МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра электронных приборов

Методические указания по выполнению лабораторных работ по курсу «Основы микропроцессорной техники»

Версия 1.41

Для аппаратной версии 4.20

Новосибирск 2011

Составители: В. К. Макуха, А. Ф. Соловьёв.
Корректор: В. В. Миронов.

Содержание

Введение

На сегодняшний день широко распространены и популярны процессоры фирмы Intel. В настоящее время самыми развитыми моделями процессоров общего назначения являются Intel® Core™ (по состоянию на 2009 год). В качестве примера процессоров, выполненных по 45-нанометровой производственной технологии, можно привести [1]:

- **Intel® Core™2 i7 Extreme Edition** (i7-965), тактовая частота 3,20 ГГц, частота системной шины 1066 МГц, кэш-память второго уровня 12 Мбайт.
- **Intel® Core™2 Extreme QX9770**, тактовая частота 3,20 ГГц, частота системной шины 1600 МГц, кэш-память второго уровня 12 Мбайт.
- **Intel® Core™2 Quad Q9650**, тактовая частота 3,00 ГГц, частота системной шины 1333 МГц, кэш-память второго уровня 12 Мбайт.
- **Intel® Core™2 Duo E8600**, тактовая частота 3,33 ГГц, частота системной шины 1333 МГц, кэш-память второго уровня 6 Мбайт.

Хотя эти процессоры обладают самыми современными характеристиками и высочайшей производительностью, они не очень удобны для использования в обучающих стендах на начальных этапах изучения микропроцессорной техники из-за высокой сложности, стоимости и энергопотребления. Но при этом они относятся к архитектуре IA-32 и Intel® 64, которая является развитием архитектуры x86, а архитектура x86 является классической архитектурой микропроцессоров в персональных компьютерах класса IBM PC [2].

В качестве изучаемого микропроцессора была выбрана модификация Intel 80386EXTB. Хотя микропроцессор 80386 выпускается с 1985 года, базовые принципы его работы не сильно отличаются от тех принципов, что используются в современных микропроцессорах, при этом такой микропроцессор намного проще для изучения, чем более сложные версии той же архитектуры (80486, Pentium). Дальнейшие разработки были направлены в основном на улучшение быстродействия, производительности, увеличение размеров памяти и разрядности обрабатываемых данных. Все эти факторы, являющиеся решающими в выборе

пользователя и продвижении продукции на рынке, нисколько не помогают в изучении основ архитектуры и программирования, для освоения которых обычно и предназначены учебные стенды. Напротив, здесь требуются простота и наглядность системы, в доступной форме иллюстрирующие работу и устройство микропроцессора. Таким образом, Intel 80386 представляется наилучшим вариантом, сочетающим простоту в изучении и современный тип архитектуры. Данный микропроцессор является статическим, что позволяет выполнять программы пользователя по тактам, по циклам, по командам.

Микропроцессор i80836EX относится к группе так называемых встраиваемых микропроцессоров. Их отличительной особенностью является наличие в них модулей, характерных для микроконтроллеров: контроллера прямого доступа к памяти, блока синхронного и асинхронного обмена, таймеров/счётчиков, параллельных портов ввода-вывода, контроллера приоритетных прерываний [3]. Причём программные модели этих модулей совпадают с программными моделями микросхем (интерфейсных модулей) серии i82XX, используемых в современных персональных компьютерах класса IBM PC. Такой микропроцессор даёт возможность проводить лабораторные работы по изучению функционирования этих интерфейсных модулей.

Особенностью микропроцессора i80836EX является его ориентированность на особенности операционной системы MS-DOS. В частности, в пространстве ввода-вывода адреса, используемые стандартными устройствами компьютера IBM PC, отведены под встроенные интерфейсные модули.

1 Общие сведения о микропроцессоре Intel 80386

Микропроцессор i80386 — первая модель 32-разрядного микропроцессора фирмы Intel. 32-разрядными являются шины адреса и данных. Процессор совместим снизу вверх с предыдущими поколениями процессоров фирмы Intel [4]. Поясним, что это значит. Системы являются совместимыми, если программы, написанные на одной системе, успешно выполняются на другой. Совместимость снизу вверх означает, что программное обеспечение, написанное для более ранних процессоров (i8086, i80186, i80286), будет работать и на процессоре i80386, потому что набор команд МП i80386 и его модули обработки являются расширениями набора команд предшествующих моделей. А вот программное обеспечение, написанное специально для МП i80386 и использующее его специфические команды, не будет работать на процессорах более ранних версий [4].

Микропроцессор выполнен на основе технологии CHMOS III фирмы Intel, которая вобрала в себя быстродействие технологии HMOS (МДП высокой плотности) и малое потребление мощности технологии CMOS (КМДП) [4]. Изготовлен по технологии 1,5 мкм.

Микропроцессор может адресовать физическую память объёмом до 4 Гбайт. Конвейер команд, высокая разрядность шин, встроенное преобразование адреса значительно уменьшают среднее время выполнения команды. Эти архитектурные особенности позволяют микропроцессору i80386 выполнять 3–4 млн. команд в секунду [4].

1.1 Сегментация памяти и режимы работы процессора i80386

Изучаемый микропроцессор i80386EX принадлежит к семейству микропроцессоров i80x86 фирмы Intel, придерживающейся идеологии преемственности в архитектуре микропроцессоров. Соответственно, микропроцессор i80386EX содержит много черт, присущих первой модели этого

семейства — микропроцессору i8086. В свою очередь, 16-битному микропроцессору i8086 предшествовал 8-битный микропроцессор i8080 — первый микропроцессор, нашедший массовое применение в промышленности. От микропроцессора i8080 в i8086 перешли названия регистров общего назначения и возможность обращения к 16-битным регистрам общего назначения как к паре 8-битных регистров.

Одной из особенностей микропроцессора i8086 было то, что при 16 разрядной шине данных, микропроцессор имел 20 разрядную шину адреса. Чтобы не вводить регистры разрядностью 20 бит фирма Intel предложила так называемую сегментированную модель памяти. При этом сегментом называется область памяти размером 64 КБ (65536 байт). В соответствии с этим подходом, для формирования 20 разрядного адреса следует использовать пару 16-битных регистров. Один из этих регистров, называемый сегментным, указывает на расположение сегмента в адресном пространстве, а другой регистр, называемый регистром смещения, указывает на расположение конкретного байта в пределах одного сегмента (очевидно, что если сегмент имеем размер 64 КБ, то для формирования адреса конкретного байта требуется 16 бит). В 32-битном микропроцессоре i808386, в котором адресная шина также является 32-битной, явной необходимости в использовании сегментированной памяти нет, тем не менее, фирма Intel оставила эту возможность. На самом деле в «защищённом» режиме, который появился начиная с микропроцессора i80286, адреса формируются совершенно по другому принципу. Название «защищённый» режим проистекает от возможности запрещать доступ к определённым областям памяти (защищать определённые области памяти). Это, в частности, позволило использовать многозадачные операционные системы типа Windows. В данном цикле лабораторных работ мы не будем рассматривать эти вопросы.

Механизм формирования физического адреса, выдаваемого на адресную шину, показан на рисунке 1.1. Таким образом, адресация ячейки памяти производится по номеру сегмента и эффективному адресу ячейки в сегменте (называемому также смещением). Если результат сложения оказывается больше,

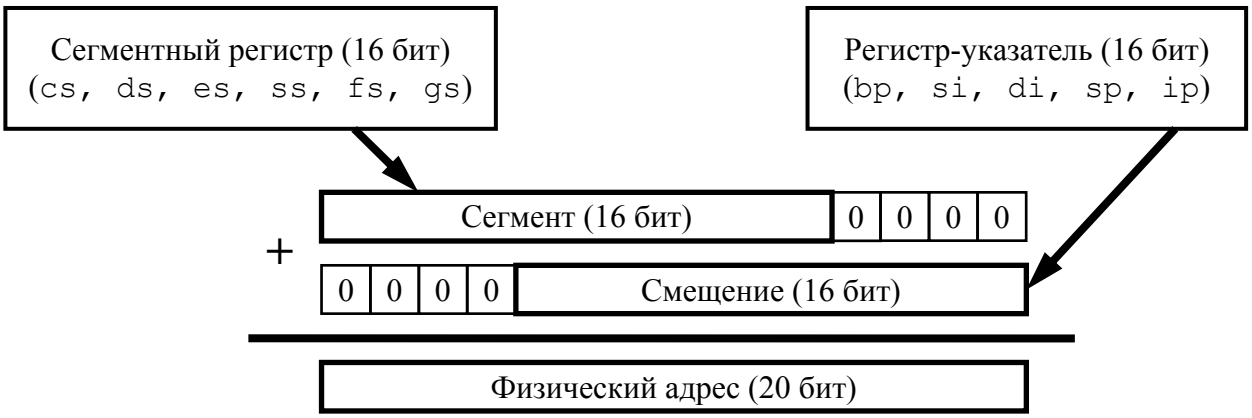


Рисунок 1.1 — Вычисление физического адреса в процессоре i8086

чем $2^{20} - 1$, то 21-й бит отбрасывается, эта процедура называется «заворачиванием» адреса (address wraparound) [5].

Ну а теперь настало время поговорить о режимах работы МП i80386. Микропроцессор i80386 может работать в трёх режимах: реальный режим, защищённый режим, виртуальный режим.

Реальный режим (эмуляция i8086) — это режим, имеющийся во всех процессорах Intel. Используется преимущественно с целью установки процессора для работы в защищенном режиме, а также с целью выполнения программ микропроцессоров предыдущих поколений [4]. При подаче сигнала сброса или при включении питания процессор всегда начинает работу в реальном режиме. В реальном режиме МП i80386 имеет такую же базовую архитектуру, что и МП i8086, но обеспечивает доступ к 32-разрядным регистрам. В реальном режиме МП 80386 для получения физического адреса использует тот же механизм что и процессор i8086. Этот механизм получения адреса в микропроцессоре i80386 был назван реальным режимом адресации. В реальном режиме размеры памяти и обработка прерываний МП i38086 полностью совпадают с аналогичными функциями МП i8086.

Единственным способом выхода из реального режима является явное переключение в защищённый режим, которое производится установкой специального флага в одном из системных регистров [4].

Защищённый режим позволяет полностью использовать все архитектурные возможности микропроцессора, но не позволяет выполнять программы, разработанные для i8086. С точки зрения программиста, защищённый режим, по сравнению с реальным, предоставляет большее адресное пространство и поддерживает новый механизм адресации. Это позволяет делать более гибкими используемые методы программирования и выполнять более крупные программы и программные пакеты.

Режим виртуального i8086. Переход в этот режим возможен, если процессор уже находится в защищённом режиме. В виртуальном режиме, в отличие от защищённого, возможна работа программ реального режима.

В этом лабораторном цикле мы не будем подробно рассматривать защищённый и виртуальный режимы.

Наши лабораторные работы мы будем проводить только в реальном режиме работы микропроцессора.

1.2 Регистры процессора i80386

Регистры МП i80386 являются расширением регистров прежних МП: 8086, 80186, 80286. Все 16-разрядные регистры МП предыдущих поколений содержатся внутри 32-разрядной архитектуры i80386. Регистры микропроцессора i80386 представлены на рисунке 1.2.

В нашем пособии мы используем классификацию, принятую для микропроцессоров i8086. В литературе, описывающей IA-32 Intel® Architecture, принята несколько другая классификация. Там к регистрам общего назначения относят индексные регистры и регистры смещения.

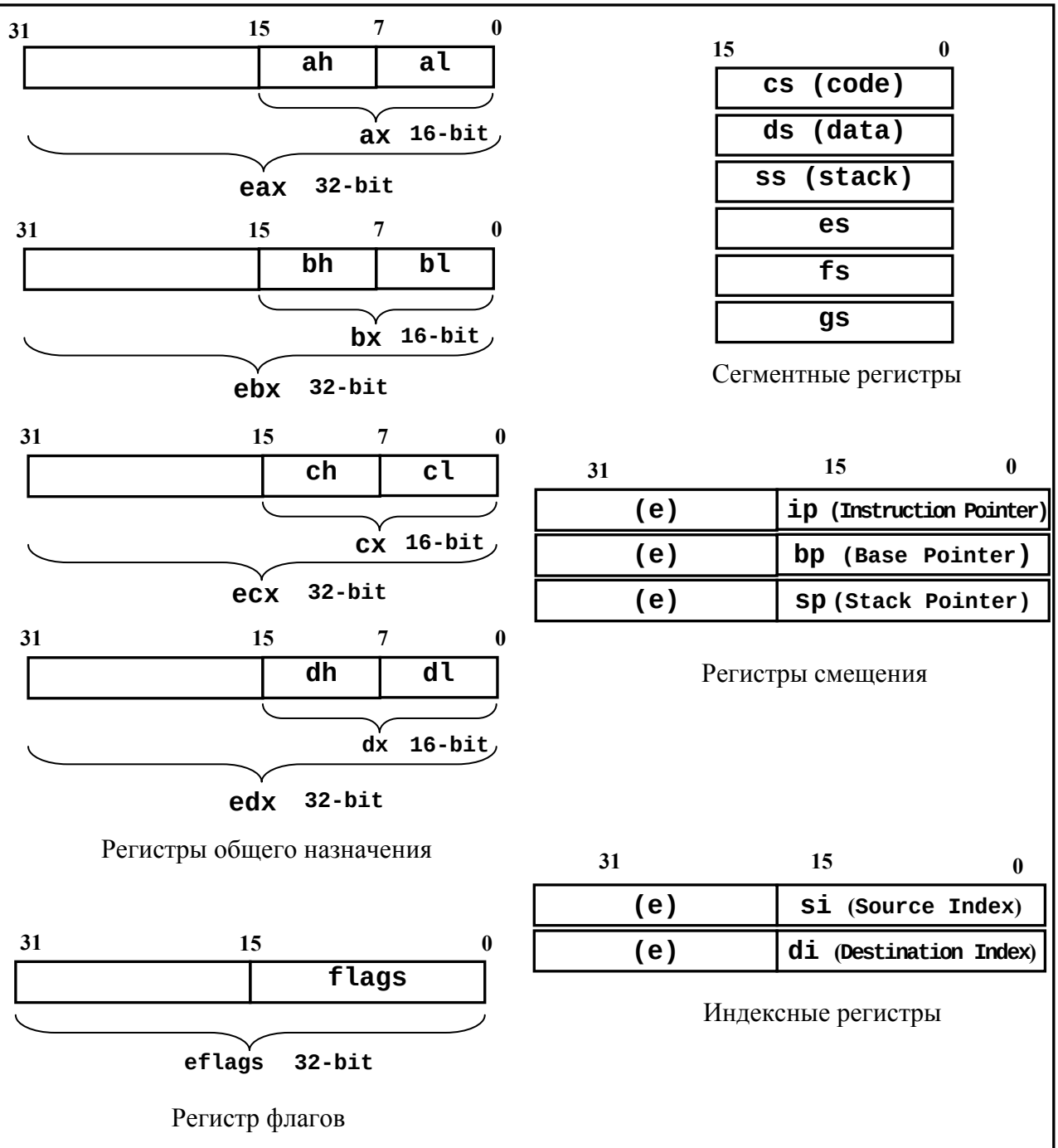


Рисунок 1.2 — Регистры

1.2.1 Регистры общего назначения

Регистры общего назначения используются для хранения операндов, компонентов адресов операндов и указателей на ячейки памяти.

