МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева - КАИ» (КНИТУ-КАИ)

Институт Компьютерных Технологий и Защиты Информации

(наименование института, в состав которого входит отделение СПО)

Отделение Колледж Информационных Технологий

(наименование отделения СПО)

Сабиров Н.А.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Для студентов специальностей 09.02.06 «Системное и сетевое администрирование»

По дисциплине ОП 02 Архитектура аппаратных средств

Казань, 2025

Оглавление

Введение

Предлагаемое учебно-методическое пособие предназначено для студентов, стремящихся закрепить теоретические знания по дисциплине «Архитектура ЭВМ» и овладеть методами и средствами обработки информации на языке Ассемблер. Кроме того, оно может быть использовано студентами, аспирантами и преподавателями в качестве справочника, в котором в краткой и ёмкой форме собраны наиболее важные сведения о микропроцессоре Intel 8086 и об основных командах языка Ассемблер.

Пособие разбито на 6 разделов, в каждом из которых приводятся задания для лабораторных работ, примеры выполнения этих заданий и команды языка Ассемблер, а также даются варианты заданий для самостоятельного выполнения.

Лабораторная работа №1.
Устройство и принципы работы микропроцессора Intel 8086

Цель работы: изучить устройство и принципы работы микропроцессора Intel 8086.

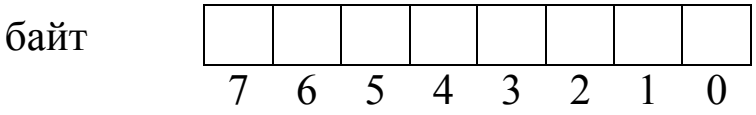
Общие теоретические сведения

Особенности персонального компьютера

Под термином «персональный компьютер» и сокращением ПК мы будем понимать только персональную ЭВМ, созданную на базе микропроцессоров семейства 80x86 фирмы Intel: 8086 (1978 г.), 80286 (1983 г.), i386 (1987 г.), i486 (1990 г.), Pentium (1993 г.). Именно к ним относятся наиболее широко распространенные в мире персональные компьютеры фирмы IBM и совместимые с ними. Все указанные процессоры объединяют в семейство 80x86, поскольку в них соблюдается преемственность: программа, написанная для младшей модели, может быть без каких-либо изменений выполнена на более старшей модели. Обеспечивается это тем, что в основе всех этих процессоров лежит система команд процессора 8086, в старшие же модели лишь добавляются новые команды. Таким образом, процессор 8086 – это база, основа для изучения всех остальных моделей данного семейства. Поэтому в дальнейшем под сокращением ПК будет пониматься персональный компьютер с процессором 8086.

1. Оперативная память

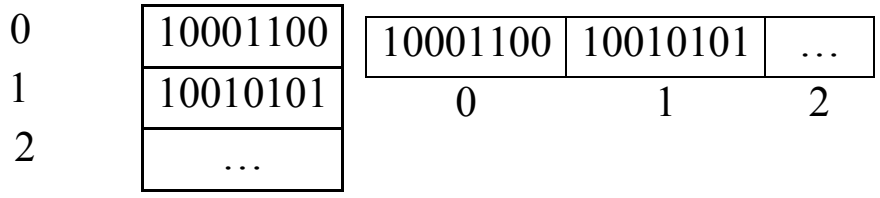
Оперативная память ПК делится на ячейки размером в 8 разрядов. Ячейки такого размера принято называть байтами (byte). Разряды байта нумеруются справа налево от 0 до 7:



При этом правые разряды (с меньшими номерами) называются младшими, а левые разряды – старшими. В каждом разряде может быть записана величина 1 или 0, такую величину называют битом (bit). Таким образом, содержимое любого байта – это набор из 8 битов, из 8 нулей и единиц. В конец двоичного числа принято добавлять букву “b”. Таким образом, мы можем определить, что 10101b - это двоичное число, которое соответствует десятичному значению 21.

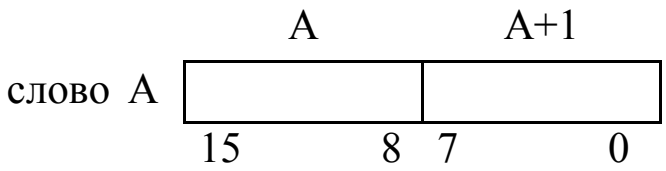
Ради краткости в дальнейшем будем записывать содержимое ячеек не в двоичной системе, а в шестнадцатеричной, указывая в конце букву “h” (hexadecimal - шестнадцатеричный). Например, если содержимом байта является 00010111, то будем записывать его как 17h (десятичное 23).

Байты нумеруются начиная с 0, порядковый номер байта называется его адресом. Объем оперативной памяти нашего ПК – 2^{20} байтов (1 Мб), поэтому для ссылок на байты памяти нужны 20-разрядные адреса – от 00000h до FFFFFh.[1]



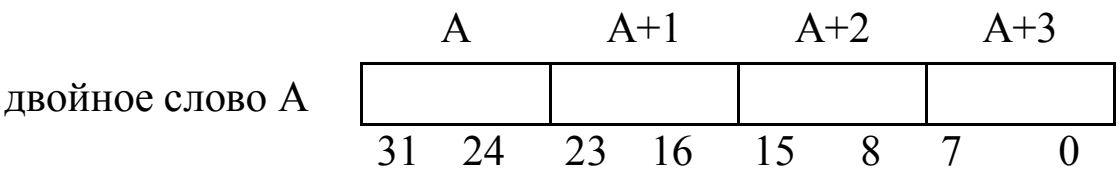
Байт – это наименьшая адресуемая ячейка памяти, также в ПК имеются и более крупные адресуемые ячейки: слова и двойные слова.

Слово (*word*) – это два соседних байта. Размер слова – 16 бит (разрядов). Они нумеруются, если рассматривать слово как единое целое, справа налево от 0 до 15. Адресом слова считается адрес его первого байта (с меньшим адресом).



[1] Отметим, что здесь и далее на всех рисунках, изображающих память, байты с меньшими адресами будут располагаться вверху (или слева), а с большими адресами – внизу (или справа).

Двойное слово (double word) – это четыре соседних байта или, что то же самое, два соседних слова. Размер двойного слова – 32 разряда, они нумеруются справа налево от 0 до 31. Адрес двойного слова – адрес первого из его байтов (с наименьшим адресом).



ПК может работать как с байтами, так и со словами и двойными словами, т.е. в ПК имеются команды, в которых ячейки этих размеров рассматриваются как единое целое. В то же время слова и двойные слова можно обрабатывать побайтно.

Данные разных типов имеют разные размеры, поэтому и нужны ячейки разных размеров. Например, байты используются для хранения небольших целых чисел (типа счетчиков) и символов. В виде же слов представляют обычные целые числа и адреса. Двойные слова используются для хранения больших чисел.

2. Регистры

Помимо ячеек оперативной памяти для хранения данных можно использовать и регистры – ячейки, расположенные в центральном процессоре. Доступ к регистрам осуществляется намного быстрее, чем к ячейкам памяти, поэтому использование регистров заметно уменьшает время выполнения программ.

Все регистры имеют размер слова (16 разрядов), за каждым из них закреплено определенное имя (AX, SP и т.п.). По назначению и способу использования регистры можно разбить на следующие группы:

- Регистры общего назначения (AX, BX, CX, DX, SI, DI, BP, SP);
- Сегментные регистры (CS, DS, SS,ES);

- Указатель команд (IP);
- Регистр флагов (Flags).

2.1 Регистры общего назначения

К этой группе относятся следующие 8 регистров:

- AX – accumulator, аккумулятор;
- BX – base, база;
- CX – counter, счетчик;
- DX – data, данные;

(буква X – от слова eXtended, расширенный: в процессоре 8080 были байтовые регистры A,B,C и D, но затем их расширили до размера слова)

- SI – source index, индекс источника;
- DI – destination index, индекс приемника;
- BP – base pointer, указатель базы;
- SP – stackpointer, указатель стека.

