памяти располагаются последовательно по возрастанию номеров.

2. Прямой доступ к элементам памяти.

Доступ к ячейке осуществляется по её адресу, в каждый момент работы компьютера можно обратиться к любой ячейке памяти.

Этот принцип обеспечивает облегчение программирования, удобство и надежность использования ЭВМ. (Вспомните машину Тьюринга. Для доступа к ячейке, отстоящей, например, на три ячейки правее данной, требовалось вводить три дополнительных состояния.)

3. Использование двоичной системы для хранения и обработки информации.

Этот принцип следует прежде всего из практических соображений: довольно легко с помощью электронных устройств реализовать два возможных состояния - 0 и 1.

4. Принцип хранимой программы.

Программа, управляющая процессом вычислений, хранится в памяти машины.

Этот принцип обеспечивает универсальность ЭВМ. (Сравним с машиной Тьюринга: каждая машина Тьюринга имела одну программу и могла решать только одну задачу!)

5. Машинные операции.

Существует набор действий по обработке данных, выполняемых аппаратно (реализованных в виде электронных схем). Эти действия называются машинными операциями.

Чем больше машинных операций, тем легче программировать для ЭВМ и тем выше ее быстродействие. (Вспомните, чтобы прибавить 1 к числу с помощью МТ, требовалось написать достаточно объемную программу.)

Каждой машинной операции соответствует машинная команда - последовательность нулей и единиц, которую может понять и выполнить процессор.

Таким образом, содержащиеся в ячейке памяти нули и единицы могут изображать данное, а могут являться командой. Что же именно записано в ячейке - данное или команда - определяется во время работы ЭВМ. В дальнейшем мы обсудим подробнее этот вопрос.

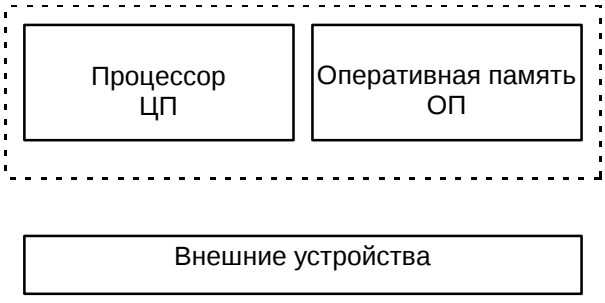
Итак, команда - это приказ процессору выполнить машинную операцию. Последовательность команд называется программой.

6. Последовательное исполнение команд.

Команды, записанные в памяти компьютера, выполняются последовательно, друг за другом.

§2. Структура ЭВМ.

Вычислительная машина состоит из следующих компонент



Назначение компонент.

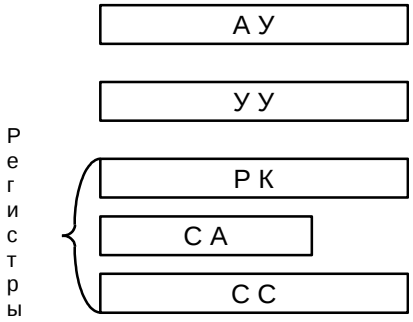
Процессор	– управляет работой ЭВМ, обеспечивает выполнение программ.
Оперативная память	– используется для хранения данных и программ во время работы ЭВМ
Внешние	– служат для связи ЭВМ с внешним миром

устройств
а

Рассмотрим структуру и работу каждой из компонент.

П.1. Процессор. Такт работы процессора.

Процессор включает в себя следующие устройства.



ЦП

АЛУ (арифметическое устройство) – оно выполняет арифметические и логические операции (например, сложение, вычитание, умножение)

УУ (устройство управления) – управляет работой процессора

Регистры - специальные ячейки, которые находятся в ЦП.

РК (регистр команды) содержит машинную команду, которую выполняет в данный момент процессор.

СА (счетчик адреса) содержит адрес следующей команды.

СС (слово-состояние) содержит информацию о результате выполнения команды.

Выполнение процессором одной машинной команды назовём тактом работы процессора.

Рассмотрим, как процессор выполняет машинную команду на примере команды

01101111001011

КОП A1 A2 A3

Пусть 01 изображает код операции (КОП) "сложение"; A1, A2, A3 - адреса первого операнда, второго операнда и результата соответственно.

1. В РК считывается из ОП команда, адрес которой записан в СА.

2. Содержание СА увеличивается на 1, так что теперь в СА получился адрес следующей команды программы.

3.УУ анализирует содержимое РК и организует выполнение команды.

Выделяется КОП. Определяется, что надо выполнить операцию "сложение". Определяются А1, А2, А3. Содержимое ячеек ОП с адресами А1, А2 пересылаются в АЛУ. Далее АЛУ выполняет действие сложение. Результат сложения из АЛУ пересылается в ячейку памяти с адресом А3. В регистр СС записывается информация об удачном (или неудачном) окончании выполнения сложения.