К регистрам общего назначения относятся eax, ebx, ecx, edx. В принципе, все эти регистры доступны для хранения операндов, однако некоторые команды процессора требуют обязательного нахождения операндов в определённых регистрах.

Младшие 16 бит каждого регистра общего назначения доступны отдельно. Эти 16-разрядные регистры адресуемы как ax, bx, cx, dx. Четыре 16-разрядных регистра ax, bx, cx, dx могут быть дальше разделены на два 8-разрядных регистра. Это даёт возможность обращаться к 8-разрядным регистрам отдельно. Названия 8-разрядных регистров заканчиваются буквой h или l, где h (high) означает старшие разряды, а l (low) младшие. Регистры al, bl, cl, dl можно считать аналогами соответствующих регистров 8-битных микропроцессоров.

Каждый из регистров имеет предпочтительное применение.

Регистры общего назначения:

(e)ax (accumulator register) — аккумулятор, применяется для хранения операндов и результатов арифметических и логических операций [4].

(e)bx (base register) — базовый регистр, применяется для хранения базового адреса некоторого объекта в памяти [4].

(e)cx (count register) — регистр-счётчик, применяемый в циклических операциях.

(e)dx (data register) — регистр данных, так же, как и аккумулятор, используется для хранения данных [4].

1.2.2 Сегментные регистры, регистры смещения и индексные регистры

Сегментные регистры предназначены для хранения адресов сегментов кода, данных и стека, к которым программа имеет доступ. При обработке машинной команды микропроцессор использует адреса из определённых сегментных регистров.

Микропроцессор 80386 имеет шесть 16-разрядных сегментных регистров: cs, ds, ss, es, fs, gs, назначение которых приведено ниже.

`cs` (code segment register) — регистр сегмента кода. Хранит адрес сегмента, содержащего машинные коды выполняемой программы. Содержимое этого регистра не может быть изменено явным способом.

`ds` (data segment register) — регистр сегмента данных. Хранит адрес сегмента, содержащего обрабатываемые программой данные. Этот регистр может быть установлен программно, и более того — если программа должна обрабатывать какие-либо данные в памяти, это делать необходимо.

`ss` (stack segment register) — регистр сегмента стека. Хранит адрес текущего сегмента стека. Если в программе будет использоваться стек, необходимо загрузить в этот регистр адрес сегмента стека.

`es`, `fs`, `gs` — дополнительные сегменты данных, впервые появившиеся в 80286.

К регистрам смещения, также часто называемым указателями, относятся следующие регистры.

Указатель команд `eip` (extended instruction pointer) является 32-битным регистром. Он содержит относительный адрес следующей команды, подлежащей выполнению. Относительный адрес отсчитывается от начала (или базового адреса) сегмента текущей программы. Указатель команд непосредственно не доступен программисту, но он управляется неявно командами управления переходами, прерываниями и исключениями.

Младшие 16 бит регистра `eip` называются `ip` и могут быть использованы процессором независимо. Фактически, ip является полным аналогом программного счётчика (PC — Program Counter), используемого в 8-битных микропроцессорах. Использование `ip` полезно при работе с командами микропроцессора i8086, который имеет только регистр `ip`.

`(e)sp` (Stack Pointer register) — указатель стека. Содержит смещение вершины стека в текущем сегменте стека.

Содержимое `sp` используется инструкциями работы со стеком, инструкциями вызова и возврата из подпрограмм. Ответственность за установку стека лежит на

программисте! Поэтому программист, разрабатывающий программу на ассемблере, должен позаботиться об инициализации этого регистра (также, как и регистра `ss`).

`(e)bp` (Base Pointer register) — указатель базы. Иногда его называют указателем базы кадра стека. В частности, он может быть применён для организации доступа к данным внутри стека. Это, например, используется в подпрограммах при передаче аргументов через стек [4].

Индексные регистры, используемые в командах пересылки массивов данных (цепочечных, строковых командах):

`(e)si` (source index register) — индекс источника, содержит текущий адрес элемента в цепочке-источнике [4].

`(e)di` (destination index register) — индекс приёмника, содержит текущий адрес элемента в цепочке-приёмнике [4].

Отметим, что при обращении к этим регистрам в ходе выполнения строковых (цепочечных) команд реализуется адресация с автоувеличением или автоуменьшением.

1.2.3 Регистр флагов `eflags`

Микропроцессор 80386 содержит регистр `eflags` с 13 полями флагов. Регистр `EFLAGS` представлен на рисунке 1.3, затенением отмечены те флаги, которые будут использоваться при выполнении лабораторных работ. Некоторые биты регистра флажков зарезервированы фирмой Intel; биты 31 — 18, 15, 5, 3, 1 всегда содержат 0, а бит 1 всегда содержит 1 [4].

31	…	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	…	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

Рисунок 1.3 — Регистр EFLAGS

Флаги можно разделить на две группы, исходя из особенностей использования.

Флаги состояния. Эти флаги могут изменяться после выполнения машинных команд, что даёт возможность анализировать результаты их выполнения и изменять ход выполнения программы. В эту группу входят:

`CF` (carry flag) — *флаг переноса, бит 0*. Установлен, если в результате арифметической операции возник перенос из старшего бита при сложении или заём в старший бит при вычитании; сброшен, если переноса или заёма не было. Для 8-, 16-, 32-разрядных операций этот бит устанавливается при переносе из битов 7, 15, 31 соответственно.

`PF` (parity flag) — *флаг чётности, бит 2.* Установлен, если младшие 8 битов операнда содержат чётное число единиц, иначе сброшен. На этот флаг влияют только младшие 8 битов независимо от длины операнда.

`AF` (auxiliary carry flag) — *флаг вспомогательного переноса, бит 4.* Используется для упрощения сложения и вычитания двоично-десятичных чисел (BCD — binary-coded decimal). Независимо от длины операнда (8, 16 или 32 бит) флаг `AF` установлен, если операция привела к заёму в бит 3 при вычитании или переносу из бита 3 при сложении, иначе он сброшен.

`ZF` (zero flag) — *флаг нуля, бит 6.* Установлен, если все биты результата равны нулю, иначе сброшен.

`SF` (sign flag) — *флаг знака, бит 7.* Установлен, если установлен старший бит результата, иначе он сброшен. Для 8-, 16-, 32-разрядных операций этот флаг отражает состояние бита 7, 15 и 31 соответственно.

`OF` (overflow flag) — *флаг переполнения, бит 11.* Флаг установлен, если операция привела к переносу только в знаковый бит результата или только из знакового бита результата (при этом нужно анализировать только переносы, связанные со знаковым разрядом). Т. е. флаг устанавливается тогда, когда результат операции превысит диапазон чисел со знаком.

Системные флаги. Изменение этих флагов влияет на работу микропроцессора. В эту группу входят:

TF (trace flag) — флаг трассировки.

IF (interrupt flag) — разрешение аппаратных прерываний.

DF (direction) flag — флаг направления. Если флаг сброшен, то для регистров (e)si и (e)di в цепочечных операциях применяется адресация с автоувеличением (направление от младших адресов к старшим), а если установлен — с автоуменьшением.

VM (virtual microprocessor) — режим виртуального микропроцессора 8086.

Флаги IOPL, NT, RF используются микропроцессором только в защищённом режиме [6]. Мы будем работать в реальном режиме микропроцессора, поэтому эти флаги не рассматриваем и приводим только для справки.

1.3 Сигналы управления микропроцессора 80386

На рисунке 1.4 представлено условное графическое обозначение микропроцессора i80386EX.

Коротко рассмотрим назначение основных выводов шины управления микропроцессора 80386 [3]. Знак «#» перед названием сигнала означает, что активным уровнем для этого сигнала является уровень логического нуля (используется негативная логика).

W/#R — указывает цикл чтения или записи.

M/#IO — указывает, идет ли обращение к памяти или устройству ввода-вывода.

D/#C — определяет цикл данных или управления.

#ADS — подтверждает адрес, выставленный на шину адреса (является аналогом сигнала ALE микропроцессора i8086).

#LOCK — блокировка шины.

#READY — завершает текущий машинный цикл.

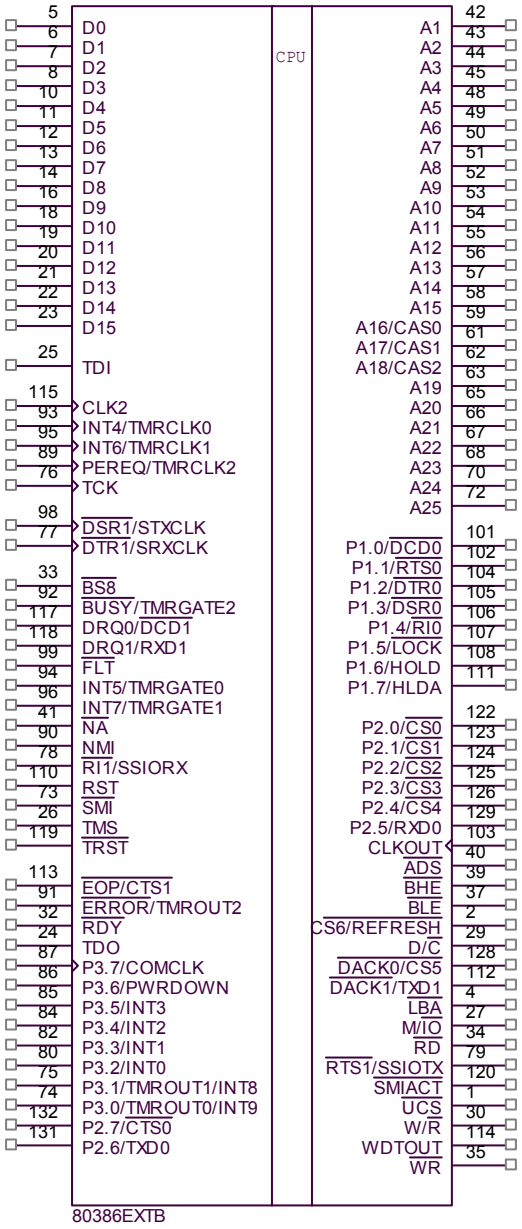


Рисунок 1.3 — Условное графическое обозначение микропроцессора i80386EX

#NA — вывод, позволяющий организовать конвейер адресов, то есть при подаче низкого уровня на этот вывод процессор генерирует сигналы адреса и состояния для следующего шинного цикла в текущем цикле, что уменьшает время доступа для памяти или устройств ввода-вывода.

#BS16 — говорит о том, что ширина шины данных составляет 16 бит.

INTR — маскируемое прерывание.

NMI — немаскируемое прерывание.

#BUSY, #ERROR, PEREQ — сигналы для взаимодействия с арифметическим сопроцессором. Сигнал #BUSY может быть использован и для других устройств.

HOLD — запрос на захват шины, позволяет другим устройствам запрашивать у микропроцессора разрешение на управление шинами (например, в режиме прямого доступа к памяти).

RESET — установка микропроцессора в начальное состояние.

CLK2 — сигнал двойной тактовой частоты.

2 Описание микропроцессорного блока стенда

2.1 Органы управления и система индикации

В данном цикле лабораторных работ на учебном стенде студент будет вводить свои коды команд с клавиатуры, при этом коды будут отображаться на жидкокристаллическом индикаторе. Запись кодов будет осуществляться в оперативную память учебного стенда (ОЗУ). После того как программа загружена, осуществляется переход к выполнению программы микропроцессором. Управление работой микропроцессора выполняется с помощью управляющих кнопок клавиатуры учебного лабораторного стенда. В ходе выполнения команд и программ микропроцессором, студент будет наблюдать состояние шин на светоизлучающих диодах (СИД) и содержимое регистров МП на жидкокристаллическом индикаторе.