Несмотря на имя регистра, программист сам определяет, для каких целей использовать эти регистры. Их можно использовать в любых арифметических, логических и т.п. машинных операциях. Основное назначение регистра - хранение числа (переменной).

Регистры общего назначения AX, BX, CX, DX разделены на две части. К каждой части можно обращаться как к отдельному регистру. Например, если AX=0011000000111001b, то AH=00110000b, а AL=00111001b. Старший байт обозначается буквой "H"(high – выше, старший), а младший байт - буквой "L"(low – ниже, младший).

2.2 Сегментные регистры

Вторую группу регистров образуют следующие 4 регистра:

названия которых расшифровывается так:

CS–codesegment, сегмент команд;

DS – datasegment, сегмент данных;

SS – stacksegment, сегмент стека;

ES – extrasegment, дополнительный сегмент.

Ни в каких арифметических, логических и т.п. операциях эти регистры не могут участвовать. Можно только записывать в них и считывать из них, да и то здесь есть определенные ограничения.

Эти регистры используются для сегментирования адресов, которое является разновидностью модификации адресов и которое используется для сокращения размера команд.

Если в ЭВМ используется память большого объема, тогда для ссылок на ее ячейки приходится использовать «длинные» адреса, а поскольку эти адреса указываются в командах, то и команды оказываются «длинными». Это плохо, т.к. увеличиваются размеры машинных программ. Сократить размеры команд при «длинных» адресах можно следующим образом.

Любой адрес A можно представить в виде суммы B+D, где B – начальный адрес (база) того участка (сегмента) памяти, в котором находится ячейка A, а D – это смещение, адрес ячейки A, отсчитанный от начала этого сегмента (от B). Если сегменты памяти небольшие, тогда и величина D будет небольшой, потому что большая часть «длинного» адреса A будет сосредоточена в базе B.

Если в команде надо указать адрес A, тогда записываем базу B в какой-нибудь регистр S, а в команде вместо A указываем этот регистр и смещение D. Поскольку для записи D надо меньше места, чем для адреса A, то тем самым уменьшается размер команды. Благодаря модификации адресов данная команда будет работать с адресом, равным сумме D и содержимого регистра S, т.е. с нужным нам адресом A.

Рассмотренный способ задания адресов в командах называется *сегментированием* адресов, а регистры, используемые для хранения начальных адресов сегментов памяти, - *сегментными*. В ПК в качестве сегментных регистров можно использовать не любой регистр, а только один из указанных четырех (CS, DS, SS, ES).

Например, если мы хотим получить доступ к памяти с физическим адресом 12345h, мы должны установить DS = 1230h и SI=0045h (т.к. физический адрес не помещается в одиночном регистре). Процессор вычисляет физический адрес, умножая значение сегментного регистра на 10h и прибавляя к полученному результату значение регистра общего назначения (1230h * 10h + 45h = 12345h). Адрес, сформированный с помощью двух регистров, называется реальным адресом.

Примечание. Для сегментных регистров в ПК приняты следующие соглашения: в регистре CS должен находится начальный адрес сегмента команд - той области памяти, где расположены команды программы; регистр DS должен указывать на начало сегмента данных, в котором размещаются данные программы; регистр SS должен указывать на начало области памяти, отведенный под стек. Если так и сделать, тогда при ссылках на эти сегменты (команд, данных и стека) можно явно не указывать в командах соответствующие сегментные регистры (CS, DS и SS), они будут подразумеваться по умолчанию.

2.3 Указатель команд.

IP (англ. Instruction Pointer) — регистр, содержащий адрес-смещение следующей команды, подлежащей исполнению. Регистр IP связан с CS в виде CS:IP, где CS является текущим кодовым сегментом, а IP — текущим смещением относительно этого сегмента (рисунок 1.1).

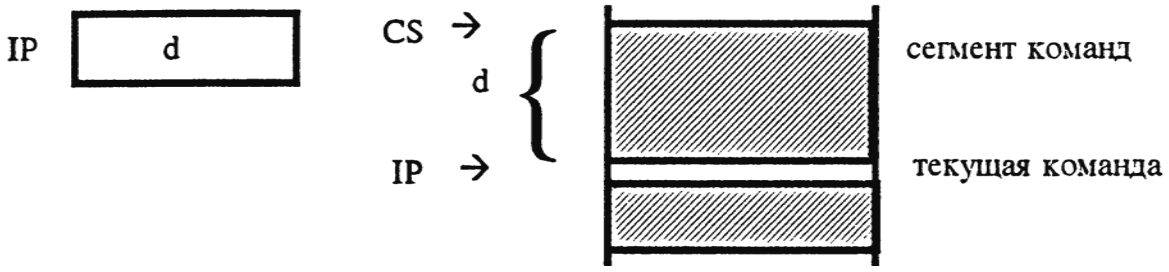


Рисунок 1.1 – Структура регистра IP

Принцип работы:

1. Например, CS содержит значение 2CB50h, в регистре IP хранится смещение 123h.
2. Адрес следующей инструкции, подлежащей исполнению, вычисляется путем суммирования адреса в CS (сегменте кода) со смещением в регистре IP:
   2CB50h + 123h = 2CC73h.

Таким образом, адрес следующей инструкции для исполнения равен 2CC73h.

При выполнении текущей инструкции процессор автоматически изменяет значение в регистре IP, в результате чего регистровая пара CS:IP всегда указывает на следующую подлежащую исполнению инструкцию.

2.4 Регистр флагов

В ПК имеется регистр флагов. *Флаг* – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и

значение 0 («флаг сброшен») в противном случае. В ПК используется 9 флагов, причем конструктивно они собраны в один 16-разрядный регистр, называемый регистром флагов и обозначаемый как *Flags*. Каждый флаг – это один из разрядов данного регистра (некоторые разряды не заняты):

Flags

				OF	DF	IF	TF	SF	ZF		AF		PF		CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Некоторые флаги принято называть флагами условий, они автоматически меняются при выполнении команд и фиксируют те или иные свойства их результата (например, равен ли он нулю), проверка этих флагов позволяет проанализировать результаты команд. Другие флаги называются флагами состояний, сами по себе они не меняются, и менять их должна программа, состояние этих флагов оказывает влияние на дальнейшее поведение процессора.

Флаги условий:

CF (carry flag) – флаг переноса. Наиболее полезен в арифметических операциях над числами без знака; например, если при сложении беззнаковых чисел получилась слишком большая сумма – с единицей переноса, которой нет места в ячейке, тогда флаг CF принимает значение 1, а если сумма «укладывается» в размер ячейки, то значением CF будет 0.

OF (overflow flag) – флаг переполнения. Полезен в арифметических операциях над числами со знаком; например, если при сложении или вычитании знаковых чисел получился результат, по модулю превосходящий допустимую величину (произошло переполнение мантиссы[2]), тогда флаг OF получает значение 1, а если переполнение мантиссы не было – значение 0.

[2] Мантисса - дробная часть логарифма числа.

ZF (zero flag) – флаг нуля. Устанавливается в 1, если результат команды оказался нулевым.

SF (sign flag) –флаг знака. Устанавливается в 1, если в операции над знаковыми числами получился отрицательный результат.

PF (parity flag) – флаг четности. Равен 1, если в 8 младших битах результата очередной команды содержится четное количество двоичных единиц.

AF (auxiliary carry flag) – флаг дополнительного переноса. Фиксирует особенности выполнения операций над двоично-десятичными числами.

Флаги состояний:

DF (direction flag) – флаг направления. Устанавливает направление просмотра строк в строковых командах: при DF = 0 строки просматриваются«вперед» (от начала к концу), при DF =1 – в обратном направлении.

IF (interrupt flag) – флаг прерываний. При IF = 0 процессор перестает реагировать на поступающие к нему прерывания, а при IF = 1 блокировка прерываний снимается.

TF (trap flag) – флаг трассировки. При TF = 1 после выполнения каждой команды процессор делает прерывание, чем можно воспользоваться при отладке программы – для ее трассировки.