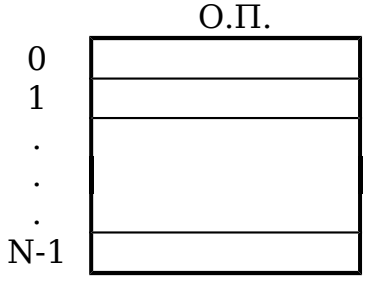
Далее работа повторяется с первого шага.

Ранее мы отмечали, что ячейка ОП может содержать данное или команду. Теперь понятно, как процессор отличает данное от команды: если адрес ячейки встретился в команде как адрес операнда, процессор обрабатывает содержимое ячейки как данное; если адрес ячейки получился в счетчике адреса, процессор обрабатывает содержимое ячейки как команду. Содержимое одной и той же ячейки в один момент работы процессора может трактоваться как данное, а в другой - как команда.

Перед началом работы процессора в регистр СА записывается аппаратно всегда один и тот же адрес, и первая команда программы должна располагаться в ОП в ячейке именно с этим адресом.

П.2. Оперативная память.

Мы касались устройства оперативной памяти в §1. Напомним основные сведения.



Оперативная память (ОП) состоит из ячеек. Каждая ячейка имеет свой адрес - число от 0 до N-1. Количество ячеек (N) называется объемом ОП. Заметим, что объем ОП и размер регистра СА взаимосвязаны: количество разрядов в СА должно быть достаточно для хранения любого возможного адреса. (Наибольший возможный адрес в нашем предположении N-1.)

Ячейки состоят из разрядов. Количество разрядов во всех ячейках одинаково и называется разрядностью машины. Каждый разряд содержит одну двоичную цифру. Иногда разряды называются битами. В программировании слово "бит" используют в двух смыслах:

Бит - один двоичный разряд ячейки.

- содержимое одного двоичного разряда.

Содержимое ячейки называют словом или машинным словом.

Содержимое ОП. Ячейки могут хранить данные и команды. Команды, составляющие программу, обычно располагаются в ОП последовательно, друг за другом. Что именно содержит ячейка - данное или команду - определяется в момент использования содержимого ячейки.

Работа ОП. Заметим, что ячейка ОП всегда имеет некоторое содержимое. В самом деле, электронное устройство, являющееся разрядом в ячейке, обязательно находится в каком-нибудь состоянии; это состояние изображает "0" или "1". Таким образом, ячейка заполнена нулями и единицами. Однако это содержимое не имеет никакого смысла. Для того, чтобы можно было обрабатывать данное, содержащееся в ячейке, надо сначала это данное записать в ячейку.

Итак, есть две основные операции работы с ОП:

1. Запись данного в ячейку.

ЦП сообщает ОП, чтó именно надо записать и по какому адресу. При записи в ячейку ее старое содержимое теряется, становится недоступным.

2. Чтение содержимого ячейки.

ЦП передает ОП нужный адрес. Содержимое ячейки с этим адресом считывается и пересылается в ЦП. При чтении содержимое ячейки не изменяется.

Чтение и запись в ОП производится специальными электронными схемами. При выключении вычислительной машины содержимое ОП теряется.

П.3. Внешние устройства.

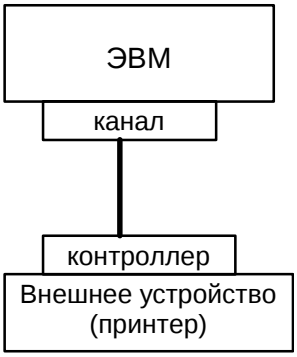
Внешние устройства служат для связи ЭВМ с окружающим миром. Обычно к ЭВМ подключаются клавиатура, монитор (экран, дисплей), внешние запоминающие устройства (жесткие диски, дисководы для гибких дисков) и мышь. В зависимости от того, для каких

целей используют ЭВМ, набор подключенных к ЭВМ устройств может сильно изменяться. К внешним устройствам относятся:

— принтер (печатающее устройство),
— устройство для работы с магнитными лентами,
— сканер (устройство для ввода графических изображений),
— модем (устройство для связывания компьютеров через телефонную сеть),
— устройство чтения с лазерных дисков

и другие специальные приборы.

Для подключения внешних устройств в ЭВМ имеются каналы.