Общий вид стенда представлен на рисунке 2.1.

Между жидкокристаллическим индикатором и клавиатурой расположен DIP переключатель выбора режима работы учебного стенда.

Для включения лабораторного стенда в режиме «Микропроцессор» необходимо выполнить следующую последовательность действий.

1 Перевести лабораторный стенд в режим работы с микропроцессором. Для этого нужно крайний правый движок 4-х разрядного DIP переключателя выбора режима работы учебного стенда установить в положение ON (в верхнее положение). При этом остальные движки должны находиться в нижнем положении, как это показано на рисунке 2.1.

2 Подключить блок питания к лабораторному стенду через разъём питания (см. рисунок 2.1).

3 Включить блок питания в сеть 220 вольт.

4 Должен загореться индикатор работы блока микропроцессора — светодиод белого цвета, расположенный рядом с надписью «Microprocessor Block»/

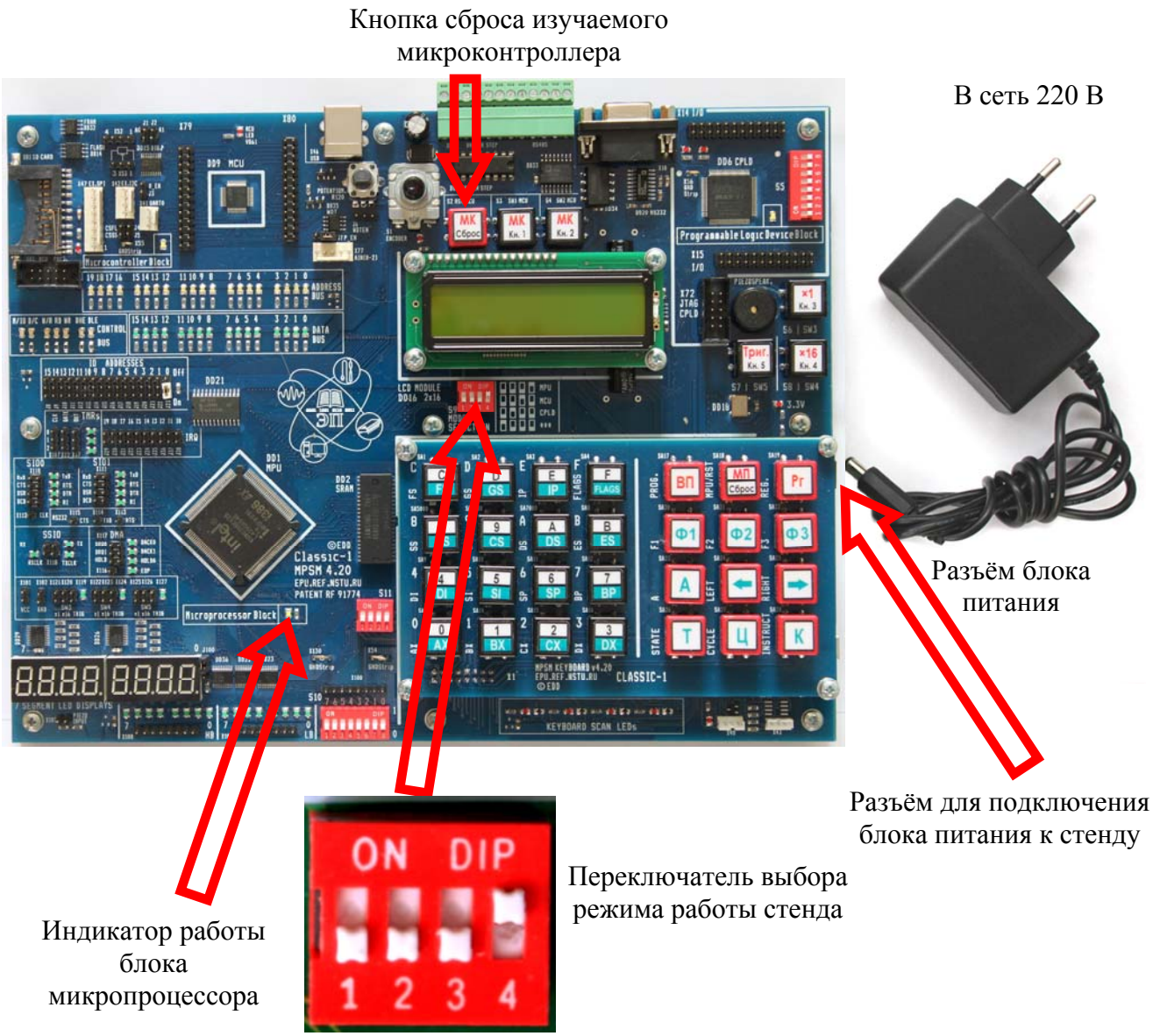


Рисунок 2.1 — Общий вид стенда

2.1.1 Клавиатура

Клавиатура учебного лабораторного стенда является очень важным узлом, который определяет, насколько быстро и правильно будут вводиться программы и производиться их отладка. Поэтому особое внимание следует уделить изучению назначения клавиш и функций, которые закреплены за этими клавишами.

При изучении микропроцессора (режим МП), сканирование клавиатуры ведёт системный МК, который, регистрируя нажатие кнопки, выдаёт на

пьезоэлектрический звуковой излучатель лабораторного стенда меандр различной частоты и продолжительности (в зависимости от функционального назначения кнопки и длительности её нажатия). Поэтому нажатие кнопок с разным функциональным назначением сопровождается характерным звуковым сигналом. При нажатии кнопок следует различать так называемые «короткое» и «продолжительное» нажатия. Продолжительным является нажатие длительностью более чем примерно 0.5 с, до появления характерного звукового сигнала.

Нажатие кнопки, не имеющей какой-либо функции, не сопровождаются звуковым сигналом (нажатие не приводит ни к каким дополнительным действиям).

Клавиатура учебного лабораторного стенда представлена на рисунке 2.2.

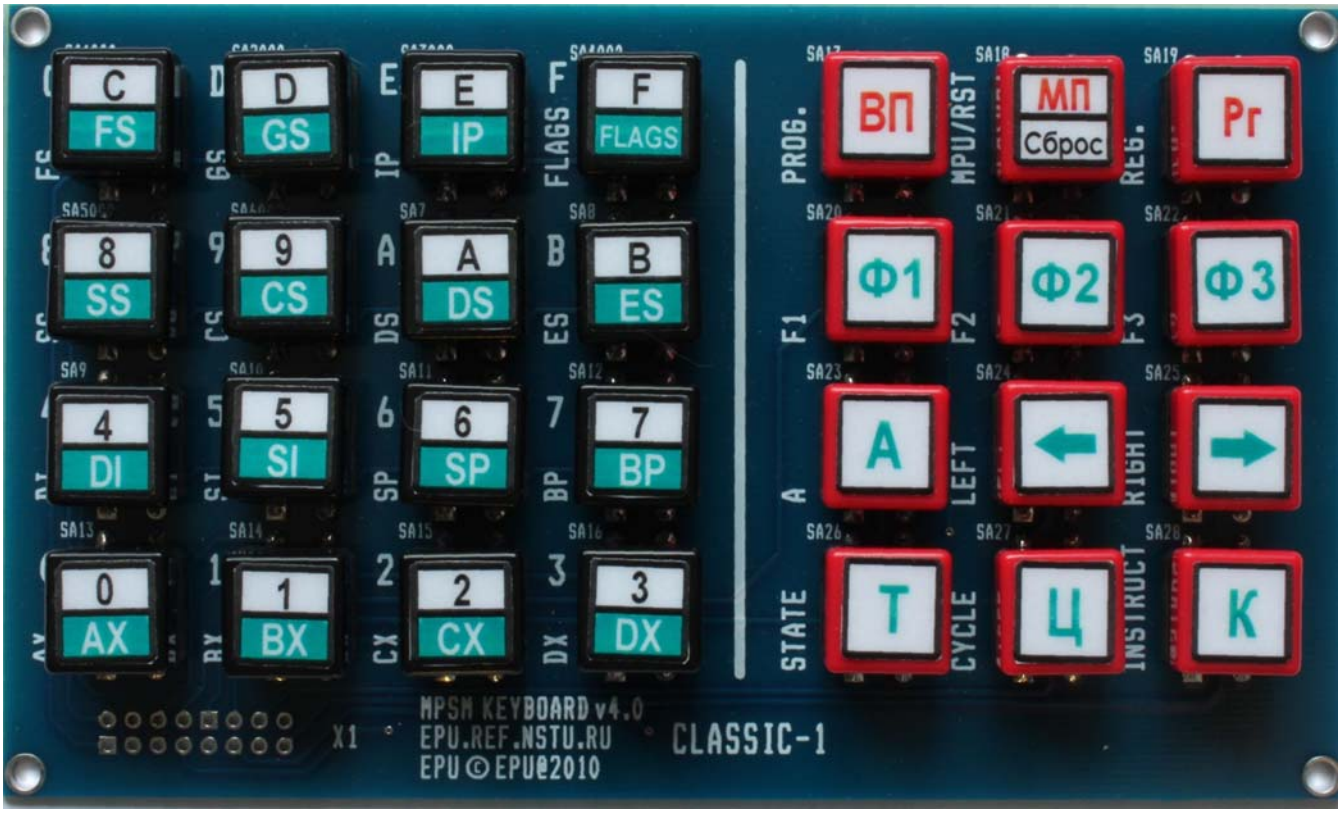


Рисунок 2.2 — Внешний вид клавиатуры учебного лабораторного стенда

Клавиатура состоит из блока цифровых клавишей (цифровая клавиатура — 16 клавишей, расположенных слева) и блока управляющих клавиш (управляющая клавиатура — 12 клавиш, расположенных справа). Обратите внимание, что надписи

на кнопках продублированы на шелкографии (слева от каждой кнопки). Это позволяет эксплуатировать клавиатуру без наклеек и без колпачков.

Рассмотрим назначение клавиш в различных режимах работы.

Управляющая клавиатура. Описание клавишей клавиатуры.

«ВП» — переход в режим ввода программы для МП (Ввод Программы в ОЗУ).

«МП/Сброс» — выполняет две функции: переход в режим выполнения программы микропроцессором и сброс микропроцессора. Сброс МП выполняется при продолжительном нажатии клавиши «МП/Сброс».

«Рг» — обеспечивает переход в режим просмотра состояния регистров микропроцессора.

«Ф1», «Ф2» и «Ф3» — три функциональные клавиши. Клавиша «Ф1» предназначена для включения/выключения звука, сопровождающего нажатие клавиш. Продолжительное нажатие «Ф1» отключает звук при нажатии клавиш, следующее продолжительное нажатие кнопки «Ф1» — включает.

Клавиша «Ф2» служит для восстановления содержимого системной области памяти. Дело в том, что при автоматическом тактировании микропроцессора (и неверной программе пользователя), процессор может выйти за пределы программы пользователя и выполнить случайный код, который может затереть системные программы, находящиеся в ОЗУ (см. карту памяти: обработчики TF, INT3, NMI, программа BIOS). Сброс процессора не приведет к восстановлению работы стенда. В этом случае правильная работа стенда в режиме выполнения программы микропроцессором не гарантируется, особенно при выполнении программы по командам, при переходе в другие режимы выполнения программы, при сбросе микропроцессора (так как в этих случая МП использует приведенные выше обработчики и BIOS). Может наблюдаться неадекватная реакция стенда при нажатии клавиш «А», «Т», «Ц», «К», «сброс». Для восстановления работоспособности стенда в режиме выполнения программы МП необходимо восстановить системные программы в ОЗУ (выполнить регенерацию ОЗУ). Для того чтобы выполнить регенерацию ОЗУ не осуществляя сброс всего стенда, и не выключая питания, необходимо произвести продолжительное нажатие кнопки

«Ф2». Пользователю также требуется проверить свою программу, которая может оказаться поврежденной.

Клавиша «Ф3» зарезервированы для будущего применения.

«А» — клавиша имеет двойную функцию. При кратком нажатии происходит запуск программы на автоматическое выполнение («А» — автомат). В этом случае микропроцессор постоянно тактируется внешним генератором. Автоматическое выполнение программы прекращается, если в программе встречается код `0CCh (int 3)`.

При продолжительном нажатии клавиши «А» происходит переход курсора в позицию адреса программы («А» — адрес). Адрес программы расположен в левой части верхней строчки. В этом состоянии можно задать стартовый адрес программы. Последующее продолжительное нажатие клавиши «А» возвращает курсор в позицию адреса данных в нижней строке ЖКИ, и предоставляет возможность выполнения программы в каком-либо режиме.

«Т», «Ц», «К» — клавиши выбирают режим выполнения программы.

При нажатии клавиши «Т» (Такт) микропроцессор выполняет один такт. При нажатии клавиши «Ц» (Цикл) микропроцессор выполняет машинный цикл. При нажатии клавиши «К» (Команда) происходит выполнение микропроцессором одной команды.

В режиме ввода программы в ОЗУ, переход в который производится нажатием клавиши ВП, используются следующие функции управляющих клавиш.

«А» — перейти к вводу адреса программы или данных в памяти. Продолжительное нажатие клавиши «А» переводит курсор в верхнюю левую часть ЖКИ, где располагается адрес ввода программ. Здесь с помощью цифровых клавиш можно изменить адрес, с которого начнётся выполнение программы в режиме МП. Следующее продолжительное нажатие клавиши «А» перемещает курсор на нижнюю строку, где можно вводить, просматривать и редактировать данные.

Короткое нажатие клавиши «А» переводит курсор в правую часть верхней строки. Здесь можно задать адрес данных, которые будут отображаться в режиме выполнения программ в нижней строке ЖКИ.