Представление данных

Представление целых чисел

В ПК целые числа представляются байтом, словом или двойным словом[3]. К тому же делается различие между целыми числами без знака (неотрицательными) и со знаком. Это объясняется тем, что в ячейках одного и того же размера можно представить больший диапазон беззнаковых чисел, чем неотрицательных знаковых чисел. Например, в

[3] Если числа занимают иное количество байтов, то все операции над ними надо реализовывать самому программисту.

байте можно представить беззнаковые числа от 0 до 255, а неотрицательные знаковые числа – только от 0 до 127. Поэтому, если известно заранее, что некоторая числовая величина является неотрицательной, то выгоднее рассматривать ее как беззнаковую, чем знаковую.

Целые числа без знака

Беззнаковые числа могут быть представлены в виде байта, слова или двойного слова – в зависимости от их размера. Такие числа записываются в ячейки в двоичной системе счисления, занимая все разряды ячейки. Например, если для целого 98 отведен байт, то содержимым байта будет двоичное число 01100010 (62h), а если отведено слово, то оно будет иметь вид 0062h.

Поскольку в ячейке из k разрядов можно записать 2^k различных комбинаций из 0 и 1, то в виде байта можно представить целые от 0 до 255 ($=2^8-1$), в виде слова – целые от 0 до 65535 ($=2^{16}-1$), в виде двойного слова – целые от 0 до 4 295 967 295 ($=2^{32}-1$).

Отметим некоторую особенность представления чисел в ПК: числа размером в слово и двойное слово хранятся в памяти в «перевернутом» виде. Если на число отведено слово памяти, то старшие (левые) 8 битов числа размещаются во втором байте слова, а младшие (правые) 8 битов - в первом байте; в терминах шестнадцатеричной системы: первые две цифры числа хранятся во втором байте слова, а две последние цифры – в первом байте. Например, 98 = 0062h хранится в памяти так (A – адреса слова):

A	A+1
62	00

Зачем так сделано? Как известно, сложение и вычитание многозначных чисел мы начинаем с действий над младшими цифрами (например, при сложении чисел 1234 и 5678 мы сначала складываем цифры 4 и 8), а затем постепенно переходим к более старшим цифрам. С другой стороны, первые модели ПК (с процессором 8080) были 8-

разрядными, в них за раз можно было считать из памяти только один байт. Поскольку в этих условиях многозначное число нельзя считать из памяти сразу целиком, то в первую очередь приходится считывать байт, где находятся младшие цифры числа, а для этого надо, чтобы такой байт хранился в памяти первым. По этой причине в первых моделях ПК и появилось «перевернутое» представление чисел. В последующих же моделях, где уже можно было сразу считать из памяти все число, ради сохранения преемственности, ради того, чтобы ранее составленные программы могли без изменений выполняться на новых ПК, сохранили это «перевернутое» представление чисел.

Целые числа со знаком

Эти числа также представляются в виде байта, слова и двойного слова. Как байт можно представить числа от -128 до 127, как слово – от -32768 до 32767, как двойное слово – от -2147483648 до 2147483647.

В ПК знаковые числа записываются в *дополнительном коде*: неотрицательное число записывается так же, как и беззнаковое, а отрицательное число x представляется беззнаковым числом 2^k-|x|, где k – количество разрядов в ячейке, отведенное под число:

$$доп(x) = \begin{cases} x, если \quad x \geq 0, \\ 2^k, если \quad x < 0. \end{cases}$$

Например, дополнительным кодом числа +98 будет байт 62h или слово 0062h, а дополнительным кодом числа -98 – байта 9Eh (=158=256-98) или слово FF9Eh (2^{16}-98=10000h-62h).

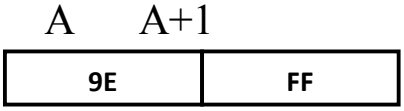
Приведем еще несколько примеров представления знаковых чисел в дополнительном коде (при ячейке размером в байт):

доп(0)	=0 =00000000	доп(-1)	=256-1=255 = 11111111
доп(1)	=1 =00000001	доп(-2)	=256-2 =254 = 00000001
доп(2)	=2 =00000010	доп(-3)	=256-3 = 253 = 11111101
доп(3)	=3 =00000011	доп(-126)	=256-126 = 130=10000010
доп(+126)	=126=01111110	доп(-127)	=256-127 =129 = 10000001

доп(+127)=127=01111111 | доп(-128) =256-128 = 128 =10000000

Из этих примеров видно, что в дополнительном коде самый левый бит играет роль знакового: для неотрицательных чисел он равен 0, а для отрицательных 1.

Как и беззнаковые, знаковые числа размером в слово и двойное слово записываются в памяти в «перевернутом» виде. Например, число -98 как слово будет храниться в памяти таким образом:



При этом знаковый бит оказывается во втором (правом) байте слова.

Представление символьных данных

Как и любая другая информация, символьные данные должны храниться в памяти ЭВМ в двоичном виде. Для этого каждому символу ставиться в соответствие некоторое неотрицательное число, называемое *кодом* символа, и это число записывается в память ЭВМ в двоичном виде. Конкретное соответствие между символами и их кодами называется системой кодировки.

В ЭВМ, как правило, используются 8-разрядные коды символов. Это позволяет закодировать 256 различных символов, чего вполне достаточно для представления многих символов, используемых в практике. Поэтому для кода символа выделяют в памяти один байт.

Представление команд

Машинные команды ПК занимают от 1 до 6 байтов. Код операции (КОП) занимает 1 или 2 первых байта команды. Команды ПК могут иметь от 0 до 2 операндов. Размер операндов – байт или слово (редко двойное).

Операнд может быть указан в самой команде (т.н. непосредственный операнд), либо может находиться в ячейке памяти, тогда в команде указывается адрес этой ячейки.

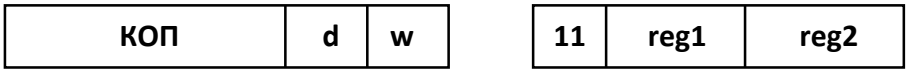
Некоторые команды требуют, чтобы их операнд находился в фиксированном месте (например, в AX), тогда операнд явно не указывается в команде. Результат выполнения команды помещается в регистр или ячейку памяти, откуда берется один из операндов. Например:

$$op1 := op1 * op2 ,$$

где *op1* – регистр или ячейка памяти, *op2* – непосредственный операнд, регистр или ячейка памяти, * - операция, заданная КОП.

Основные форматы машинных команд (с двумя операндами)

1. Формат «регистр- регистр» (2 байта):



Команды этого формата обычно описывают действие

*reg1:=reg1*reg2* или *reg2:=reg2*reg*,

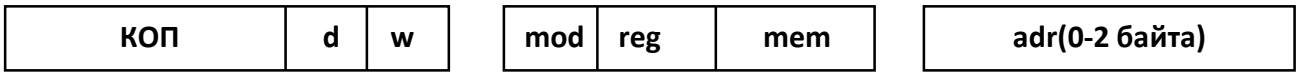
где *reg1* и *reg2* – регистры общего назначения. Поле КОП указывает на операцию (*), которую надо выполнить. Бит *w* определяет размер операндов, а бит *d* указывает, в какой из двух регистров записывается результат.