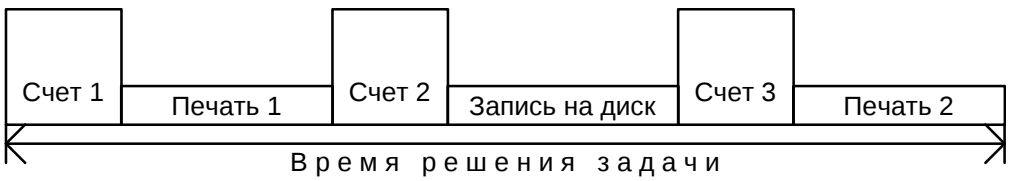
Канал - аппаратура и программное обеспечение (программы), занимающиеся передачей сигналов между ЭВМ и внешним устройством.

Во внешнем устройстве есть контроллер.

Контроллер - аппаратура и программное обеспечение, которое обрабатывает сигналы ЭВМ и подготавливает данные для устройства. Контроллер учитывает особенности работы своего внешнего устройства.

Внешние устройства работают намного медленнее процессора. Для того, чтобы процессор не простаивал во время работы внешнего устройства, организуется параллельная работа процессора и внешнего устройства.

Пусть процесс решения некоторой задачи состоит из трех отрезков счета (работы ЦП), двух печатей и одной записи данных на диск:



По окончании этапа Счет 1, когда получены данные для первой печати, принтер может заняться печатью, а ЦП может

продолжить решение задачи. После получения данных для диска, дисковод начинает записывать, а ЦП может продолжить работу. В итоге, общее время решения задачи сократится.

Параллельная организация работы устройств позволяет повысить эффективность использования ЭВМ, увеличить быстродействие, обеспечивая выполнение второго свойства вычислительных машин.

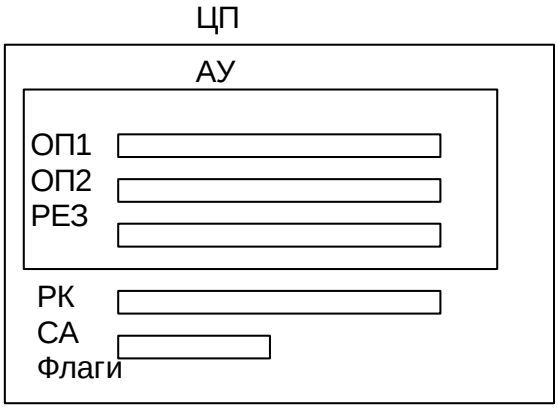
§3. Учебные машины.

Ранее мы рассмотрели общие принципы построения ЭВМ. В данном параграфе мы разберем 4 учебные вычислительные машины. Эти машины называются учебными, потому что они не существуют на самом деле. Однако они помогут нам представить работу реальных ЭВМ, разобрать некоторые проблемы, возникающие при построении ЭВМ, и способы решения этих проблем.

Для записи содержимого ячеек будем пользоваться шестнадцатиричными цифрами.

П.1. Учебная трехадресная машина УМ-3.

1. Структура процессора.



Кроме тех регистров, которые мы обсуждали в §2, в арифметическом устройстве УМ-3 имеются регистры первого операнда ОП1, второго операнда ОП2 и результата РЕЗ.

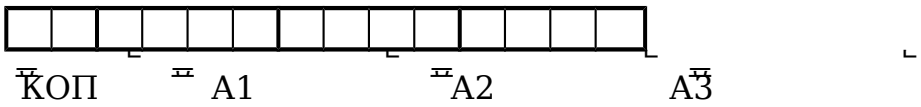
2. Оперативная память.

Ячейка - 14 шестнадцатиричных разрядов.

Объем ОП - 16^4 ячеек с адресами 0000$_{16}$-FFFF.

Представление чисел: число содержится в одной ячейке; отрицательные числа записываются в дополнительном коде.

Представление команд: команда занимает одну ячейку. Разряды ячейки имеют следующий смысл (формат команд):



Здесь КОП – код операции (указывает, какую машинную операцию надо выполнить); A1, A2, A3 – адреса первого, второго операндов и результата. Разрядность адресов совпадает с разрядностью регистра СА и определяется объемом ОП (а именно, количеством возможных адресов). Количество разрядов в регистрах ОП1, ОП2, РЕ3 и РК должно совпадать с количеством разрядов ячейки ОП, так как в ОП1, ОП2 и РЕ3 должны поместиться числа, а в РК – команда, которые занимают по одной ячейке ОП.

3. Система команд УМ-3.

Система команд ЭВМ – это набор всех машинных команд данной ЭВМ.