«←»/«→» — эти клавиши функционируют только в режиме ввода данных, и позволяют получить доступ к предыдущей/следующей тетраде данных (адреса). Продолжительное нажатие клавиши со стрелкой вызывает перемещение на 5 байт в адресном пространстве в соответствующем направлении.

Цифровая клавиатура. С помощью цифровой клавиатуры можно вводить шестнадцатеричные цифры (двоичные тетрады), и выбирать регистры микропроцессора (`ax`, `bx`, `cx`, `dx`, `di`, `si`, `sp`, `bp`, `ss`, `cs`, `ds`, `es`, `fs`, `gs`, `ip`, `flags`) для отображения их содержимого на экране ЖКИ.

В режиме ввода программы в ОЗУ, который выбирается нажатием клавиши «ВП», цифровая клавиатура используется для ввода соответствующих шестнадцатеричных цифр. При нажатии клавиши вводится цифра, указанная на кнопке сверху.

В режиме просмотра регистров МП, вызываемого нажатием управляющей клавиши Рг, цифровая клавиатура используется для выбора регистра, состояние которого необходимо отобразить на ЖКИ. При нажатии кнопки выбирается регистр, имя которого указано на клавише снизу.

В этом режиме продолжительное нажатие клавиши, соответствующей выбранному регистру `ax, bx, c, dx, di, si, sp bp`, переводит стенд в режим редактирования содержимого регистра. Нажимая на цифровые клавиши можно изменить содержимое указанных регистров. Выход из режима редактирования содержимого регистров происходит при повторном продолжительном нажатии клавиши, соответствующей выбранному регистру.

2.1.2 Блок светодиодной индикации

Блок светодиодной индикации служит для отображения состояния системных шин лабораторного стенда. Системные шины — это шины, связывающие между

собой микропроцессор, ОЗУ, устройства ввода-вывода. Системными шинами являются шина адреса, шина данных и шина управления. В первом ряду сверху располагается линейка светодиодов шины адреса. На них отображается состояние девятнадцати разрядов шины адреса. Процессор не имеет вывода A0, поэтому в

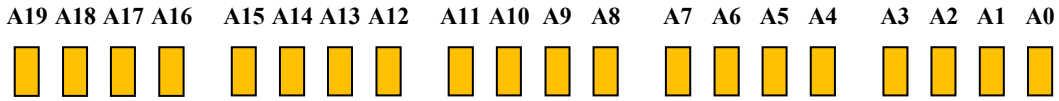


Рисунок 2.3 — Линейка светодиодов шины адреса

качестве A0 отображается инверсия сигнала `#BLE` (чтении/запись младшего байта ОЗУ). Линейка светодиодов шины адреса представлена на рисунке 2.3.

Во втором ряду располагаются светодиоды, отображающие состояние шины данных. Линейка светодиодов шины данных представлена на рисунке 2.4.

Применительно к шине адреса и данных состояние светодиода означает следующее. Если светодиод светится, то на шине выставлен сигнал высокого уровня, если не светится — низкого.

Правее индикаторов шины данных располагаются светодиоды, отображающие

Рисунок 2.4 — Линейка светодиодов шины данных

Рисунок 2.5 — Линейка светодиодов шины управления

состояние сигналов управления МП. Последовательность сигналов управления МП, представлена на рисунке 2.5. Шина управления работает в инверсной (негативной) логике. Поэтому если на линии выставлен сигнал низкого уровня, то светодиод светится, а если присутствует высокий уровень напряжения, то светодидо не

светится. Т. е. светящийся светодиод отображает активность сигнала управления низкого уровня.

В шину управления входят следующие сигналы:

M/#IO — показывает, что процессор в данный момент обращается к памяти (Memory — высокий уровень) или к внешним устройствам (IO — низкий уровень);

D/#C — показывает, что процессор работает с данными (Data — высокий уровень) или с командами (Control — низкий уровень);

W/#R — показывает, является ли текущий шинный цикл циклом записи (Write — высокий уровень) или чтения (Read — низкий уровень);

#RD — показывает, что процессор осуществляет чтение;

#WR — показывает, что процессор осуществляет запись;

#BHE — показывает, что осуществляется передача старшего байта данных (Byte High Enable);

#BLE — показывает, что осуществляется передача младшего байта данных (Byte Low Enable).

Традиционно на шине управления действует негативная логика, поэтому свечение СИД говорит об активном уровне сигнала. Например, свечение светодиода M/IO# говорит о том, что активна функция ввода-вывода IO.

2.1.3 Жидкокристаллический индикатор

В режиме ввода программы и в режиме редактирования на ЖКИ отображаются вводимые пользователем адреса и данные. В режиме выполнения на ЖКИ отображается содержимое регистра IP и расположенные по соответствующему адресу машинные коды, а также область данных, адрес которой был выбран заранее. В режиме просмотра регистров МП ЖКИ отображает текущее состояние регистров.

2.2 Начальная инициализация микропроцессора и карта памяти

При включении лабораторного стенда, сразу же после сброса процессора автоматически выполняется его инициализация. Программы, выполняющие инициализацию, обычно называются программами BIOS (Basic Input-Output System). Программа BIOS обнуляет регистры МП, устанавливает стек, конфигурирует внутренние ресурсы МП, сохраняет состояние регистров МП в ОЗУ и передаёт управление программе пользователя по адресу 4000h. Карта памяти представлена на рисунке 2.6.

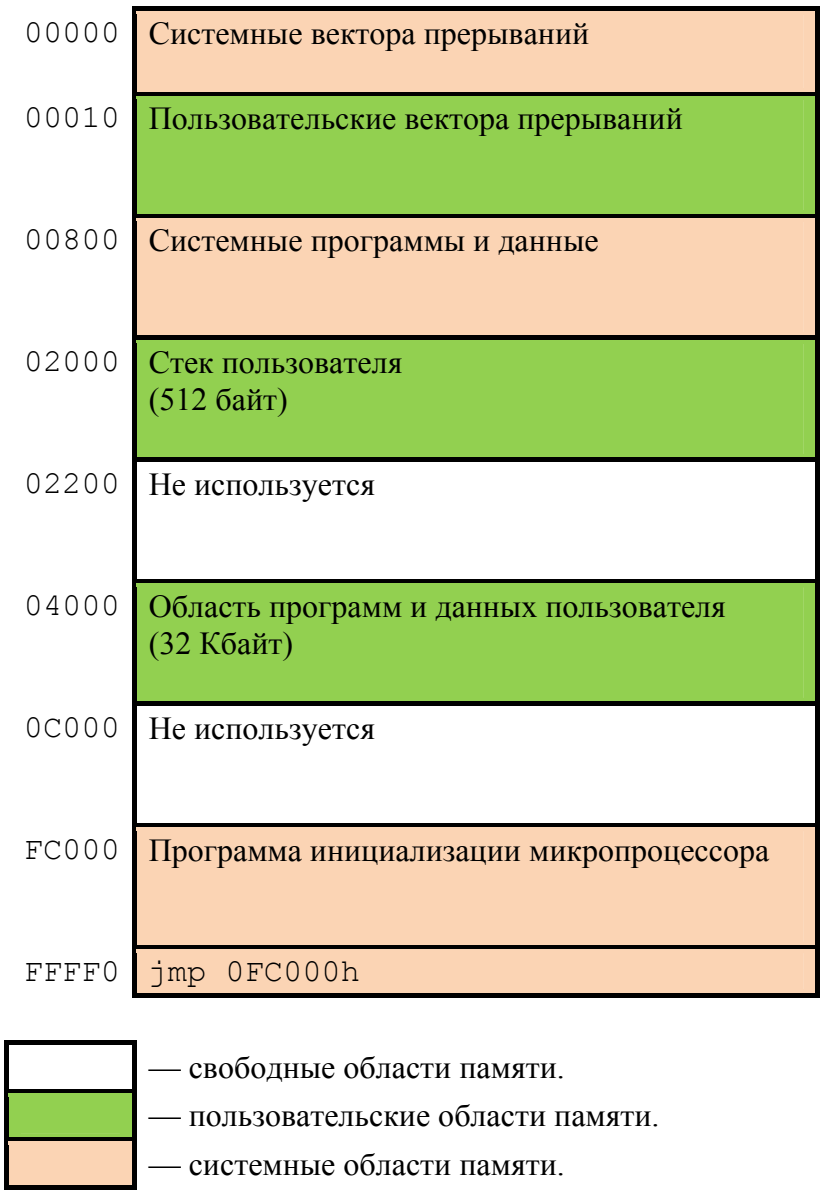


Рисунок 2.6 — Карта памяти стенда

Состояние регистров микропроцессора сохраняется в области системных данных. Поэтому после инициализации МП пользователь может посмотреть начальное состояние регистров.

После инициализации стенд переходит в режим ввода программы в ОЗУ, в котором ожидает дальнейших действий пользователя.

Для пользовательских программ и данных отведена область ОЗУ, начиная с адреса 4000h и заканчивая адресом BFFFh (32 Кбайт).

Пользователю не нужно устанавливать стек, т. к. он автоматически установлен программой BIOS на адрес 2200h. Объём установленного стека — 512 байт. При необходимости пользователь может изменить указатель стека, но при условии, что он будет находиться в пользовательской области памяти[1].

Системные области памяти, пользователю модифицировать нельзя, т. к. там находятся программы и данные, обеспечивающие работоспособность стенда.

2.3 Порядок работы со стендом

При работе с микропроцессорным блоком лабораторный стенд может функционировать в трёх основных режимах:

- режим ввода в ОЗУ программы пользователя;
- режим выполнения микропроцессором программы пользователя;
- режим просмотра состояния регистров микропроцессора.

2.3.1 Ввод программы

В режиме ввода программы, пользователь может вводить программы и данные в память (ОЗУ), а также просматривать содержимое памяти на экране ЖКИ. Переход в режим ввода программы осуществляется по нажатию на управляющей клавиатуре клавши «ВП» (Ввод Программы). Поскольку ввод данных будет

[1] Следует учесть, что для правильной работы стенда размер стека должен быть установлен на 6 байт больше необходимого пользователю.

производиться потетрадно, полезно вспомнить, как адресуются байты и двухбайтные слова в памяти.

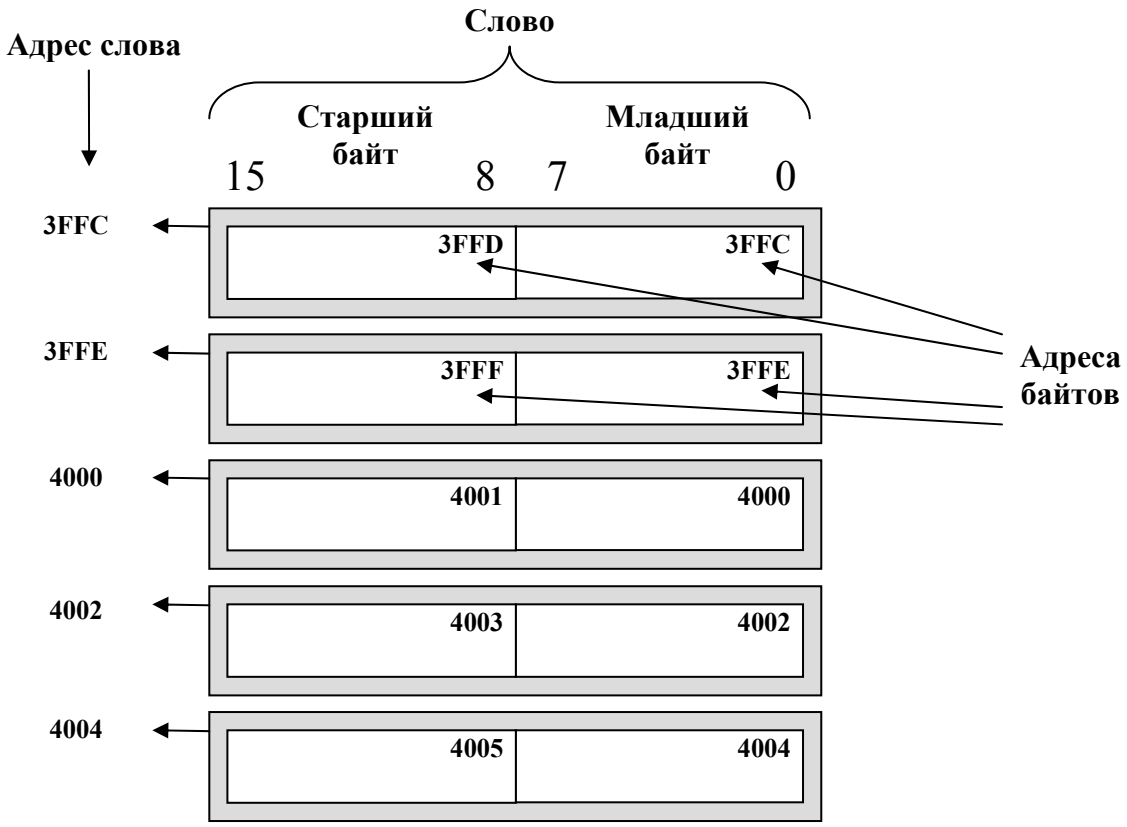


Рисунок 2.7 — Адреса ячеек памяти ОЗУ

Минимально адресуемым элементом памяти практически для всех микропроцессоров является байт. Микросхема памяти на стенде имеет разрядность 16 бит, т. е. 2 байта, или одно слово. МП при работе с памятью может обращаться как к байтам, так и к словам, при этом адрес байта может быть любым, а адрес слова — только чётным. Когда данные имеют чётные адреса, говорят, что они выровнены по границе слова. На рисунке 2.7 представлены адреса байтов и слов в памяти. Как видно из рисунка, все слова имеют чётный адрес (разряд A0 адресной шины всегда равен нулю).