Во втором байте 2 левых бита фиксированы (для данного формата), а поля *reg1*, *reg2* указывают на регистры, участвующие в операции, согласно таблице 1.1:

Таблица 1.1 – Регистры команд

reg	w=1	w=0	reg	w=1	w=0
000	AX	AL	100	SP	AH
001	CX	CL	101	BP	CH
010	DX	DL	110	SI	DH
011	BX	BL	111	DI	BH

2. Формат «регистр-память» (2-4 байта):



Эти команды описывают операции *reg:=reg*adr* или *adr:=adr*reg*, где *reg* – регистр, а *adr* – адрес ячейки памяти. Бит w первого байта определяет размер операндов, а бит *d* указывает, куда записывается результат: в регистр (*d*=1) или в ячейку памяти (*d*=0).Трехбитовое поле reg второго байта указывает операнд-регистр (см. выше), двухбитовое поле mod определяет, сколько байтов в команде занимает операнд-адрес (00 – 0 батов, 01 – 1 байт, 10 – 2 байта), а трехбитовое поле mem указывает способ модификации этого адреса. В таблице 1.2 указаны правила вычисления исполнительного адреса в зависимости от значений полей mod и mem (a8 – адрес размером в байт, a16 – размером в слово, [r] – содержимое регистра r):

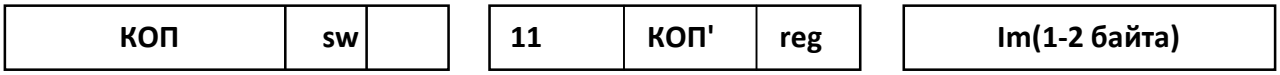
Таблица 1.2 – Правила вычисления исполнительного адреса

mod / mem	00	01	10
000	[BX]+[SI]	[BX]+[SI]+a8	[BX]+[SI]+a16
001	[BX]+[DI]	[BX]+[DI]+a8	[BX]+[DI]+a16
010	[BP]+[SI]	[BP]+[SI]+a8	[BP]+[SI]a16
011	[BP]+[DI]	[BP]+[DI]+a8	[BP]+[DI]+a16
100	[SI]	[SI]+a8	[SI]+a16
101	[DI]	[DI]+a8	[DI]+a16
110	a16	[BP]+a8	[BP]+a16
111	[BX]	[BX]+a8	[BX]+a16

Примечание:если в команде не задан адрес, то он считается нулевым. Если адрес задан в виде байта (a8), то он автоматически расширяется до слова (a16). Случай mod=00 и mem=110 указывает на отсутствие

регистров-модификаторов, причем адрес должен иметь размер слова. Случай mod=11 соответствует формату «регистр-регистр».

3. Формат «регистр-непосредственный операнд»(3-4 байта):



Команды этого формата описывают операции *reg:=reg*im* (im – непосредственный операнд). Бит *w* указывает на размер операндов, а поле *reg* – на регистр-операнд (см. выше). Поле КОП в первом байте определяет лишь группу операций, в которую входит операция данной команды, уточняет же операцию поле КОП' из второго байта. Непосредственный операнд может занимать 1 или 2 байта (в зависимости от значения бита *w*), при этом операнд размером в слово записывается в команде в «перевернутом» виде. Ради экономии памяти в ПК предусмотрен случай, когда в операции над словами непосредственный операнд может быть задан байтом (на это указывает 1 в бите s при *w*=1), и тогда перед выполнением операции байт автоматически расширяется до слова.

4. Формат «память-непосредственный операнд» (3-6 байтов):

Команды этого формата описывают операции типа *adr:=adr*im*. Смысл всех полей – тот же, что и предыдущих форматах.

Уже из рассмотренных форматов команд видно, что записывать машинные команды ПК в цифровом виде – вещь чрезвычайно неприятная. Поэтому нужен какой-то иной, более удобный способ записи команд и данных. И таким способом является язык ассемблера.

Порядок выполнения работы

Подготовиться по теоретическому материалу и ответить на контрольные вопросы.

Контрольные вопросы

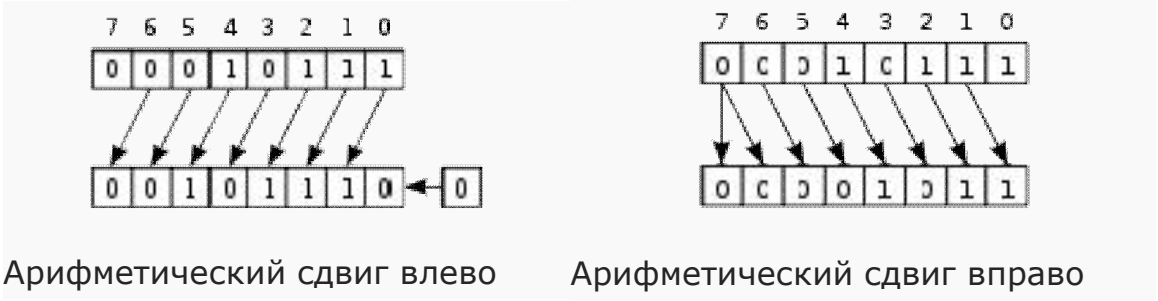
1. Как происходит организация оперативной памяти: представление данных в виде байта, слова, двойного слова?
2. Что такое регистр микропроцессора, основные группы регистров? Каково их назначение?
3. Какие существуют особенности представления и именования двухбайтовых регистров общего назначения?
4. Что такое сегментные регистры (метод "сокращения" адресов)?
5. Что такое флаг, и для чего он нужен?
6. Объясните назначение флагов переноса и нуля.
7. Объясните назначение флагов переполнения и знака.
8. В чем различие представления между целыми числами без знака (неотрицательными) и со знаком?
9. Поясните особенности хранения в памяти чисел размером в слово и двойное слово. С чем это связано?
10. Что называется дополнительным кодом числа? Приведите примеры.
11. Принципы хранения символьных данных в памяти компьютера?
12. Перечислите основные форматы машинных команд и дайте краткое описание каждой из них.

Лабораторная работа №2.
Вычисление простых формул на языке Ассемблер

Цель работы: реализовать программу для вычисления простой формулы на языке Ассемблер.

Краткие теоретические сведения

Арифметический сдвиг



Арифметический сдвиг влево Арифметический сдвиг вправо

При этом сдвиге слово рассматривается не просто как группа битов, а как целое число в дополнительном коде. При сдвиге влево ведёт себя как логический сдвиг, при сдвиге вправо уходящий бит исчезает, не влияя на оставшиеся биты, а на месте появившегося бита устанавливается бит, соответствующий знаку.

Пример работы операции сдвига:

Пусть у нас есть число 11111010b = −6 (в двоичной системе).

Если сделать сдвиг влево на 1 бит, то получим число 11110100b = −12.

Если сделать сдвиг исходного числа вправо на 1 бит, то получим число 11111101b = −3.

Легко заметить, что при арифметическом сдвиге сдвиг влево соответствует умножению на 2, а сдвиг вправо — делению на 2.

Например:

```
1011 = −5        1111 = −1
>>a 1            >>a 1
_____________
1101 = −3        1111 = −1
```

Схемотехническая реализация операций сдвига очень проста. Именно поэтому эти операции рекомендуют использовать для операций умножения и деления целых чисел на числа, равные степени 2 (2, 4, 8, 16, 32, 64 и т. д.) — если, конечно, такое округление отрицательных чисел не мешает.

Пример выполнения работы

Вычислить X=3A+(B+5)/2-C-1,

где A,B,C,X – целые знаковые числа занимающие слово. Написать программу, реализующую заданную формулу. Формат команд для процессора x8086 приведен в приложениях 1, 2, 3.

Распишем формулу по отдельным операциям:

```
AX←A                ; значение Ав регистре AX
AX← 2*(AX)          ; 2A в AX
AX←(AX)+A           ; 3A в AX
BX← B               ;B в BX
BX←5+(BX)           ;B+5 в BX
BX←(BX)/2           ;(B+5)/2 в BX
AX←(BX)+(AX)        ;3A+(B+5)/2 в AX
AX←(AX)-C           ;3A+(B+5)/2-C в AX
AX←(AX)-1           ;3A+(B+5)/2-C-1 в AX
X←(AX)              ;3A+(B+5)/2-C-1 в X
```

Ниже приведена программа, реализующая вычисления по заданной формуле.

```
MOV AX, A
SAL AX, 1
```

```
ADD AX, A
MOV BX, B
ADD BX, 5
SAR BX, 1
ADD AX, BX
SUB AX, C
DEC AX
MOV X, AX
HLT
A DW 10
B DW 20
C DW 5
X DW ?
```

Варианты заданий

Разработать программу, реализующую указанную формулу. Запустить программу с тремя – четырьмя наборами исходных данных и проверить правильность результатов.