Название	КОП	Операция	Примечание
останов	99	стоп	A1,A2,A3 - любые
пересылка	00	[A1] $\circ$ [A3]	A2 - любой
арифметические			
сложение	01	[A1]+[A2] $\circ$ [A3]	Устанавливаются арифметические флаги
вычитание	02	[A1]-[A2] $\circ$ [A3]	
умножение со знаком	03	[A1]*[A2] $\circ$ [A3]	
без знака	13		
деление со знаком	04	[A1] div [A2] $\circ$ [A3], [A1] mod [A2] $\circ$ [A3+1]	

без знака	14	

переходы			
безусловный	80	перейти к А3	А1,А2 - любые
условные			После выполнения команды перехода работа продолжается с команды, записанной в ячейке с адресом А3
по =	81	при [А1]=[А2] перейти к А3	
по ≠	82	при [А1]≠[А2] перейти к А3	
по ≥	с/зн - 83 б/зн - 93	при [А1]≥ [А2] перейти к А3	
по ≤	с/зн - 84 б/зн - 94	при [А1]≤ [А2] перейти к А3	
по >	с/зн - 85 б/зн - 95	при [А1]> [А2] перейти к А3	
по <	с/зн - 86 б/зн - 96	при [А1]< [А2] перейти к А3	

В таблице использовано обозначение [А] – содержимое ячейки с адресом А.

Арифметические команды выполняются следующим образом: содержимое ячеек с адресами А1 и А2 пересылается в регистры ОП1 и ОП2, выполняется арифметическая операция (результат – в регистре РЕЗ), содержимое регистра РЕЗ пересылается в ячейку с адресом А3. Следует обратить внимание на то, что есть две команды умножения и две команды деления. Программист использует подходящую команду в зависимости от того, с какими данными (числами со знаком или без знака) должна работать программа. Команда деления дает два результата – частное и остаток. Они записываются в соседние ячейки: частное – по адресу А3, остаток – в ячейку с адресом А3+1.

Действие команд перехода заключается в изменении содержимого регистра СА. При этом на следующем такте работы ЦП будет выполнять команду, записанную в ячейке с адресом А3. (Вспомните §2.)

В командах условного перехода производится проверка соответствующего условия. Если условие выполняется, тогда в СА записывается адрес А3. Сравнение значений [А1] и [А2] выполняется так: в ОП1 записывается [А1], в ОП2 – [А2];

выполняется вычитание ОП1-ОП2, при этом получают значение все четыре арифметических флажка. По значениям флагов и определяется истинность условия. Некоторым условиям (например,<) соответствуют разные наборы значений флагов для разных данных – чисел со знаком и чисел без знака. Поэтому команды переходов для чисел со знаком и чисел без знака разные.

Рассмотрим на примерах соответствие между условиями и значениями флагов.

1) Переход по равенству. В результате вычитания [A1] - [A2] получится ZF=1 тогда и только тогда, когда [A1] = [A2].

2) Переход по "меньше", числа без знака.

[A1] < [A2] ⇔ [A1] - [A2] < 0 ⇔ CF=1.

3) Переход по "меньше", числа со знаком. Если вычитаются числа со знаком, то при [A1] < [A2], [A1] - [A2] < 0, возможны ситуации

а) результат вычислили правильно (OF=0) и он оказался отрицательным , значение SF=1,

б) результат вычислили неправильно (OF=1), при этом знак вычисленного значения противоречит знаку числа [A1] - [A2], вычисленный результат оказался положительным, то есть SF=0.

Таким образом, для чисел со знаком
[A1] < [A2] ⇔ [A1] - [A2] < 0 ⇔ (OF=1) *and* (SF=0) *or* (OF=0) *and* (SF=1), или более коротко OF ⊻ SF. При выполнении команды перехода по "меньше" для чисел со знаком процессор проверяет выполнимость условия OF ⊻ SF.

4. Разберем теперь несколько примеров программ для УМ-3. Условимся, что перед началом работы

1) в СА записывается 0100, то есть выполнение начинается с команды по адресу 0100,

2) в ячейки ОП уже записаны все данные, необходимые для работы программ (ввод и вывод не рассматриваем).

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

Поскольку в выражении есть операция вычитания, лучше работать с числами со знаком. Пусть данные располагаются в следующих ячейках

Адрес	Содержимое

0000	*a*
0001	32_{16} (=50_{10})
0002	*b*
0003	*x*
0004	рабочая (потребуется для деления)

Программа.

Адрес	Содержимое ячейки	Комментарий
0100	04 0000 0001 00 03	*a mod 50* ⟵ [0004]
0101	02 0004 0002 00 03	[0004] - *b* ⟵ *x*
0102	03 0003 0003 00 03	*x* ⟵ *x* ⟵ *x*
0103	99 0000 0000 00 00	стоп

Напомним, что деление даёт два результата, поэтому первая команда программы записывает в [0003] *x div 50* (это значение в дальнейшем не используется) и в [0004] *a mod 50*.

Пример 2. Вычислить

*S1 = max(a,b) * 20,*
S2 = min(a,b) div 3.