Способы отображения вводимой информации на дисплее. Существует несколько способов отображения вводимой информации на дисплее: слева направо без сдвига, слева направо со сдвигом, справа налево со сдвигом [7]. При вводе слева

направо без сдвига каждый вводимый символ помещается правее предыдущего. При вводе слева направо со сдвигом каждый вводимый символ помещается слева от предыдущего, причём предыдущий символ сдвигается на одну позицию вправо. При вводе справа налево со сдвигом каждый вводимый символ помещается справа от предыдущего, причём предыдущий символ сдвигается на одну позицию влево (такой ввод применяется обычно в калькуляторах).

Для удобства ввода и контроля вводимой информации в учебном лабораторном стенде ввод на ЖКИ осуществляется слева направо без сдвига, когда

ЖКИ

Рисунок 2.8 — Вид ЖКИ при вводе данных

каждый вводимый байт (в шестнадцатеричном виде) помещается справа от предыдущего.

Ввод программы. При переходе в режим ввода программы пользователя на ЖКИ отображается сообщение «A=4000», показывая начальный адрес области памяти программ пользователя по умолчанию. Начиная с этой ячейки памяти, мы будем загружать код нашей программы. Вид ЖКИ в этот момент показан на рисунке 2.8. В нижней строчке при включении питания находятся случайные числа.

Если необходимо сменить адрес области памяти, куда будут вводиться данные, нужно произвести продолжительное нажатие клавиши «A» и курсор переместится в верхнюю строчку. Теперь можно ввести новое значение адреса в шестнадцатеричной системе счисления с помощью цифровой клавиатуры. Теперь после продолжительного нажатия клавиши «A» курсор перейдёт в нижнюю строку и можно продолжить загрузку кодов программы.

Данные также вводятся в шестнадцатеричных кодах с помощью цифровой клавиатуры. Ввод данных осуществляется по порядку, тетрада за тетрадой, байт за байтом. Для данного примера первый введённый байт (кода нашей программы или фрагмента данных) будет иметь в памяти адрес 4000h. Следующий введённый байт будет иметь в памяти адрес 4001h и так далее.

В процессе заполнения нижней строки ЖКИ в верхней строке будет отображаться адрес вводимого байта. Если какой-либо символ был введён неправильно, то можно воспользоваться клавишами со стрелками «→» и «←», чтобы переместить курсор в нужную позицию и ввести правильные данные. При продолжительном нажатии клавиш «→» и «←» происходит автоматическое перемещение в адресном пространстве на 5 байт.

Коды можно брать из справочной таблицы, приведённой в приложении А, из листинга, создание которого описано в приложении Б, или получать с помощью программы ApBuilder. Например, введём по адресу 4000h следующий фрагмент кода:

```
BB 00 06       mov bx, 0600h       ; команда имеет код BB 00 06
C7 07 40 00    mov ds:[bx], 040h   ; команда имеет код C7 07 40 00
```

После ввода фрагмента кода, ЖКИ будет выглядеть так, как показано на рисунке 2.9. Запись в память вводимых данных происходит автоматически после каждого введённого символа, никаких дополнительных клавиш, таких как «Запись», нажимать не нужно. Ввод и запись в память каждого байта сопровождаются специальным звуковым сигналом.

ЖКИ

Рисунок 2.9 — ЖКИ после ввода адреса и данных

Когда курсор дойдёт до крайней правой позиции, дальнейший ввод данных будет сдвигать текущую вводимую строку влево.

Для того чтобы посмотреть введенную информацию, начиная с младшего адреса (с дальнейшим инкрементом адреса), следует выставить начальный адрес и использовать продолжительное нажатие кнопки «→». Чтобы просмотреть данные со старшего адреса (с дальнейшим декрементом адреса), следует выставить начальный адрес и последовательно продолжительно нажимать клавишу «←».

ЖКИ

Рисунок 2.10 — Состояние ЖКИ при переходе в режим выполнения программы

2.3.2 Выполнение программы

Переход в режим выполнения программы осуществляется по нажатию кнопки «МП». При этом содержимое дисплея ЖКИ примет вид, показанный на рисунке 2.10. Для изменения адреса выполняемой программы требуется продолжительное нажатие клавиши «А» (при продолжительном нажатии клавиши «А» букву «А» следует трактовать не как «Автомат», а как «Адрес»). Возврат в режим выполнения программы после изменения стартового адреса программы производится продолжительным нажатием клавиши «А».

Пользователь может выполнять свою программу по тактам, по циклам, по командам и автоматически. Для этого необходимо нажимать соответствующую клавишу: «Т», «Ц», «К», «А».

Например, для того чтобы выполнить первые две команды программы в покомандном режиме, а выполнение третьей команды наблюдать в потактовом, необходимо выполнить следующие действия:

1) нажать клавишу «МП» (переход в режим выполнения программы);

2) нажать клавишу «К» (выполнить первую команду);

3) нажать клавишу «К» (выполнить вторую команду);

4) нажать клавишу «Т» (выполнить первый такт третьей команды);

5) нажать клавишу «Т» (выполнить второй такт третьей команды), и т. д.

Непосредственное наблюдение за ходом выполнения программы в потактовом, поцикловом, покомандном режимах осуществляется по блоку светодиодной индикации. Блок светодиодной индикации позволяет наблюдать стадию выборки команд и данных из памяти, а также в некоторых случаях и стадию выполнения (например: `out dx ax; push ax; mov ds:[4100], bx`). Потактовый режим даёт подробную картину выполнения машинного цикла: выставляемые адреса, данные, управляющие сигналы. После выполнения машинного цикла в поцикловом режиме, блок светодиодной индикации отображает начало следующего машинного цикла (начало стадии выборки). После выполнения команды в покомандном режиме, блок светодиодной индикации отображает начало стадии выборки следующей команды. При выполнении команд в потактовым и поцикловом режиме нужно знать, что микропроцессор i80386 имеет буфер очереди команд на 16 байт, заполнением которого управляет устройство опережающей выборки.

В режиме выполнения программы по командам или автоматически на ЖКИ отображается дополнительная информация о содержимом памяти по адресу Ad, указанному в слева в нижнее строке. Пример дополнительно отображаемой на ЖКИ информации представлен на рисунке 2.10.

В верхней строке отображается текущее состояние регистра микропроцессора `IP` (на рисунке это 4000h) и соответствующие данные, на которые указывает регистр `IP` (`C70600BB`). Причём следует учесть, что отображение данных в этом режиме отличается от отображения данных в режиме ввода программы и

соответствует расположению байт в памяти. Справа располагается байт, на который указывает IP, этот байт является младшим байтом среди всех байт, отображаемых на дисплее. Т. е. в данном случае старший байт находится левее младшего. Например, если по адресу 4000h находится команда `mov bx, 0600h`, то справа

ЖКИ

Рисунок 2.11 — Состояние ЖКИ при выполнении программы

налево первым байтом идет код этой команды (BBh), вторым младший байт данных (00h), третьим старший байт данных (06h). Байт C7h — это начало кода следующей команды (кода команды `mov ds:[bx], 040h`).

После нажатия кнопки «К» будет выполнена команда `mov bx, 0600h`, и IP укажет на следующую команду, т. е. на ячейку памяти с адресом 4003h. ЖКИ будет отображать информацию так, как представлено на рисунке 2.11.

ЖКИ

Рисунок 2.12 — Отображение адреса данных

Последовательность байт C7 07 40 00 — это код команды `mov ds:[bx], 040h`.

Нижняя строка ЖКИ позволяет следить за какой-то определённой областью памяти **(Ad — Address data)**, например, за данными, которые загружает наша программа в память, или за содержимым стека. Как и в первой строке, старший байт данных находится левее младшего. На рисунке 2.12 адрес, с которого начинаем просматривать данные, равен 5000h. По адресу 5000h находится байт BDh, по адресу 5001h — D5, по адресу 5002h — 08h и так далее.

Изменение адреса отображаемой области данных (окна в памяти) достигается заданием адреса отображаемой области данных в режиме ввода программы.

ЖКИ

Рисунок 2.13 — Отображение адреса данных в режиме ввода программы

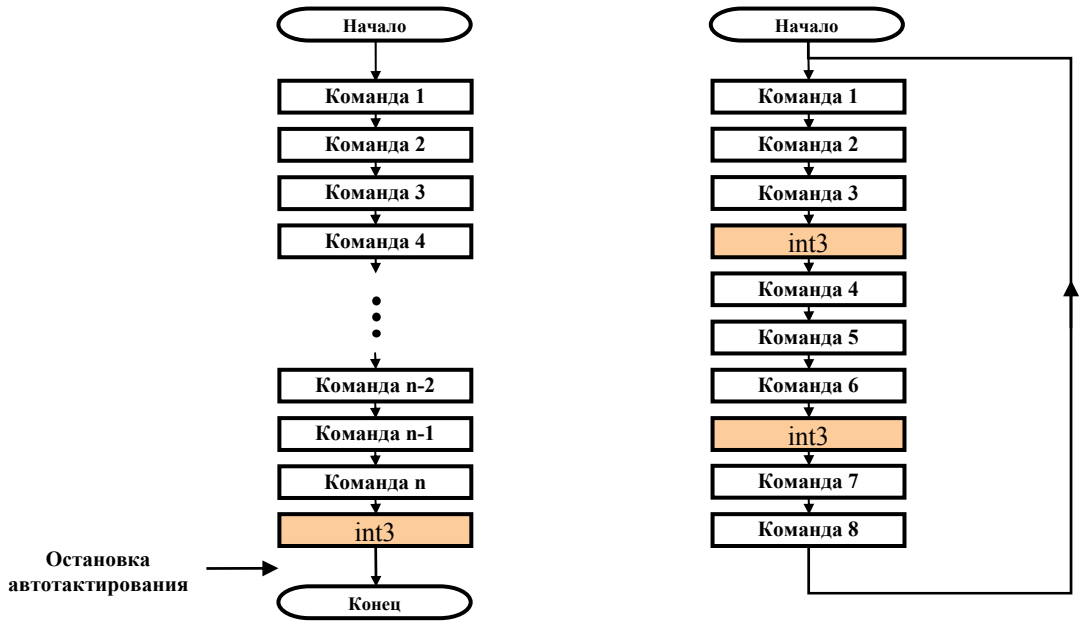
Для этого сначала нужно перевести курсор в верхнюю строку, используя короткое нажатие кнопки «А». Затем необходимо ввести соответствующий адрес, и снова коротко нажать кнопку «А». После характерного звукового сигнала введённый адрес будет зафиксирован как адрес отображаемой на ЖКИ области данных, при этом он будет указан в правом верхнем углу ЖКИ, как показано на рисунке 2.13.

Контроль состояния регистров МП доступен только при выполнении программы пользователя по командам и в автоматическом режиме. При выполнении программы по тактам и по циклам контроль состояния регистров недоступен.

В автоматическом режиме тактирование микропроцессора осуществляется от генератора частотой 25 МГц.

Чтобы остановить автоматическое тактирование микропроцессора по завершению программы пользователя, необходимо в конце программы ставить команду `int 3` (`int 3` имеет код CCh). Команда `INT 3` для микропроцессора является командой, которая выполняет функцию точки останова. По каждой точке останова можно контролировать состояние регистров микропроцессора. Пример использования команды `int 3` представлен на рисунке 2.14.

Если по окончании программы не предусмотреть команду `int 3`, то процессор продолжит выполнение кодов за пределами программы пользователя, что приведет к непредсказуемым результатам.



Линейный алгоритм Циклический алгоритм

Рисунок 2.14 — Примеры использования команды `int 3`

При повторном продолжительном нажатии кнопки «МП» в режиме выполнения программы, выполняется сброс микропроцессора, причём это событие сопровождается характерным звуковым сигналом.

2.3.3 Просмотр и редактирования содержимого регистров микропроцессора

Для того чтобы перейти в режим просмотра состояния регистров МП, необходимо нажать кнопку «Рг». При этом на ЖКИ отобразится содержимое четырёх регистров. Для того чтобы просмотреть состояние определённого регистра микропроцессора, на цифровой клавиатуре требуется нажать кнопку с названием нужного регистра. Пользователь имеет возможность просматривать состояние шестнадцати 16-разрядных регистров микропроцессора: `ax`, `bx`, `cx`, `dx`, `di`, `si`, `sp`, `bp`, `ss`, `cs`, `ds`, `es`, `fs`, `gs`, `ip`, `flags`. Одновременно на ЖКИ можно просматривать состояние четырёх регистров: двух в верхней и двух в нижней строке ЖКИ (см. рисунок 2.15).

ЖКИ

Рисунок 2.15 — Одновременное отображение состояния нескольких регистров МП

Имеется возможность побитного просмотра состояния флагового регистра. Для этого необходимо после выбора флагового регистра продолжительно нажать кнопку «flags». При этом на ЖКИ отобразится состояние 9 флагов. Позиция, занимаемая ими на ЖКИ, соответствует разряду, занимаемому в регистре `flags`. При этом также отображаются и зарезервированные биты, которые имеют всегда определённые значения. На ЖКИ они помечены звёздочкой. Последовательность флагов, отображаемых на ЖКИ, представлена на рисунке 2.16.

Чтобы вернуться в режим ввода или выполнения программы, необходимо нажать соответственно кнопку «ВП» или «МП».