Таблица 1 – Задания для самостоятельного выполнения

1. X=A-5(B-2C)+2	14. X=-(-(C+2A)4B+38)
2. X=-4A+(B+C)/4+2	15. X=A-3(A+B)+C mod 4
3. X=7A-2B-100+C	16. X=3(A-2B)+50-C/2
4. X=-A/2+4(B+1)+3C	17. X=(3A+2B)-C/4+217
5. X=5(A-B)-2C+5	18. X=3(C-2A)+(B-C+1)/2
6. X=(A/2+B)/4+C-1	19. X=(2A+B)/4-C/2+168
7. X=-(C+2A+4B+B)	20. X=6(A-2B+C/4)+10
8. X=6C+(B-C+1)/2	21. X=3(A-4B)+C/4
9. X=2-B(A+B)+C/4	22. X=-(-(C+2A)5B-27)
10. X=2B-1+4(A-3C)	23. X=A/2-3(A+B)+C*4
11. X=(2A+B)/4-C/2+168	24. X=3(A-2B)+50-C/2
12. X=6(A-2B+C/4)+10	25. X=5A+2B-B/4+131
13. X=5(A-B)+Cmod 4	

Лабораторная работа № 3.
Программирование разветвляющихся алгоритмов на языке Ассемблер

Цель работы: Освоить практические навыки программирования разветвляющихся алгоритмов на языке Ассемблер

Задания для самостоятельного выполнения

1. Составить программу для индивидуального задания на основе приведенного примера
2. Выполнить программу в среде EMU8086 .
3. Выполнить программу в двух возможных случаях при X≥N и X<N.
4. Составить аналогичную программу на языке C++ и проверить полученные результаты.

Порядок выполнения работы

Рассмотреть пример приведенный ниже, изучить структуру программы, используемые команды и функции и использовать их при решении индивидуального задания.

Пример. Составить программу на языке Ассемблер для вычисления кусочно-заданной функции:

$$Y = \begin{cases} AX - B, & если \quad X \geq 2 \\ CX + D, & если \quad X < 2 \end{cases}$$

```
include 'emu8086.inc'
ORG    100h
;GOTOXY 10, 2          ; установить курсор на 2 ряд и
                        10 столбец
LEA    SI, msgX        ; Запрос на ввод  X
CALL   print_string    ;
CALL   scan_num        ; ввод числа в CX.
MOV    X,CX
```

```
;GOTOXY 10,3
LEA    SI, msgA       ; Запрос на ввод  A
CALL   print_string   ;
CALL   scan_num       ; ввод числа в CX.
 MOV   A,CX
;GOTOXY 10,4
LEA    SI, msgB       ; Запрос на ввод B
CALL   print_string   ;
CALL   scan_num       ; ввод числа в CX.
MOV   B,CX
;GOTOXY 10,5
LEA    SI, msgC       ; Запрос на ввод C
CALL   print_string   ;
CALL   scan_num       ; ввод числа в CX.
MOV    C,CX
;GOTOXY 10,6
LEA    SI, msgD       ; Запрос на ввод D
CALL   print_string   ;
CALL   scan_num       ; ввод числа в CX.
MOV   D,CX
CMP X,2
JGE VAG1
MOV AX,C
IMUL  X
ADD AX,D
JMP VAG
VAG1: MOV AX,A
IMUL X
SUB AX,B
VAG: ;GOTOXY 10,8
CALL   pthis
DB  13, 10, 'Otvet: ', 0
CALL   print_num      ; Ввод числа в AX.
RET                  ; Возврат в операционную систему.
msgX   DB  'Vvedite X : ', 0
msgA   DB  'Vvedite A : ', 0
msgB   DB  'Vvedite B : ', 0
msgC   DB  'Vvedite C :' , 0
```

```
msgD   DB  'Vvedite D : ', 0
X      DW  ?
A      DW  ?
B      DW  ?
C      DW  ?
D      DW  ?
Y      DW  ?
DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ; Требуется для print_num.
DEFINE_PTHIS
DEFINE_CLEAR_SCREEN
END             ; Конец компиляции.
```

1. variant $X \geq 2$

```
Vvedite X : 3
Vvedite A : 2
Vvedite B : 2
Vvedite C : 3
Vvedite D : 4
Otvet: 4
```

2. variant $X < 2$

```
Vvedite X : 1
Vvedite A : 2
Vvedite B : 2
Vvedite C : 3
Vvedite D : 4
Otvet: 7
```

Индивидуальные задания

Таблица 2 – Задания для самостоятельного выполнения

№	Выражение	Данные				
		A	B	C	D	X
1.	$Y=\begin{cases} AX^2-B, \ если \ \ X\geq 4 \\ CX^2+D, \ если \ \ X<4 \end{cases}$	1	2	5	3	7 и 2
2.	$Y=\begin{cases} AX^2-B, \ если \ \ X\geq 6 \\ CX+D, \ если \ \ X<6 \end{cases}$	2	3	4	2	10 и 3
3.	$Y=\begin{cases} AX+B^2, \ если \ \ X\geq 6 \\ C^2X+D, \ если \ \ X<6 \end{cases}$	1	1	4	5	8 и 3
4.	$Y=\begin{cases} A^2X-B, \ если \ \ X\geq 8 \\ C^2X+D, \ если \ \ X<8 \end{cases}$	2	1	4	5	4 и 10
5.	$Y=\begin{cases} AX+7B, \ если \ \ X\geq 10 \\ CX-4D, \ если \ \ X<10 \end{cases}$	1	2	3	5	16 и 4
6.	$Y=\begin{cases} AX-B^2, \ если \ \ X\geq 8 \\ CX+12D, \ если \ \ X<8 \end{cases}$	1	2	3	1	9 и 3
7.	$Y=\begin{cases} 2AX-B^2, \ если \ \ X\geq 2 \\ CX+D, \ если \ \ X<2 \end{cases}$	6	4	3	1	4 и 1
8.	$Y=\begin{cases} AX-8B, \ если \ \ X\geq -6 \\ 7CX+4D, \ если \ \ X<-6 \end{cases}$	1	2	1	4	0 и -9
9.	$Y=\begin{cases} AX-7B, \ если \ \ X\geq 7 \\ CX+12D, \ если \ \ X<7 \end{cases}$	2	4	5	1	9 и 3
10.	$Y=\begin{cases} AX-B, \ если \ \ X\geq 5 \\ C^2X+5D, \ если \ \ X<5 \end{cases}$	8	1	2	1	7 и 2
11.	$Y=\begin{cases} A^2X^2-B, \ если \ \ X\geq 3 \\ CX+D, \ если \ \ X<3 \end{cases}$	2	1	6	5	6 и 2

Продолжение таблицы 2

№	Выражение	Данные				
		A	B	C	D	X
12.	$Y=\begin{cases}A+XB, & если \quad X\geq 13\\ C+XD, & если \quad X<13\end{cases}$	1	1	4	5	14 и 10
13.	$Y=\begin{cases}AX-B, & если \quad X\geq 14\\ CX+D, & если \quad X<14\end{cases}$	5	1	4	2	18 и 12
14.	$Y=\begin{cases}AX-B^2, & если -X\geq 20\\ CX+D^2, & если -X<20\end{cases}$	2	1	4	2	23 и 6
15.	$Y=\begin{cases}AX-B, & если \quad X\geq 24\\ CX-D^2, & если \quad X<24\end{cases}$	2	1	3	4	30 и 20
16.	$Y=\begin{cases}AX+B, & если \quad X\geq 10\\ 5CX-D, & если \quad X<10\end{cases}$	1	2	4	2	12 и 8
17.	$Y=\begin{cases}AX+2B, & если \quad X\geq 4\\ CX^2-D, & если \quad X<4\end{cases}$	3	2	4	1	6 и 3
18.	$Y=\begin{cases}A^2X-3B, & если \quad X\geq 5\\ CX-D^2, & если \quad X<5\end{cases}$	5	1	5	2	10 и 4
19.	$Y=\begin{cases}2AX+B, & если \quad X\geq 4\\ CX+2D, & если \quad X<4\end{cases}$	2	3	1	1	8 и 2
20.	$Y=\begin{cases}AX-3B, & если \quad X\geq 6\\ CX+D, & если \quad X<6\end{cases}$	4	1	3	4	10 и 4
21.	$Y=\begin{cases}3AX-B, & если \quad X\geq 12\\ 4CX-D, & если \quad X<12\end{cases}$	3	1	4	1	14 и 8
22.	$Y=\begin{cases}AX-4B, & если \quad X\geq 18\\ CX-2D, & если \quad X<18\end{cases}$	5	1	2	1	20 и 16
23.	$Y=\begin{cases}2AX-B^2, & если \quad X\geq 16\\ CX-3D, & если \quad X<16\end{cases}$	5	1	3	2	18 и 10
24.	$Y=\begin{cases}A^2X-B, & если \quad X\geq 8\\ 3CX+D, & если \quad X<8\end{cases}$	4	1	1	1	10 и 6