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.
```

Данные:

0000 *a*
0001 *b*
0002 14_{16} (=20_{10})
0003 3
0004 *S1*
0005 *S2*
0006 рабочая.

Будем считать, что работаем с числами без знака.
Программа:

Адрес	Содержимое ячейки	Комментарий	
0100	94 0000 0001 01 04	*if a ≠ b, go to* 0104	
0101	00 0001 0000 00 04	*S1 := b*	часть
0102	00 0000 0000 00 05	*S2 := a*	*then*
0103	80 0000 0000 01 06	*go to* 0106	
0104	00 0000 0000 00 04	*S1 := a*	часть
0105	00 0001 0000 00 05	*S2 := b*	*else*
0106	13 0004 0002 00 04	*S1 := S1 * 20*	
0107	14 0005 0003 00 05	*S2 := S2 div 3*	
0108	99 0000 0000 00 00	стоп	

Пример 3. Вычислить $p = n!$.

Алгоритм:

```
begin      p:=1; k:=2;
    while k ≠ n do
    begin      p := p * k;
        k := k + 1
    end
end.
```

В этой задаче могут возникнуть довольно большие положительные числа, поэтому будем работать с числами без знака.

Данные:

0000 *n*
00011
00022
0003 *p*
0004 *k*

Программа:

0100	00 0001 0000 00 03	*p :=1*
0101	00 0002 0000 00	*k := 2*

	04	
0102	95 0004 0000 01	*if k > n, go to*
	06	0106
0103	13 0003 0004 00	*p := p * k*
	03	
0104	01 0004 0001 00	*k := k + 1*
	04	
0105	80 0000 0000 01	*go to* 0102
	02	
0106	99 0000 0000 00	стоп
	00	

Заметим, что в машинной программе начало цикла *while k ≠ n do* заменилось на переход на конец программы по отрицанию условия. Тело цикла заканчивается безусловным переходом на начало цикла.

П.2. Учебная двухадресная машина УМ-2.

Анализ большого числа программ показывает, что только 25% команд содержат три различных адреса. В остальных командах либо два, либо все три адреса совпадают. Этот факт наталкивает на идею отказаться от одного из адресов — сделать команды двухадресными. Тогда можно использовать освободившиеся разряды для задания более длинных адресов — появляется возможность адресовать бóльший объем памяти. Либо можно сократить размер ячейки, сделав более дешевой оперативную память.

Как избавиться от третьего адреса? В УМ-3 существенно трехадресными были арифметические команды и команды условного перехода.

а) Арифметические команды. Договоримся, что результат записывается на место первого операнда (по адресу A1).

б) Команды условного перехода. Можно разбить команду на две, выделив в отдельную команду действие сравнения.

1. Описание УМ-2.

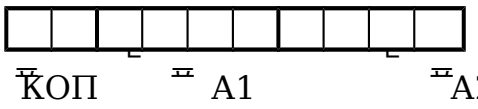
Структура процессора та же, что и в УМ-3; разрядность регистров соответствует разрядности ячейки и объему ОП.

Ячейка ОП — 10 шестнадцатиричных разрядов.

Объем ОП — 16^4 ячеек.

Представление чисел. Число занимает одну ячейку. Отрицательные числа записываются в дополнительном коде.

Формат команд. Команда занимает одну ячейку:



Система команд. Коды операций те же, что в УМ-3. Отличия заключаются в следующем.

Команды	Действие
Пересылка 00 A1 A2	[A1] := [A2]
Арифметические команды	[A1] := [A1] ⊏ [A2], где ⊏ ∈ {+, -, *}
Деление	вычисляет частное и остаток от деления [A1] на [A2]; частное записывается в [A1], остаток в [A1+1]
Сравнение 05 A1 A2	вычисляется [A1] - [A2], устанавливаются арифметические флаги; результат вычитания не сохраняется
Условные переходы КОП A1 A2	переход по A2, если условие выполняется; A1 не используется

Остальные команды те же, что в УМ-3.

2. Рассмотрим два примера программ для УМ-2.

Пример 1. Вычислить значение $x = (a \bmod 50 - b)^2$ по заданным a и b.

Распределение памяти:

0000 a
0001 x
$0002 32_{16}$ ($=50_{10}$)
0003 b

Программа:

0100	04 0000 00 02	*x := a mod* *50*
0101	02 0001 00	*x := x - b*

	03	
0102	03 0001 00 01	*x := x * x*
0103	99 0000 00 00	стоп

Пример 2. Вычислить *p = n!*.

Алгоритм:

```
begin      p:=1; k:=2;
      while k ⫣ n do
      begin      p := p * k;
            k := k + 1
      end
end.
```

Распределение памяти:

00001
00012
0002n
0003p
0004k.

Программа:

0100	00 0003 0000	*p := 1*
0101	00 0004 0001	*k := 2*
0102	05 0004 0002	*k ⫣ n ?*
0103	95 0000 0107	*k > n, go to* 0107
0104	13 0003 0004	*p := p * k*
0105	01 0004 0000	*k := k + 1*
0106	80 0000 0102	*go to* 0102
0107	99 0000 0000	стоп

П.3. Учебная машина с регистрами УМ-Р.

Вспомним, как выполняется арифметическая команда. Процессор обращается к ОП для получения операндов и для записи результата. Поскольку ОП внешнее по отношению к ЦП устройство, на это взаимодействие затрачивается бóльшая часть времени выполнения команды. Кроме того, часто оказывается, что результат, полученный ЦП в предыдущей

команде, требуется в качестве операнда для выполнения следующей команды. Принимая во внимание эти обстоятельства, имеет смысл включить в ЦП несколько ячеек памяти. Так как этих ячеек будет немного, можно сделать их более быстродействующими (и, видимо, более дорогими), чем ячейки ОП. Такие ячейки, расположенные в ЦП, называются регистрами. Программист может использовать эти регистры по своему желанию для хранения информации, в отличие от других регистров процессора. Поскольку регистров немного, для указания их в команде требуется меньше разрядов, чем для задания адресов ОП. Использование регистров позволяет сократить размер программ.

1. Описание УМ-Р.

Кроме обычных регистров, ЦП содержит 16 регистров общего назначения. Регистры нумеруются от 0 до F_{16}. Размер регистров совпадает с разрядностью чисел.

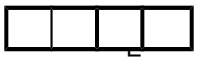
Ячейка ОП состоит из четырех шестнадцатиричных разрядов.

Объем ОП — 16^5 ячеек.

Число занимает две соседние ячейки, отрицательные числа записываются в дополнительном коде.

Команды бывают двух типов.

а) Регистр-регистр. Занимает одну ячейку. Формат команды



КОП R1 R2,

где R1 — первый операнд, R2 — второй операнд.

б) Регистр-память. Занимает две ячейки. Формат команды

КОП R1 A2,

где R1 — первый операнд, A2 — адрес второго операнда.

Система команд следующая:

Название	КОП	Регистр-память	КОП	Регистр-регистр
останов			99	стоп
пересылки	00	R1:=(A2,A2+1)	20	R1:=R2

	10	R1 ○(A2,A2+1)		
арифметические				
сложение	01	R1:=R1+ (A2,A2+1)	21	R1:=R1+R2
вычитание	02	R1:=R1-(A2,A2+1)	22	R1:=R1-R2
умножение				
со знаком	03	R1:=R1*(A2,A2+1)	23	R1:=R1*R2
без знака	13		33	
деление				
со знаком	04	R1:=R1 div (A2,A2+1), (R1+1):=R1 mod (A2,A2+1)	24	R1:=R1 div R2, (R1+1):=R1 mod R2
без знака	14		34	
сравнение	05	R1-(A2,A2+1)	25	R1-R2
переходы по A2 (типа регистр-память)(R1- любой)				
безусловный	80			
по =	81			
по ≠	82			
по <	с/зн - 83	б/зн - 93		
по ≤	с/зн - 84	б/зн - 94		
по >	с/зн - 85	б/зн - 95		
по ≥	с/зн - 86	б/зн - 96		

2. Примеры.

Для удобства чтения программы будем записывать, располагая по одной команде на строке. Тот факт, что команды имеют разную длину, будем учитывать при подсчете адресов команд.

Пример 1. Вычислить значение $x= (a \bmod 50 - b)^2$ по заданным a и b.

Распределение памяти:

00000 a
00002 b
00004 32_{16} $(=50_{10})$
00006 x

Программа:

00100	00 0 00000	$R_0 := a$
00102	04 0 00004	$R_1 := a \bmod 50$

00104	02 1 00002	$R_1 := R_1 - b$
00106	23 1 1	$R_1 := R_1 * R_1$
00107	10 1 00006	$x := R_1$
00109	99 0 0	стоп

Пример 2. Вычислить

$S1 = max(a,b) * 20,$

$S2 = min(a,b) div 3.$

Алгоритм:

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

Данные:

00000 *a*
00002 *b*
00004 14_{16} ($=20_{10}$)
00006 3
00008 *S1*
0000A *S2*

Значение *S1* будем вычислять в регистре R_1, значение *S2* — в регистре R_0 .

Программа:

00100	00 0 00000	$R_0 := a$
00102	00 1 00002	$R_1 := b$
00104	25 0 1	$R_0 > R_1?$
00105	96 0 0010A	$R_0 \nmid R_1$ go to 10A
00107	20 2 0	
0010	20 0 1	

8		
0010	20 1 2	$R_0 \leftrightarrow R_1$
9		
0010	13 1 00004	$R_1 := R_1 * 20$
A		
0010	10 1 00008	$S1 := R_1$
C		
0010	14 0 00006	$R_0 := R_0$ *div* 3
E		
0011	10 0 0000A	$S2 := R_0$
0		
0011	99 0 0	стоп
2		

Пример 3. Вычислить $p = n!$.

Напомним алгоритм:

```
begin     p:=1; k:=2;
      while k ≤ n do
      begin     p := p * k;
            k := k + 1
      end
end.
```

Распределение памяти:

00000	1
00002	2
00004	n
00006	p

Для ускорения работы цикла будем использовать регистры $R_1 \leftarrow 1$, $R_2 \leftarrow n$, $R_3 \leftarrow k$, $R_4 \leftarrow p$.

Программа:

00100	00 1 00000	$R_1 := 1$
00102	00 2 00004	$R_2 := n$
00104	20 4 1	$p := 1$
00105	00 3 00002	$k := 2$
00107	25 3 2	$k \le n$?
00108	95 0 0010E	$k > n$, *go to* 0010E
0010A	33 4 3	$p := p * k$
0010B	21 3 1	$k := k + 1$
0010C	80 0 00107	*go to* 0107

0010E	10 4 00006	сохранили *p*
00110	99 0 0	стоп

По сравнению с программой для УМ-2, потребовалась подготовительная работа по записи начальных значений в регистры (загрузка регистров). Однако основной цикл выглядит очень коротким, да и весь текст программы сократился (если считать длину в шестнадцатиричных разрядах, а не в ячейках).

Пример 4. Вычислить сумму элементов массива $x_1, \ldots, x_{20}$..

$$S = x_1 + x_2 + \ldots + x_{20}$$

Алгоритм:

```
begin    S := 0; i := 1;
    repeat  S := S + x[i];
            i := i + 1
    until i = 21
end.
```

Распределение памяти:		Используемые регистры:	
00000	x_1	R_0	0
00002	x_2	R_1	1
* * *		R_2	2
00026	x_{20}	R_3	21
00028	S	R_4	i
0002A	0	R_5	S
0002C	1	R_6 , R_7	вспомогательные
0002E	15_{16} (=21_{10})		
00030	2		

Программа:

00100	00 0 0002A	$R_0 := 0$	загрузка
00102	00 1 0002C	$R_1 := 1$	константных
00104	00 2 00030	$R_2 := 2$	регистров

00106	00 3 0002 E	$R_3 := 21$
00108	00 6 0010 D	$R_6 :=$ [0010D]
0010A	20 7 6	$R_7 := R_6$
0010B	20 5 0	$S := 0$
0010C	20 4 1	$i := 1$
0010D	01 5 0000 0	$S := S + x[1]$
0010F	21 6 2	подготовка программы к обработке
00110	10 6 0010 D	следующего элемента массива
00112	21 4 1	$i := i + 1$
00113	25 4 3	$i = 21?$
00114	82 0 0010 D	$i \leq 21$, go to 0010D
00116	10 5 0002 8	сохранили S
00118	10 7 0010 D	восстановили первоначальный вариант команды 0010D
0011A	99 0 0	стоп

При решении данной задачи возникла проблема: как добиться того, чтобы в разные моменты работы программы к S прибавлялись различные элементы массива? По сути, требуется, чтобы во время работы программы менялся адрес второго операнда в команде 0010D: 01 5 00000 ($S := S + x[1]$). Мы сделали это так. Регистр R_6 содержит копию команды 0010D. Когда команда 0010D выполнится, прибавим к R_6 число 2 (адреса соседних элементов массива отличаются на 2) и запишем новую команду из R_6 в программу по адресу 0010D. Таким образом, на следующем шаге цикла будет выполнена другая команда 0010D: 01 5 00002 ($S := S + x[2]$). это изменение текста программы производится командами 0010F, 00110. Команда 00118 восстанавливает первоначальный вариант команды 0010D. Если этого не сделать, после окончания работы программы команда 0010D будет содержать адрес ячейки, следующей за последним элементом массива x.

При этом повторный запуск программы приведет к неверным результатам.

Мы встретились в этом примере с программой, которая изменяет свой текст во время работы. Такие программы называются самомодифицирующимися. Их применение довольно опасно из-за непредсказуемых последствий возможных ошибок в изменении команд.

П.4. Учебная машина с модификацией адресов УМ-М.

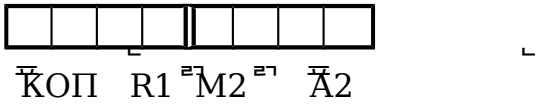
В программировании большое количество задач связано с обработкой массивов. Поэтому желательно сделать работу с массивами удобной. Решается эта проблема с помощью механизма модификации адресов. Суть этого механизма состоит в следующем. В команде кроме адреса операнда указывается регистр-модификатор. При выполнении команды процессор вычисляет сначала исполнительный адрес как сумму адреса, указанного в команде, и содержимого регистра-модификатора. Затем из ячейки с полученным адресом берется операнд. Изменив содержимое модификатора, мы заставляем процессор использовать другой адрес операнда. Подчеркнем, что действие по вычислению исполнительного адреса выполняется аппаратно, это дает преимущество в скорости по сравнению с самомодифицирующимся кодом (пример 4 из предыдущего пункта). Кроме того, текст программы остается всегда одним и тем же, так как изменение адреса получается за счет изменения содержимого регистра-модификатора. Это повышает надежность программ.

1. Описание учебной машины с модификацией адресов.

Машина УМ-М отличается от УМ-Р в следующем.

1) Объем ОП 16^4 ячеек.

2) Формат команд регистр-память



M2 — регистр, используемый для модификации адреса A2. Это может быть любой регистр общего назначения, кроме R_0

3) При выполнении команд регистр-память процессор вычисляет исполнительный адрес по правилу

$$A_{исп} = \begin{cases} A2, \text{если в команде на месте } M2 \text{ стоит } 0 \\ ([M2] + A2) \bmod 16^4, \text{если } M2 \neq 0 \end{cases}$$

Второй операнд берется из ячейки с адресом $A_{исп}$.

Пример 1. Вычислить сумму элементов массива $x_1, \ldots, x_{20}$..

$$S = x_1 + x_2 + \ldots + x_{20}$$

Распределение памяти:	Используемые регистры:
0000 x_1	R_0 0
0002 x_2	R_1 1
* * *	R_2 2
0026 x_{20}	R_3 21
0028 S	R_4 i
002A 0	R_5 S
002C 1	R_6 модификатор
002E 15_{16} (=21_{10})	
0030 2	

Программа:

0100	00 0 0 002A	$R_0 := 0$	загрузка
0102	00 1 0 002C	$R_1 := 1$	константных
0104	00 2 0 0030	$R_2 := 2$	регистров
0106	00 3 0 002E	$R_3 := 21$	
0108	20 6 0	$R_6 := 0$	
0109	20 5 0	$S := 0$	
010A	20 4 1	$i := 1$	
010B	01 5 6 0000	$S := S + x[R_6]$	
010D	21 6 2	$R_6 := R_6 + 2$	
010E	21 4 1	$i := i + 1$	
010F	25 4 3	$i = 21?$	
0110	82 0 0 010B	$i \neq 21$, go to 0010B	

0112	10 5 0 0028	сохранили S
0114	99 0 0	стоп

В этой программе модификация адресов происходит только в одной команде 010B. Отметим, что во время работы программы модификатор R_6 принимает значения 0, 2,4,...Значение R_6 увеличивается на 2 на каждом шаге цикла, так как элементы массива занимают по две ячейки.

Можно использовать модификатор R_6 для управления циклом. Тогда не потребуются команды, работающие с переменной i .

Пример 2. Вычислить сумму элементов массива $x_1, \ldots, x_{20}$..

$$S = x_1 + x_2 + \ldots + x_{20}$$

Распределение памяти:	Регистры:
00000	
0002 x_1	
0004 x_2	R_2 2
* * *	R_3 38
0026 x_{20}	R_5 S
0028 S	R_6 модификатор
002A 26_{16} (=38_{10})	
002C 2	

Программа:

0100	00 2 0 002E	$R_2 := 2$
0102	00 3 0 002C	$R_3 := 38$
0104	20 6 3	$R_6 := 38$ адрес последнего элемента
0105	00 5 0 002A	$S := 0$
0107	01 5 6 0000	$S := S + x\,[R_6]$
0109	22 6 2	$R_6 := R_6 - 2$
010A	82 0 0 010	$R_6 \neq 0$, *go to* 00107

	7	
010C	10 5 0 002 8	сохранили *S*
010E	99 0 0	стоп

В этой программе элементы массива просматриваются, начиная с последнего. Поэтому начальное значение R_6 — адрес последнего элемента. Для получения адреса предыдущего элемента из R_6 вычитается 2 на каждом шаге цикла. После завершения выполнения программы значение R_6 вышло за рамки массива и указывает на 0000.

При выполнении перехода по "не равно" процессор проверяет значение флага ZF (материал п1 §4). Команда 0109: 22 6 2 ($R_6 := R_6 - 2$) устанавливает такое же значение флага ZF, что и последовательность команд

22 6 2 ($R_6 := R_6 - 2$)

25 6 0 ($R_6 := 0$).

В самом деле, числа ($R_6 - 2$) и (($R_6 - 2$) - *0)* могут равняться нулю только одновременно.

Следовательно, команду сравнения можно опустить.