FLAGS = 0002

11	10	9	8	7	6	5	4	3	2	1	0
O	**D**	**I**	**T**	**S**	**Z**	*	**A**	*	**P**	*	**C**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

Шестнадцатеричный формат *Двоичный формат*

Рисунок 2.16 — Состояние флагового регистра МП после сброса

Пользователь может изменить содержимое восьми регистров — `ax`, `bx`, `cx`, `dx`, `di`, `si`, `sp`, `bp`. Для перехода в режим редактирования содержимого регистров следует выбрать регистр с помощью кнопки с названием регистра или кнопок «→» и «←» и произвести продолжительное нажатие кнопки с названием выбранного регистра. Переход в режим редактирования сопровождается продолжительным

звуковым сигналом, и старший разряд выбранного регистра отмечается символом подчёркивания. В режиме редактирования цифровая клавиатура выполняет функцию ввода цифр, кнопки со стрелками функцию позиционирования в тетрадах редактируемого регистра.

Возврат из режима редактирования осуществляется продолжительным нажатием кнопки с названием выбранного регистра.

3 Порядок выполнения лабораторных работ

3.1 Домашняя подготовка к работе и оформление отчёта

При домашней подготовке к лабораторной работе следует повторить необходимый материал, пользуясь конспектом лекций и рекомендованной литературой. В ходе домашней подготовки к каждой работе необходимо придерживаться следующего порядка:

1 Ознакомьтесь с контрольными вопросами и ответьте на них.

2 Изучите задание по работе и продумайте пути выполнения всех пунктов. Составьте алгоритмы выполнения и соответствующие им программы на ассемблере. Составьте программы в машинных кодах.

Пример алгоритма изображен на рисунке 3.1. Количество повторений цикла определяется начальным значением в регистре `cx`. Следует учесть, что для организации циклов в системе команд процессора i80386 есть специальные команды, например `LOOP`.

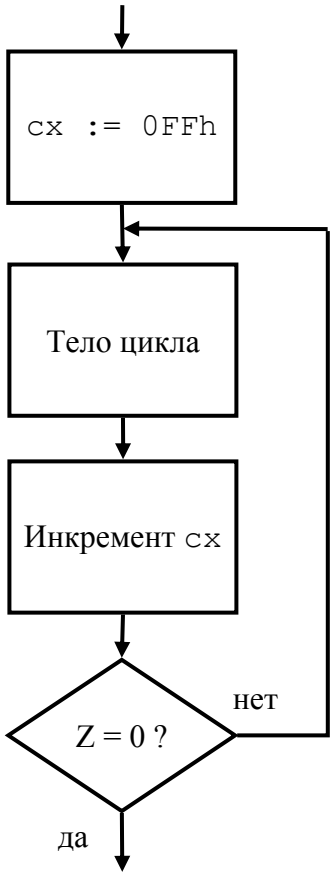


Рисунок 3.1 — Пример алгоритма цикла.

3 Сделайте заготовку отчёта — титульный лист с наименованием работы, датой выполнения, фамилией и группой студента. В отчёт занесите цель работы и основные данные, разработанные алгоритмы (в виде блок-схем) и программы с их двоичными кодами, подготовленными для отладки. Программы на ассемблере и их двоичные коды занесите в отчёт в виде таблицы. Пример оформления такой таблицы представлен на рисунке 3.2. Таблица обязательно должна содержать следующие графы: адрес памяти (причём адрес памяти

не из листинга, а в соответствии с картой памяти лабораторного стенда), код команды, мнемоника и комментарий. В каждую отдельную строчку таблицы должны заноситься данные только об одной команде, как это представлено на рисунке 3.2.

№ команды	Адрес	Код (hex)	Мнемоника	Комментарий
1	4000h	90	nop	;Задержка
2	4001h	B8 00 00	mov ax, 0000h	;Поместить в ax ноль
3	4004h	8B D8	mov bx, ax	;Поместить в bx ноль
4	4006h	89 1E 08 4A	mov ds:[4A08h], bx	;Поместить в ячейку ;памяти ноль

Рисунок 3.2 — Пример оформления программы в отчёте

Перед началом каждой лабораторной работы студент предъявляет преподавателю отчёт с подготовленным заданием. После предварительного опроса к выполнению работы допускаются студенты, подготовившие отчёт и чётко представляющие цель и методику проведения предстоящей работы.

При выполнении лабораторных работ каждый студент должен строго соблюдать правила техники безопасности, а также указания преподавателя.

3.2 Получение значений кодов команд для микропроцессора

Для получения двоичных кодов своих программ следует пользоваться таблицей, приведённой в приложении А, или программой ApBuilder v2.21 от фирмы Intel. Описание работы с программой ApBuilder v2.21 дано в приложении Б. В этой программе, кроме кодов, приведены форматы команд и их описание.

Самый быстрый путь — брать двоичные коды команд и проводить их анализ из листинга, генерируемого транслятором. Описание работы с транслятором TASM5 дано в приложении В. При разработке своих программ на языке ассемблера следует придерживаться определенного оформления текста программ. Пример оформления программы на ассемблере приведён ниже.

```
;###############################################################
;Программа DipToLed осуществляет чтение Dip переключателя
;и вывод его состояния на светодиоды.
;Примечание: Для Dip переключателя и регистра светодиодов
;назначен один и тот же адрес (010Eh)
;    Входные параметры отсутствуют
;    Выходные параметры отсутствуют
;Имя модуля:   DipToLed.asm
;Язык:         ассемблер, TASM5 от фирмы Borland
;Объект:       Intel 80386EXTB
;Автор:        Соловьёв А.Ф., Группа:РЭМ-31
;Дата:         20.10.2008
;Версия:       1.0.0
;###############################################################

CODE SEGMENT
     assume    cs:code
     assume    ds:code
     assume    es:code
     org  4000h                ;Working address area in stand memory
%NOLIST
     INCLUDE   80386EX.INC     ;File is located in
                               ;C:\Intel\APBLDR\386\CODE folder
                               ;Lines 6 – 15 must be commented
%LIST
START:
     mov  al, 01               ;Immediate addressing
LabLabel:
     mov  dx, 00B0h            ;Set port address
     out  dx, al               ;Output for address in dx to LED
     in   al,dx                ;Get DIP status into al
     nop                       ;Some delay
     nop
     nop
     nop
     nop
     nop
     jmp LabLabel                   ;Make loop
CODE ENDS
     END START
```

Если этот текст поместить в текстовый файл с расширением .asm и выполнить ассемблирование и линковку (см. Приложение В), то вы получите листинг и исполняемый файл:

```
B0 01 BA B0 00 EE EC 90 90 90 90 90 90 EB F3.
```

Все соответствующие программы расположены на диске.

Отметим, что эта программа не может выполняться в автоматическом режиме, поскольку в ней нет кода `0CCh`.

4 Комплекс лабораторных работ

4.1 Лабораторная работа № 1. Знакомство со стендом. Изучение простых команд и методов адресации

Цель работы: изучение различных методов адресации, знакомство с работой стенда и исследование простых команд.

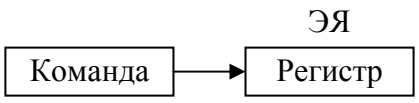
4.1.1 Теоретические сведения

Методы адресации. Ниже перечислены основные методы адресации, использующиеся в 80386 [4; 6; 9; 10]. Дадим определение понятию эффективная ячейка (ЭЯ). *Эффективная ячейка* — это ячейка (ячейка памяти или регистр), содержащая операнд-источник или операнд-приёмник. Для изучения методов адресации удобно использовать команды, принадлежащие к группе команд пересылки. В командах этой группы должны использоваться два метода адресации: один метод для операнда-источника и один метод — для операнда приёмника.

Регистровая адресация (register addressing) — эффективной ячейкой является регистр.

Например, команда (здесь сначала идут шестнадцатиричные коды команды, а затем мнемоническое обозначение команды):

```
8B D8          mov bx, ax
```

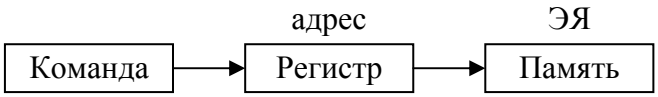


означает, что микропроцессор берёт 16-разрядное значение операнда-источника (содержимое `ax`) и перемещает его в 16-разрядный регистр `bx`. В этой команде и для источника, и для приёмника используется регистровая адресация.

Косвенная регистровая адресация (*indirect addressing)* — эффективной ячейкой является ячейка памяти, адрес которой расположен в регистре адресации.

К команде с косвенно-регистровым методом адресации относится команда

```
8A 07          mov al, [bx]
```



Здесь для операнда-источника используется косвенно-регистровая адресация, а для операнда приёмника применяется регистровая адресация. Отметим, что в соответствии с обозначением оператора-приёмника команда оперирует с байтом.

Для пересылки в память с использование косвенно-регистровой адресации в качестве регистров адресации могут быть использованы базовый регистр смещения `bp` и индексные регистры `si` и `di`.

Команды строковой пересылки `movs` могут служить примером косвенной регистровой адресации с автоувеличением/автоуменьшением, полезной для пересылки массивов данных:

```
A5             movsw
```

Здесь операндом-приёмником является ячейка памяти, адрес которой равен содержимому `di`, а операндом-источником — ячейка памяти, адрес которой равен содержимому регистра `si`. После выполнения команды содержимое `di` и `si` автоматически изменяется (будет ли оно увеличиваться или уменьшаться, зависит, как уже говорилось выше, от состояния флага направления d), а на сколько — зависит от формата операнда. В данном случае изменение должно быть равно двум (формат операнда — слово, то есть два байта).

Отметим, что для команд ввода-вывода при косвенной адресации используется только регистр `dx`.

Непосредственная адресация (*immediate addressing)* — метод адресации, при котором значение операнда находится в самой команде. Данные размещается непосредственно в поле операнда источника. Отметим, что непосредственная адресация не может быть использована для операнда-приёмника. Пример команд с непосредственной адресацией:

```
B4 00          mov ah, 00h
B8 05 00       mov ax, 05h
```

Часто удобно операнд задавать не в виде числа, а присваивать ему символическое имя. Тогда для получения кодов в TASM 5.0 можно использовать такой вид записи:

```
     MyData   equ 0005h
B8 05 00          mov ax, MyData
```

Здесь регистр `ah` обнуляется, а в регистр `al` загружается двоичный код 00000101. На рисунке 4.1 приведено расположение этих команд в памяти.

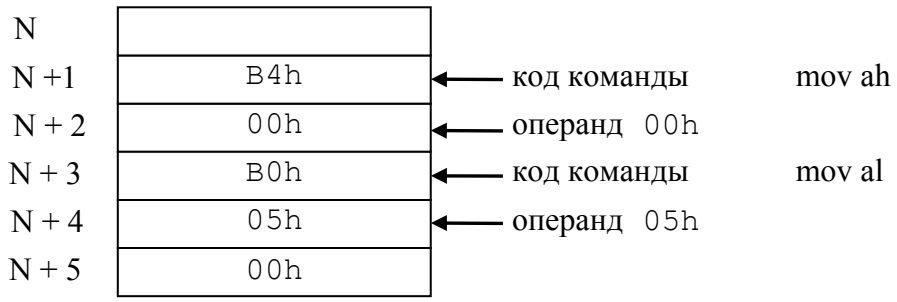


Рисунок 4.1 — Пример расположения команд в памяти при непосредственной адресации

Прямая адресация (*direct addressing)* — метод адресации, при котором адрес ячейки памяти указывается в команде. При прямой адресации адрес задаётся в виде смещения в пределах сегмента. Это смещение содержится в команде в виде 16(8)-разрядного параметра. Смещение прибавляется к сдвинутому значению регистра сегмента данных `ds` (или другого сегментного регистра, указанного в команде), что дает 20-разрядный физический адрес, по которому и хранится искомый операнд. Адрес в команде может быть представлен непосредственно числом или меткой. Например:

```
A1 00 0F       mov ax,word ptr ds:[0F000h]
```

или

```
     ORG            0F000h
     MyDataAddress  dw   0
     ORG       4000h
A1 00 F0       mov ax,MyDataAddress
```

или

```
A1 00 F0        mov ax,word ptr ds:[MyDataAddress]
```

По этой команде микропроцессор загружает регистр `ax` значением ячейки памяти 0F000h, адрес которой хранит метка MyDataAddress. Следует заметить, что микропроцессор сначала загружает младший байт адреса, затем старший (формат little endian, Intel-формат — младший байт располагается по младшему адресу). На рисунке 4.2 представлено расположение трехбайтовой команды в памяти.

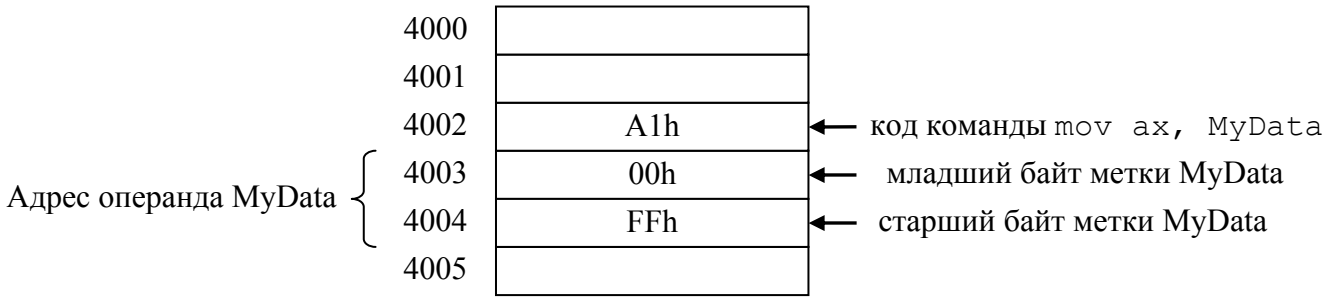


Рисунок 4.2 — Расположение команды `mov ax, MyData` в памяти.