Окончание таблицы 2

№	Выражение	Данные				
		A	B	C	D	X
25.	$Y=\begin{cases} A^2X-B, \ если \ \ X\geq 16 \\ CX^2-4D, \ если \ \ X<16 \end{cases}$	5	1	4	2	18 и 12

Лабораторная работа № 4. Программирование циклических алгоритмов на языке Ассемблер

Цель работы: Освоить практические навыки программирования и выполнения циклических алгоритмов на языке Ассемблер.

Задания для самостоятельной работы

1. На основе приведенного примера составить программу для индивидуального задания.
2. Выполнить программу в среде EMU8086.
3. Составить аналогичную программу на языке C++ и сравнить результаты.

Порядок выполнения

Рассмотреть пример приведенный ниже, изучить структуру программы, применяемые команды и функции, и использовать их при решении индивидуального задания.

Пример. Составить программу на языке Assembler для вычисления суммы значений функции $y=f(x)$ на отрезке [a,b] с шагом h и выполнить в среде EMU8086.

$$S = \sum_{x=a}^{x=b} f(x),$$

где x=a,a+h, a+2h.....

Текст программы:

```
include 'emu8086.inc'
;s= SUM(y=2x+7) where x=(a to b) with step H
ORG    100h
GOTOXY 10,3
LEA    SI, msgA        ; запрос числа  A
CALL   print_string    ;
CALL   scan_num        ; запись A в CX.
```

```
MOV   A,CX
GOTOXY 10,4
LEA    SI, msgB       ; запрос числа  A
CALL   print_string   ;
CALL   scan_num       ; запись B в CX.
MOV   B,CX
GOTOXY 10,5
LEA    SI, msgh       ; запрос числа  H
CALL   print_string   ;
CALL   scan_num       ; запись H в CX.
MOV    H,CX
MOV bX,A
 MOV X,BX
vag:MOV AX,X
IMUL   DD
ADD ax,7
MOV bx,h
ADD x,bx
ADD s,ax
MOV bx,b
CMP x,bx
JLE vag
MOV ax,s
GOTOXY 10,8
CALL   pthis
DB  13, 10, 'Netice: ', 0
CALL   print_num      ; запись числа в AX.
RET                   ; Возврат в операционную систему.
msgA   DB  'Vvedite A : ', 0
msgB   DB  'Vvedite B: ', 0
msgH   DB  'Vvedite H :' , 0
X      DW  ?
A      DW  2
B      DW  6
H      DW  1
F      DW  ?
S      DW  0
DD     DW  2
```

```
DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ; необходимо для print_num.
DEFINE_PTHIS
DEFINE_CLEAR_SCREEN
END                   ; Конец компиляции.
```

Задания для индивидуального выполнения:

Таблица 3 – Задания для самостоятельного выполнения

№	Выражение	Данные		
		A	B	H
1.	Y=3X-8	1	6	1
2.	Y=7X+5	2	5	1
3.	Y=4X-6	1	6	2
4.	Y=5X+2	2	5	1
5.	Y=3X-6	1	5	1
6.	Y=4X+7	0	6	2
7.	Y=2X+3	1	6	2
8.	Y=3X+8	1	4	1
9.	Y=3X-2	2	4	1
10.	Y=5X+1	1	6	1
11.	Y=2X+3	2	8	2
12.	Y=4X+2	1	6	1
13.	Y=2X-5	1	6	1
14.	Y=4X+1	2	8	2
15.	Y=2X+3	2	10	4
16.	Y=2X+3	2	10	4
17.	Y=2X+3	2	10	4
18.	Y=2X+3	2	10	4

Окончание таблицы 3

№	Выражение	Данные		
		A	B	H
19.	Y=2X+3	2	10	4
20.	Y=2X+3	2	10	4
21.	Y=2X+3	2	10	4
22.	Y=2X+3	2	10	4
23.	Y=2X+3	2	10	4
24.	Y=2X+3	2	10	4
25.	Y=2X+3	2	10	4

Лабораторная работа № 5.
Обработка одномерных массивов на языке Ассемблер

Цель работы: разработать программу на языке Ассемблер, выполняющую обработку одномерного массива

Пример выполнения работы

Дан массив из десяти слов, содержащих целые числа. Требуется найти максимальное значение в массиве.

Текст программы:

```
LEABX, MASS
MOVCX, 10
MOV AX, [BX]
BEG: CMP [BX], AX
JL NO
MOV AX, [BX]
NO: INCBX
INCBX
LOOP BEG
MOV MAX, AX
HLT
MAX DW ?
MASS DW 10, 24,76,479, -347, 281, -24, 70, 124, 97
```

Варианты заданий

При сдаче задания, помимо исходного кода программы необходимо представить блок-схему алгоритма. Для составления блок-схемы рекомендуется использовать программу MSVisio.

Дан массив из десяти знаковых чисел (слов или байт). Требуется:

1. Найти количество отрицательных чисел. Массив байт.
2. Найти сумму всех положительных и отрицательных чисел. Массив слов

3. Найти сумму абсолютных величин. Массив байт.
4. Найти количество положительных чисел. Массив байт.
5. Поменять местами пары соседних чисел. Массив слов.
6. Переставить числа в обратном порядке. Массив байт.
7. Заменить все отрицательные числа нулями. Массив байт.
8. Найти среднее арифметическое чисел. Массив слов.
9. Найти количество чисел больших 10h. Массив слов.
10. Найти наименьшее по абсолютной величине числа. Массив байт.
11. Найти наибольшее отрицательное число. Массив байт.
12. Найти произведение положительных элементов последовательности. Массив слов.
13. Найти среднее арифметическое квадратов ненулевых элементов последовательности. Массив слов.
14. Найти полусумму наибольшего и наименьшего чисел. Массив байт.
15. Найти среднее арифметическое отрицательных элементов последовательности. Массив слов.
16. Найти сколько в массиве чисел больше 12h и меньше 0AFh. Массив байт.
17. Найти есть ли в массиве два нуля, идущих подряд. Массив слов.
18. Найти сумму абсолютных величин меньших 6. Массив байт.
19. Найти среднее арифметическое чисел больших 10. Массив слов.
20. Найти сколько чисел равно12h. Массив байт.
21. Заменить все отрицательные числа их модулями. Массив байт.
22. Найти среднее арифметическое положительных чисел Массив слов.
23. Найти количество чисел меньших 10h. Массив байт.
24. Найти наименьшее среди положительных чисел. Массив слов.
25. Найти наибольшее отрицательное число. Массив байт.

Лабораторная работа № 6.
Применение логических инструкций

Цель работы: разработать программу для применения логических инструкций.

Краткие теоретические сведения

Логические команды служат для сброса или установки отдельных бит в байте или слове. Они включают булевы операторы НЕ, И, ИЛИ, исключающее ИЛИ и операцию тестирования, которая устанавливает флаги, но не изменяет значения своих операндов.