Также следует отметить, что программа ассемблер, создавая листинг программы, отразит не реальное расположение байт в памяти, а представит адрес в виде, удобном для программиста. Например, для анализируемой команды в листинге в области кодов мы увидим: `A1 0F 00`.

Относительная адресация (базовая адресация) (base-displacement addressing) — эффективный адрес операнда вычисляется путём суммирования значения базовых регистров `bx` или `bp` и смещения. Например:

```
8B 47 04        mov ax,[bx+4]
```

Команда загружает в `ax` значение, содержащееся по адресу, равному сумме содержимого `bx` и 4.

В режиме относительной адресации обычно выполняется доступ к массивам данных. Смещение указывает на первый элемент массива, а регистр базы определяет положение других элементов внутри массива.

Индексная адресация (indexed addressing) — эффективный адрес операнда вычисляется путём суммирования значения прямого адреса и индекса. Для индекса чаще всего используются регистры `si` и `di`. Например:

```
                MyTable   equ  5000h
8B 8C 00 50     mov cx, MyTable[si]
```

Команда загружает в регистр `cx` значение, содержащееся по адресу, равному сумме `si` и значения MyTable. Отметим, что использование в качестве метки слова TABLE привело бы к предупреждению, о том, что слово TABLE является зарезервированным словом в языке ассемблер.

Этот способ адресации обычно применяется для доступа к элементам массива, начинающегося с адреса MyTable. Иногда говорят по-другому: этот способ удобен для доступа к элементам статического вектора (вектора, адрес базы которого не меняется во время выполнения программы).

Относительная (базовая) индексная адресация (based indexed addressing) — эффективный адрес операнда вычисляется путем суммирования базового регистра `bx` или `bp`, индексного регистра `si` или `di` и смещения, указанного в команде. Например:

```
                Element   equ  5000h
89 81 00 50     mov word ptr ds:[bx+si+Element],ax
```

Команда помещает в ячейку памяти, адрес которой равен сумме значения `Element`, базы `bx` и индекса `si`, содержимое регистра `ax`. Это способ адресации обычно применяется для доступа к структурированным массивам данных [6]. Структура, состоящая из 3-х массивов, представлена на рисунке 4.3. Для доступа к отдельным элементам массива, регистр базы указывает на базу структуры, а смещение (хранящееся в `DI`) содержит расстояние между началом структуры и началом одного из массивов. Положение элемента в массиве определяется значением содержащемся в переменной элемента.

Неявная адресация — в обозначении команды подразумевается один из операндов – например, `cli` — команда запрета прерываний, которая сбрасывает

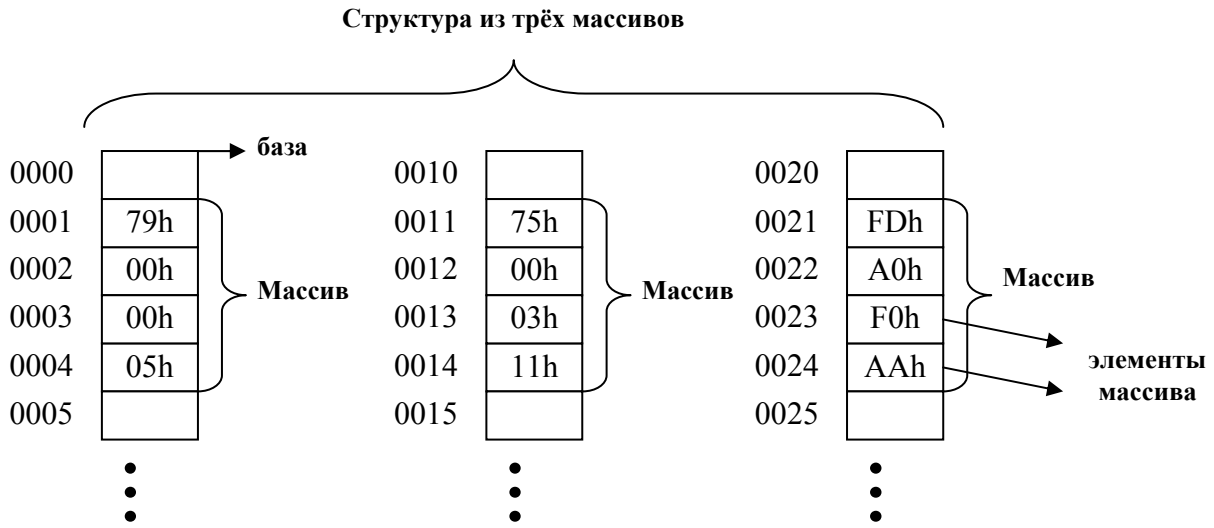


Рисунок 4.3 — Относительная индексная адресация на примере доступа к структуре из трёх массивов

флаг `IF` в 0. Операнд в этой команде не указывается явно. Следует отметить, что есть и безадресные команды, например, такие как `HALT`, `NOP`.

Формат команд. Формат инструкции микропроцессоров архитектуры Intel 64 и IA-32 представлен на рисунке 4.4 [10] (электронная версия этого документа есть на прилагаемом диске).

Состав (поля):

Префиксы команды	КОП	Mod R/M	SIB	Смещение	Непосредственный операнд
Instruction Prefixes	Opcode	Mod R/M	SIB	Displacement	Immediate

Число байт:

0 .. 4	1 .. 3	0 .. 1	0 .. 1	0 .. 4	0 .. 4

Рисунок 4.4 — Общий формат инструкции

На рисунке 4.4 названия полей приведены и на русском, и на английском языках.

На этом же рисунке приведено количество байт, которые могут формировать поле. Как видно из рисунка, только в коде операции (КОП) минимальное число байт равно единице, для всех остальных полей минимальное значение равно нулю. Это означает, что все поля, кроме поля кода операции, являются необязательными, т. е. в одних командах они могут присутствовать, а в других — нет.

Однако же эта информация относится ко всем микропроцессорам, имеющим архитектуру архитектуры Intel 64 и IA-32, куда входят микропроцессоры от i80386 до самых современных микропроцессоров, очень сильно отличающиеся по сложности внутреннего устройства, и, как следствие, имеющие различные по количеству наборы команд (но сохраняющие преемственность по отношению к младшим моделям!).

Поэтому не удивительно, что применительно к микропроцессору i80386 формат будет несколько отличаться — а именно, код операции может содержать только до двух байт информации. Посмотреть формат команд для собственно i80386EX можно с помощью программы ApBuilder, которая располагается на диске, поставляемом вместе со стендом. Сведения о работе с этой программой приведены в Приложении Б.

Можно посоветовать перед изучением системы команд i80386 познакомиться с системой команд микропроцессора i8086/8088, поскольку подавляющее большинство команд микропроцессору i80386 «достались» от микропроцессора i8086. Например, по классификации [8] (достаточно условной, по этой классификации используется одна команда mov, которая, на самом деле, за счёт различных методов адресации имеет 27 вариаций) из 83 команд микропроцессора i80386 шестьдесят шесть команд существовали в микропроцессоре i8086/8088, ещё семь появились в микропроцессоре i80186 и только десять команд появились в i80386. Однако следует отметить, что появившиеся 10 команд очень важны для специалистов, работающих с аппаратурой — это команды работы с битами. Кроме того, можно сказать, что хотя некоторые команды считаются командами

микропроцессора i8086, в процессоре i80386 они стали обладать более широкими возможностями. Например, команды работы со стеком в микропроцессоре i80386 получили возможность использовать непосредственную адресацию. Для более подробного знакомства с системой команд i8086/8088 рекомендуется использовать классические книги [9, 11], а также [12].

Кратко рассмотрим назначение полей. При этом мы будем руководствоваться не порядком следования полей в формате команды, а частотой появления этих полей в различных командах.

Код операции описывает операцию, выполняемую командой — всегда присутствует в команде! Младший значащий бит КОП w (от слова word) может означать, что команда оперирует со словом (бит равен 1) или с байтом (бит равен 0).

Все остальные поля являются необязательными и зависят от кода команды, методов адресации и необходимости модифицировать процесс выполнения команды.

Поле Mod R/M. Поле Mod R/M (иногда называемое постбайтом) расширяет код операции и несёт дополнительную информацию об операндах, способе адресации и/или информацию для уточнения КОП. Формат поля Mod R/M представлен на рисунке 4.5 [4, 10]. Некоторые команды содержат сведения об операндах внутри кода операции (КОП), а некоторые — внутри байта Mod R/M. Байт состоит из трёх полей: Mod (от слова Mode — режим) — старшие 2 бита, Reg (от слова Register) — 3 бита, которые определяют номер регистра, и R/M — показывает либо один из 8 регистров общего назначения, либо указывают режим адресации к памяти (Memory). Подробная расшифровка полей приведена в таблицах 4.1 и 4.2 [3, 4, 10, 11, 12].

Mod		Reg/КОП			R/M		
7	6	5	4	3	2	1	0

Рисунок 4.5 — Формат поля Mod R/M

Таблица 4.1 — Расшифровка поля mod

Код поля Mod	**Влияние на смещение или выбор регистра**

00	Смещение отсутствует
01	Команда содержит 8-битовое смещение, которое расширяется со знаком до 16 бит
10	Команда содержит 16 битовое смещение
11	Операнд содержится в регистре, определяемом полем R/M

Если поле Mod = 11b, то два следующих поля будут использоваться для указания регистров (оба операнда команды находятся в регистрах). Каждый регистр имеет свой код. В этом случае расшифровка полей Reg и R/M представлена в таблице 4.2. Для того чтобы идентифицировать тип операндов, с которыми мы работаем, используется бит w (word). Бит w находится в нулевом разряде КОП [3].

Таблица 4.2 — Коды регистров в полях reg

Код	**w = 0**	**w = 1**
000	al	ax
001	cl	cx
010	dl	dx
011	bl	bx
100	ah	sp
101	ch	bp
110	bh	si
111	dh	di

По байту Mod R/M нельзя точно установить разрядность регистра. Размер регистра определяется по коду операции (младший значащий бит) и префиксам размера операндов.

Если поле Mod = 00b, 01b, 10b, то поле R/M используется вместе с полем Mod для идентификации определенного типа адресации. В команде допустима любая комбинация в полях Mod и R/M, что обеспечивает многообразие режимов адресации памяти в семействе микропроцессоров [9].

Поле R/M всегда обозначает либо регистр, либо режим адресации [13]. Поле Reg в некоторых командах используются для идентификации группы, к которой

относится машинная команда (например, команда and в группе логических команд), т.е. эти биты уточняют КОП.

Таблица 4.3 [11] поясняет использование поля M/R.

Таблица 4.3 — Кодировка поля M/R

Mod / R/M	00	01	10	11	
				w=0	w=1
000	bx+si	bx+si+d8	bx+si+d16	al	ax
Сегментный регистр	ds	ds	ds		
001	bx+di	bx+di+d8	bx+di+d16	cl	cx
Сегментный регистр	ds	ds	ds		
010	bp+si	bp+si+d8	bp+si+d16	dl	dx
Сегментный регистр	ss	ss	ss		
011	bp+di	bp+di+d8	bp+di+d16	bl	bx
Сегментный регистр	ss	ss	ss		
100	si	si+d8	si+d16	ah	sp
Сегментный регистр	ds	ds	ds		
101	di	di+d8	di+d16	ch	bp
Сегментный регистр	ds	ds	ds		
110	d16	bp+d8	bp+d16	dh	si
Сегментный регистр	ds	ss	ss		
111	bx	bx+d8	bx+d16	bh	di
Сегментный регистр	ds	ds	ds		

Поля *Смещение и Непосредственный операнд*, как следует из их названия, используются при определённых методах адресации, задаваемых кодом операции и постбайтом Mod R/M. Количество байт, задействованных в этих полях зависит от объекта, к которому происходит адресация и разрядности данных.

Префиксы — один или несколько байт, предшествующих команде и модифицирующих операцию этой команды. Имеются четыре группы префиксов:

1 Префиксы блокировки и повторения:

`0F0h` — `lock` (префикс блокировки);

`0F2h` — `repnz` (префикс повторения, если `ecx>0` и `z=0`, только для строковых инструкций);

`0F3h` — `rep` (префикс повторения, если `ecx>0` и `z=1`, только для строковых инструкций).

Префиксы повторения используются с командами обработки строк, заставляют команду воздействовать на каждый элемент строки.

2 Префиксы переопределения сегмента:

```
02Eh – cs;
036h – ss;
03Eh – ds;
026h – es;
064h – fs;
065h – gs;
```

Префикс переопределения в явной форме указывает, какой сегментный регистр должна использовать команда. Префикс отменяет действующий по умолчанию выбор сегментного регистра, обычно осуществляемый МП 80386 при выполнении команды.

3 Префикс переопределения размеров операндов:

`066h`

Переключает разрядность операндов, устанавливая их 32-разрядными или 16-разрядными.

4 Префикс переопределения размеров адреса:

`067h`

Префикс переключает разрядность адреса, определяя образование 32-разрядных или 16-разрядных адресов.