Логические инструкции

not *dst*

Инструкция **not** инвертирует все биты байта или слова.

and *dst, src*

Инструкция **and** выполняет операции логическое И двух операндов (байтов или слов) и возвращает результат в операнд-приемник. Бит результата устанавливается в 1, если установлены в 1 оба соответствующих ему бита операндов, и устанавливаются в 0 в противном случае.

or *dst, src*

Инструкция **or** выполняет операции логическое ИЛИ двух операторов (байтов или слов) и помещает результат на место операнда-приемника. Бит результата устанавливается в 1, если равен 1 хотя бы один из двух соответствующих ему битов операндов и устанавливается в 0 в противном случае.

xor *dst, src*

Инструкция **xor** выполняет операцию логическое исключающее ИЛИ двух операндов и помещает результат на место операнда-приемника. Бит результата устанавливается в 1, если соответствующие ему биты операндов имеют противоположные значения, и устанавливается в 0 в противном случае.

test *dst, src*

Инструкция **test** выполняет логическое И двух операндов (байтов или слов), модифицирует флаги, но результат не возвращает, то есть операнды не изменяются.

В таблице 4 приведены значения регистра флагов, устанавливаемые логическими командами.

Таблица 4 – Значения регистра флагов

Логические инструкции								
Мнемокод		Флаги						Действие
Код	Операнды	O	S	Z	A	P	C	
and	dst, src	0	x	x	u	x	0	Логическое И
or	dst, src	0	x	x	u	x	0	Логическое ИЛИ
xor	dst, src	0	x	x	u	x	0	Логическое исключающее ИЛИ
not	dst	-	-	-	-	-	-	Логическое НЕТ
test	dst, src	0	x	x	u	x	0	Логическое И без изменения dst

Примечание:

- флаг не модифицируется;

x – устанавливается или сбрасывается в соответствии с результатом;

u – не определен;

0 – сбрасывается в 0.

Примеры использования логических команд

1. Установить 3 и 0 биты в регистре **al**, остальные биты не изменять.
   ```
   or al, 00001001b
   ```
2. Сбросить 4 и 6 биты в регистре **al**, остальные биты не изменять.
   ```
   and al, 10101111b
   ```
3. Инвертировать 2 и 4 биты в регистре **al**, остальные биты не изменять

```
xor al, 00010100b
```

4. Перейти на метку LAB, если установлен 4 бит регистра **al**, в противном случае продолжить выполнение программы.

```
test al, 00010000b
jnz LAB
продолжаем
…..
LAB:
```

5. Посчитать число единиц в регистре **al,** рассматривая байт, как набор бит.

```
MOV CX, B       ; число сдвигов
XOR BL, BL      ; обнуление BL
LL:  SH1 AL, 1      ; сдвиг влево на один разряд
JNC NO     ; переход, если нет переноса
INC BL              ;иначе увеличить BL
NO:  LOOP LL        ; возврат, если cx≠0
```

Пример выполнения работы

Дан массив из 10 байт. Все байты имеют нулевые старшие биты. Необходимо каждый байт, содержащий единицу в нулевом бите, дополнить до четного числа единиц установкой седьмого бита.

Текст программы:

```
MOV BX, 0
MOV CX, 10
BEG: MOV AL, NB[BX]
TEST AL, 1B
JZ BITOCLR
TEST AL, 0FFH
JP OK
OR AL, 80H
JMP SHORT OK
BITOCLR: TEST AL, 0FFH
JNP OK
OR AL, 80H
OK: MOV NB[BX], AL
```

```
;INT 10H
INC BX
LOOP BEG
HLT
NB DB 04H, 07H, 14H, 23H, 04H, 38H, 3FH, 2AH, 0DH, 34H
```

Варианты заданий

При сдаче задания, помимо исходного кода программы необходимо представить блок-схему алгоритма. Для составления блок-схемы рекомендуется использовать программу MSVisio.

1. Дан массив из 10 байт. Посчитать количество байт, в которых сброшены 6 и 4 биты.
2. Дан массив из 8 байт. Рассматривая его как массив из 64 бит посчитать количество единиц.
3. Дан массив из 8 байт. Рассматривая его как массив логических значений x0x1 x2 x3 x4 x5 x6 x7 (true – ненулевые биты, false – нулевые биты) вычислить логическую формулу:
   f = (x7 & x6 & x1) v (x6 & x4 & x2 & x1 & x0) v (x7 & x6 & x3 &&x1).
4. Дан массив из 10 байт. Посчитать количество байт с числом единиц в байте равным 3.
5. Рассматривая байт как набор логических значений x7 x6 x5 x4 x3 x2 x1 x0 (true – 1, false – 0) вычислить логическую формулу:
   f = (x7 & x6 & x3) v (x6 & x4 & x2 & x1) v (x7 & x6 & x2 &&x0).
6. Дан массив из 8 байт. Рассматривая его как массив из 64 бит посчитать длину самой длинной последовательности единиц.
7. Дан массив из 10 байт. Посчитать количество единиц во всех разрядах кратных трём, т.е. 3, 6, 9, … 75, 78.
8. Дан массив из 5 байт. Рассматривая его как массив из 8 пятиразрядных слов, найти «исключающее или» всех слов для выражения «10101».

9. Дан массив из 6 байт. Рассматривая его как массив из 48 бит, посчитать в нем количество нулей.
10. Дан массив из 8 байт. Рассматривая его как массив из 64 бит посчитать количество пар единиц в окружении нулей. Конец последовательности рассматривать как нуль.
11. Дан массив из 7 байт. Рассматривать его как массив из восьми семибитных слов, посчитать количество слов с нечетным числом нулей в слове.
12. Дан массив из 9 байт. Рассматривая его как массив из 72 бит посчитать число переходов между нулями и единицами.
13. Дан массив из 3 байт. Рассматривая его как массив из 24 бит посчитать количество одиночных единиц в окружении нулей. Конец последовательности рассматривать как нуль.
14. Дан массив из 6 байт. Посчитать количество байт, число единиц в которых не превышает 3.
15. Дан массив из 11 байт. Посчитать количество байт, в которых нет единиц стоящих рядом.
16. Дан массив из 4 байт. Рассматривая его как массив из 32 бит посчитать длину самой длинной последовательности нулей.
17. Дан массив из 6 байт. Посчитать количество единиц во всех разрядах кратных пяти: 5¸ 10, …, 45.
18. Дан массив из 3 байт. Рассматривая его как массив из 8 трехразрядных слов, найти «исключающее или» всех 8 слов для выражения «101».
19. Дан массив из 7 байт. Рассматривая его как массив из 56 бит, посчитать в нем количество нулей стоящих после единицы. Конец последовательности рассматривать как нуль.
20. Дан массив из 8 байт. Рассматривая его как массив из 64 бит, посчитать количество пар единиц в окружении нулей. Конец последовательности рассматривать как нуль.

21. Дан массив из 5 байт. Рассматривая его как массив из восьми пяти-битных слов, посчитать количество слов с четным числом единиц в слове.
22. Дан массив из 6 байт. Рассматривая его как массив из 48 бит, посчитать число двух единиц, стоящих между нулями. Конец и начало последовательности рассматривать как нули.
23. Дан массив из 3 байт. Рассматривая его как массив из 24 бит, посчитать количество одиночных единиц в окружении нулей. Конец последовательности рассматривать как нуль.
24. Дан массив из 6 байт. Посчитать количество байт, число единиц в которых не превышает 3.
25. Дан массив из 11 байт. Посчитать количество байт, в которых нет единиц стоящих рядом.

Список источников

1. Жмайлов Б. Б., Александров П. В. Лабораторный практикум по дисциплине "Архитектура ЭВМ и систем". – Ростов-на-Дону: Изд. ЮФУ, 2013. – 86 с.
2. Жмакин А. П. Архитектура ЭВМ. — СПб.: БХВ-Петербург, 2006. — 320 с.
3. Магда Ю. С. Ассемблер для процессоров Intel Pentium. – СПб: Питер, 2006. – 416 с.
4. Максимов Н. В., Партыка Т. Л., Попов И. И. Архитектура ЭВМ и вычислительных систем: учебник / – 5-е изд., перераб. и доп. – М.: ФОРУМ: ИНФРА-М, 2013. – 512 с.
5. Пильщиков В. Н. Программирование на языке Ассемблера IBM PC. – Диалог-МИФИ, 2005. – 288 с.
6. Таненбаум А. Э., Архитектура компьютера. 5-е изд. — СПб.: Питер, 2007. — 844 с.
7. Юров В. И. Assembler. Учебник для вузов. 2-е изд. — СПб.: Питер, 2003. —637 с.
8. Юров В. И., Assembler. Практикум. 2-е изд. – СПб.: Питер, 2006. – 399 с.

Приложение 1

Список команд процессора x8086

Таблица 5 – команды процессора x8086

Команда	Расшифровка	Действие
AAA	ASCII adjust AL after addition	Выравнивание символов после сложения
AAD	ASCII adjustAX before division	Выравнивание символовперед делением
AAM	ASCII adjustAXafter multiplication	Выравнивание символов послеперемножения
AAS	ASCII adjust AL aftersubtraction	Выравнивание символов послевычитания
ADC	Add with carry	Сложение с переносом единицы
ADD	Add	Сложение
AND	Logical AND	Логическая операция И
CALL	Call procedure	Вызов процедуры
CBW	Convert byte to word	Преобразование байта в слово
CLC	Clear carry flag	Очистка флага переноса
CLD	Clear direction flag	Очистка флаганаправления
CLI	Clear interrupt flag	Очистка флага прерывания
CMC	Complement carry flag	Дополнение флага переноса
CMP	Compare operands	Сравнение операндов

Продолжение таблицы 5

Команда	**Расшифровка**	**Действие**
CMPSB	Compare bytes in memory	Сравнение операндов в памяти
CMPSW	Compare words	Сравнение слов
CWD	Convert word to doubleword	Преобразование слова в двойное слово
DAA	Decimal adjust AL after addition	Выравнивание двоично-десятичных символов после сложения
DAS	Decimal adjust AL after subtraction	Выравнивание двоично-десятичных символов послевычитания
DEC	Decrement by 1	Уменьшение на 1
DIV	Unsigned divide	Деление чисел без знака
HLT	Enter halt state	Остановить процессор
IDIV	Signed divide	Деление чисел со знаком
IMUL	Signed multiply	Перемножение чисел со знаком
IN	Input from port	Ввод данных из порта
INC	Increment by 1	Увеличение на 1
INT	Call to interrupt	Вызов по прерыванию
INTO	Call to interrupt if overflow	Вызов по переполнению

Продолжение таблицы 5

Команда	**Расшифровка**	**Действие**
IRET	Return from interrupt	Возврат из прерывания
Jxx (JE, JG, JZ ит. д.)	Jump if condition	Условный переход (JE – переход, если «равно»,JG – переход если больше, JZ – переход, если «0», и т.д.)
JMP	Jump	Безусловный переход
LAHF	Load flags into AH register	Загрузить флаги в регистр AH
LDS	Load pointer using DS	Загрузить указатель, используя сегмент данных (DS)
LODSB	Load byte	Загрузить (в регистр) байт
LODSW	Load word	Загрузить слово
LOOP/LOOPx	Loop control	Управлениециклом «ДО/FOR» (LOOPE – повторить, если «равно», LOOPNE - повторить, если «неравно», LOOPZ – повторить, если «0», ипр.)
MOV	Move	Переслать
MOVSB	Move byte from string to string	Переслать байт из строки в строку
MOVSW	Move word from string to string	Переслать слово из строки в строку

Продолжение таблицы 5

Команда	**Расшифровка**	**Действие**
MUL	Unsigned multiply	Перемножение чисел без знака
NEG	Two’s complement negation	Дополнение к отрицанию
NOP	No operation	Нет операции
NOT	Negate the operand, logical NOT	Инвертировать операнд (логическое отрицание)
OR	Logical OR	Логическое ИЛИ
OUT	Output to port	Вывести в порт
POP	Pop data from stack	Принять («поднять») данные из стека
POPF	Pop datafrom flags register	Принять данные из регистра флагов
PUSH	Push data onto stack	Поместить («опустить») данные в стек
PUSHF	Push flags onto stack	Принять данные из регистра флагов
RCL	Rotate left (with carry)	Циклический сдвиг влево (с переносом)
RCR	Rotate right (with carry)	Циклический сдвиг вправо (с переносом)

Продолжение таблицы 5

Команда	**Расшифровка**	**Действие**
REPxx	Repeat	Управление циклом «ПОКА/ WHILE» (REPE – повторить, если «равно», REPNE – повторить, если «не равно», и т.д.)
RET/RETN.RETF	Return from procedure	Возврат из процедуры
ROL	Rotate left	Циклический сдвиг влево
ROR	Rotate right	Циклический сдвиг вправо
SAHF	Store AH into flags	Поместить регистр AH во флаги
SAL	Shift Arithmetically left (multiply)	Арифметический сдвиг влево (перемножение)
SAR	Shift Arithmetically right (signed divide)	Арифметический сдвиг вправо (деление со знаком)
SBB	Subtraction with borrow	Вычитание с займом единицы
SCASB	Compare byte string	Сравнить битовые строки
SCASW	Compare word string	Сравнить строки слов
SHL	Shift left (multiply)	Сдвиг влево (перемножение)
SHR	Shift right (unsigned divide)	Сдвиг вправо (беззнаковое деление)
STC	Set carry flag	Установить флаг переноса
STD	Set direction flag	Установить флаг направления

Окончание таблицы 5

Команда	**Расшифровка**	**Действие**
STI	Set interrupt flag	Установить флаг прерывания
STOSB	Store byte in string	Поместить байт в строку
STOSW	Store word in string	Поместить слово в строку
SUB	Subtraction	Вычитание
TEST	Logical compare (AND)	Логическое сравнение (И)
WAIT	Wait until not busy	Ожидать, пока линия BUSY# не занята (используется с блоком ПЗ)
XCHG	Exchange data	Обмен данными
XLAT	Table look-up translation	Использование таблицы транслитерации
XOR	Exclusive OR	Исключающее ИЛИ

Приложение 2

Формат команд передачи управления

Таблица 6 – Команды передачи управления

Команда	Формат ввода
Команды безусловной передачи управления:	
CALL	CALL имя
RET	RET [число удаляемых из стека значений]
JMP	JMP имя
Команды условной передачи управления:	
JA / JNBE	JA / JNBE близкая метка
JAE / JNB	JAE / JNB близкая метка
JNC	JNC близкая метка
JB / JNAE	JB / JNAE близкая метка
JC	JC близкая метка
JBE / JNA	JBE / JNA близкая метка
JCXZ	JCXZ близкая метка
JE / JZ	JE / JZ близкая метка
JG / JNLE	JG / JNLE близкая метка
JGE / JNL	JGE / JNL близкая метка
JL / JGNE	JL / JGNE близкая метка
JLE / JNG	JLE / JNG близкая метка
JNE / JNZ	JNE / JNZ близкая метка
JNO	JNO близкая метка
JNP / JPO	JNP / JPO близкая метка

Окончание таблицы 6

Команда	Формат ввода
JNS	JNS близкая метка
JO	JO близкая метка
JP / JPE	JP / JPE близкая метка
JS	JS близкая метка
Команды управления циклами LOOP	LOOP близкая метка
LOOPE / LOOPZ	LOOPE / LOOPZ близкая метка
LOOPNE / LOOPNZ	LOOPNE / LOOPNZ близкая метка

Приложение 3

Формат арифметических команд

Таблица 7 – арифметические команды

Команда	Формат ввода
Команды сложения	
ADD	ADD приемник, источник
ADC	ADC приемник, источник
AAA	AAA приемник, источник
DAA	DAA приемник, источник
INC	INC приемник
Команды вычитания	
SUB	SUB приемник, источник
SBB	SBB приемник, источник
AAS	AAS приемник, источник
DAS	DAS приемник, источник
DEC	DEC приемник
NEG	NEG приемник
CMP	CMP приемник, источник
Команды умножения	
MUL	MUL источник
IMUL	IMUL источник
AAM	AAM

Окончание таблицы 7

Команда	**Формат ввода**
Команды деления	
DIV	DIV источник
IDIV	IDIV источник
AAD	AAD
Команды расширения знака	
CBW	CBW
CWD	CWD