Поле *SIB*. Определенные значения кода поля Mod R/M могут указывать на наличие второго адресного байта SIB [4]. Байт SIB используется для определения индексного регистра и регистра базы при относительной, индексной, относительно индексной адресации. Формат поля SIB представлен на рисунке 4.6 [4]:

Scale		Index			Base		
7	6	5	4	3	2	1	0

Рисунок 4.6 — Формат поля SIB

В поле Scale помещается масштабный множитель для поля Index, на который его содержимое будет умножаться при вычислении эффективного адреса. Допустимыми значениями являются 1, 2, 4, 8. Поля Index и Base используются для хранения номеров индексного и базового регистров соответственно. Эти регистры применяются для вычисления эффективного адреса операнда.

Вообще-то существуют некоторые соглашения, касающиеся формата байта команды [12]. Например, можно считать, что в КОП, как уже говорилось разряд 0 (LSB) w задаёт слово (если бит установлен, то операнд 16-битный, если нет — 8-битный), а разряд 1 задаёт направление пересылки. Если бит 1 установлен в единицу, то операндом-приёмником является регистр, задаваемый полем Reg в байте Mod R/M. Если же бит 1 установлен в 0, то регистр, задаваемый полем Reg в байте Mod R/M является операндом-источником. Следовательно, этот бит может быть назван битом d – destination.

Как же на практике применить всё это? Следуя принципу, приписываемому Исааку Ньютону «При изучении наук примеры полезнее правил», рассмотрим применение вышеизложенного к некоторым командам mov. Если обратиться к таблице «Система команд i80386» в Приложении А, то мы увидим, что команда mov стоит в 27 ячейках этой таблицы. Если воспользоваться, например, [9], то мы увидим, что все команды mov имеют один байт КОП и или постбайт Mod R/M, или смещение, или непосредственный операнд.

Необходимость знать коды возникает в двух случаях. Первый случай — мы знаем, какая должна быть выполнена команда, и должны узнать код этой команды.

Перевод мнемоники команд в коды называется ассемблированием. Обычно эту работу выполняет специальная программа, называемая ассемблером. На диске, прилагаемом к стенду, находится программа TASM5, которая и решает эту задачу. Как пользоваться этой программой описано в приложении В. Однако эту задачу можно решить и без компьютера. Для этого можно воспользоваться таблицей из приложения А и сведениями, изложенными в данном разделе.

Найдём код команды `mov bx,ax`, рассмотренной выше. Начинаем поочерёдно просматривать все 27 ячеек в таблице «Система команд i80386» в Приложении А в поисках наиболее подходящего варианта. Нам, наверное, должно подходить что-то вроде `mov` с операндом-источником `ax`, или с операндом-приёмником `bx`, или, в общем случае, с `r16` (последнее означает, что в качестве одного из операндов может быть использован любой 16 битный регистр). Нашим условиям отвечают ячейки `0A1h` (`mov ax,mem16`), `089h` (`mov r/m,r16`) и `08Bh` (`mov r16,r/m`). Код `0A1h` мы отбросим, так как в качестве второго операнда используется 16 битная ячейка памяти (`mem16`).

Нужно выбрать из двух оставшихся кодов: `089h` и `08Bh`. Оба байта имеют в качестве LSB значение «единица», указывая, что в качестве операнда будет использоваться слово. Отличия заключаются в значении бита 1 (d). Установленный в 1 бит говорит, что приёмником будет регистр, поэтому вариант `08Bh` видится более предпочтительным. На самом деле это явно прописано и в самой таблице: запись `mov r16,r/m`, соответствующая коду `08Bh`, и говорит, что приёмником является регистр.

Теперь осталось выяснить код байта Mod R/M. В соответствии с таблицей 4.1 два старших разряда должны быть равны 1 («Операнд содержится в регистре, определяемом полем R/M»). По таблице 4.2, учитывая, что регистром приёмником должен быть регистр bx, задаём значения поля Reg (биты 5:3) — `001`. Из таблицы 4.3 видим, что для задания в поле R/M регистра ax нужно установить биты 2:1 в 0.

Итого получаем, что байт Mod R/M должен иметь значение `11001000B` = `0D8h`, а полностью код команды `mov bx,ax` записывается как `08Bh 0D8h`.

Вторая задача, требующая знания кодов, это задача восстановления команд на языке ассемблер из кодов команд, она называется дизассемблированием. Существуют специальные программы, называемые дизассемблерами, которые решают такую задачу. А мы попробуем дизассемблировать, например, код `088h 017h` «вручную».

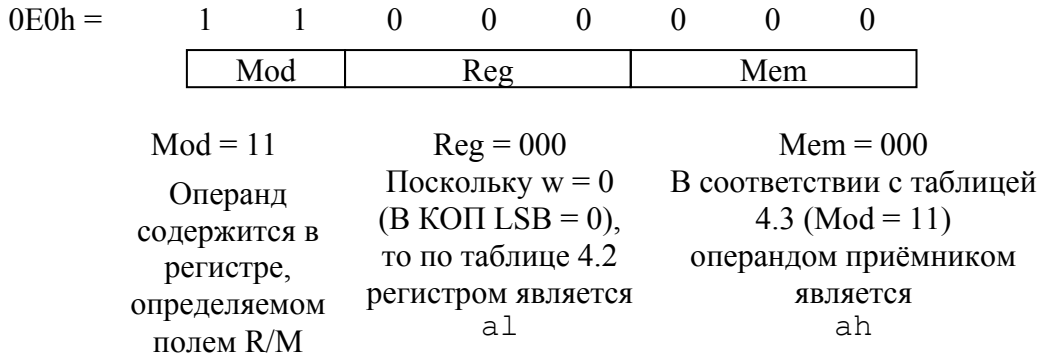
В ячейке 088h имеем `mov r/m,r8`, что означает перемещение 8 битного регистра (r8) в область, задаваемую r/m. Следовательно, это команда является двух байтовой, с байтом Mod R/M.

Таким образом, первый байт является байтом кода операций `mov`, второй — Mod R/M. Два младших бита КОП, равные 0, говорят о том, что команда будет использовать 8-битные операнды, и регистр, задаваемый полем Reg, будет операндом-источником.

Попробуем расшифровать второй байт — Mod R/M = 017h:

Следовательно, код должен трактоваться как команда `mov ds:[bx],dl`. Поскольку обычно для данных сегментным регистром является регистр `ds`, то допустима запись: `mov [bx],dl`.

Чтобы оценить влияние байта Mod R/M рассмотрим ещё команду `088h 0E0h`:



Рассмотрим теперь код `0B0h 001h`, встречающийся в приведённом выше примере программы. Как следует видно из таблицы в Приложении А, ячейке с кодом `0B0h` соответствует запись `mov al,im8`. Отсюда следует, что это должна быть команда без байта Mod R/M. И два младших бита уже не играют роль битов w и d. Длина команды должна составлять два байта: один байт — код операции и один байт — непосредственный операнд (im — immediate, непосредственная адресация, 8 обозначает разрядность операнда). Суть общепринятых обозначений для команд микропроцессора i80386 (правда, на английском языке) можно понять, если воспользоваться приложением ApBuilder, работа с которым описана в Приложении Б.

Дизассемблируем такую последовательность байт: `066h 0B8h 0FFh 0EEh 0DDh 0CCh`. Из таблицы «Система команд i80386» находим, что `066h` соответствует «opSize prefix», т. е. этот байт является, как говорилось выше, префиксом переопределения размеров операндов. Следующий байт `0B8h` по таблице трактуется как `mov ax,im16`. Таким образом, префикс, увеличивающий размер операнда, должен переопределить размер регистра на 32-битный (т. е. на `eax`), и задать размер непосредственного операнда в 32 бита. Тогда следующие за кодом операции 4 байта должны рассматриваться как операнд-источник. Следовательно, мы смело можем утверждать, что последовательность байт `066h 0B8h 0FFh 0EEh 0DDh 0CCh` является командой `mov eax, 0CCDDEEFFh`.

Надеемся, что идея кодировки/раскодировки команд понятна, и вы сможете произвести интерпретацию команд, встречающихся в лабораторных работах, самостоятельно.

4.1.2 Задание к работе № 1

Во всех заданиях N является номером вашего варианта в лабораторном журнале.

4.1.2.1 Задание для домашней подготовки

1 Составьте блок-схему алгоритма программы для выполнения пункта 6 подраздела 4.1.2.2 Задание для выполнения в лаборатории.

2 Занесите в заготовку отчёта блок-схемы алгоритмов по всем пунктам Задания для выполнения в лаборатории.

3 Найдите коды команд и составьте программы по пунктам 2 – 7 задания для выполнения в лаборатории. Занесите полученные команды в заготовку отчёта. В качестве примера используйте оформление пункта 1 Задания для выполнения в лаборатории.

4.1.2.2 Задание для выполнения в лаборатории

1 Занесите в ОЗУ и выполните следующую программу:

№ команды	Адрес	Код (hex)	Мнемоника	Комментарий
1	4000h	BA 34 12	mov dx, 1234h	;поместить в dx число 1234h
2	4003h	8B CA	mov cx, dx	;переслать это число в cx
3	4005h	CC	int 3	;остановить программу

2 Запишите число N, равное вашему варианту, в аккумулятор. Воспользуйтесь командой `mov ax, Nh`. Код операции: B8 Nh.

3 Перешлите это число в регистр `bx`.

4 Перешлите это число по адресу памяти 6000h+N с помощью команды `mov`, с использованием косвенно-регистровой адресации.

5 Проделайте то же самое с использованием прямой адресации.

6 Занесите в память и отладьте программу для записи в память массива из 20 элементов со стартовым адресом 7000h+N с использованием индексной адресации. Для чётных вариантов массив должен состоять из слов, для нечётных — из байт.

7 Используя прямую адресацию, прочитайте элемент вашего массива, номер которого равен 2N-1, в один из регистров общего назначения.

Проделайте все задания в потактовом, поцикловом и покомандном режимах, поясняя состояния всех светоизлучающих диодов на системных шинах и содержимое регистров. При выполнении работы необходимо помнить, что в соответствии с картой памяти по умолчанию программа пользователя начинается с адреса 4000h.

4.1.3 Контрольные вопросы

1 Перечислите методы адресации, которые можно использовать при работе с командами группы пересылки данных. Приведите примеры.

2 Опишите структуру формата инструкции МП 80386.

3 Опишите структуру байта modr/m.

4 Что такое префикс в формате команд и для чего он используется?

5 В каких командах и для чего используется поле «смещение» (см. формат команд)?

6 Проведите сравнительный анализ методов адресации.

7 Перечислите регистры общего назначения, доступные в реальном режиме и приведите их предпочтительное использование.

8 Охарактеризуйте основные группы системы команд микропроцессора 80386. Приведите примеры команд той или иной группы.

9 Как зависит длина команды от метода адресации?

10 Какие флаги находятся в регистре флагов?

11 Приведите группы команд модифицирующие и не модифицирующие флаги МП.

12 Сколько байт занимает команда, и что обозначает каждый байт? Варианты команд представлены в таблице 4.3.

13 Что такое карта памяти? Приведите карту памяти учебного стенда.

14 Сгруппируйте и опишите все программно доступные в реальном режиме регистры i80386.

Таблица 4.3 — Варианты команд

Вариант	**Команда**	**Вариант**	**Команда**
1	`mov ds:[0a2ah], 0400h`	9	`mov cx, ss:[09ffh]`
2	`mov es:[0c2ch], 04h`	10	`mov al, [bx][di]`
3	`mov bx, ax`	11	`mov al, cl`
4	`movs [di], [si]`	12	`mov di, ax`
5	`mov ax, 00h`	13	`mov [bx][di], dl`
6	`mov al, 88h`	14	`mov [cx], ds:[0777h]`
7	`mov [di], 00h`	15	`mov cs:[0a00h], ax`
8	`mov cx, [dx]`		

4.2 Лабораторная работа № 2. Команды управления переходами. Подпрограммы и стек

Цель работы: изучение особенностей выполнения команд управления переходами, обращения к подпрограммам, возврата из них, использования стека. Изучение стандартных методов передачи параметров подпрограмме.

4.2.1 Теоретические сведения

Для того чтобы иметь возможность переходить по фиксированному адресу и создавать ветвящиеся алгоритмы в системе команд x86 имеются специальные команды безусловного (например, `jmp`) и условного перехода (например, `jc`).

Команды безусловного перехода можно разбить на команды относительного (относительно значения `eip` после выполнения команды `jmp`) перехода и абсолютного перехода. Адрес при относительном переходе определяется смещением со знаком, следующим за кодом команды. Среди относительных переходов различают «короткие» (jmp short) переходы (код `0EB`h и смещение в виде байта), «ближние» (jmp near), иногда называемые внутрисегментными переходами, (код `0E9`h и смещение в виде двух байт) и переходы со смещением в виде четырёх байт.

В командах абсолютного перехода адрес может быть задан прямо в виде значения сегмента и смещения («длинный» абсолютный переход (`jmp far`), код `0EA`h) и косвенно, через значения, содержащиеся в регистрах или памяти («длинный» косвенный переход, адрес содержится в регистрах общего назначения или в памяти, код `0FF`h).

При выполнении команд условного перехода сначала проверяется соответствующее условие (состояние какого-либо флага), и, только если это условие истинно, осуществляется переход, иначе выполняется следующая команда. Для

организации циклов используются команды условного перехода группы loop, выполнение которых зависит от содержимого регистра-счётчика (e)cx.

Рассмотрим в качестве примера команды jmp short Address (короткий переход) и jmp far cs:ip.

Команда короткого перехода применяется, когда нужно осуществить переход в пределах от минус 128 до плюс 127 байт. Адрес задаётся в виде байта смещения, интерпретируемого как число со знаком в дополнительном коде. Пример команды jmp short Address:

```
    ORG          04020h
    MyDataAddress    dw  0
    ORG      4000h
EB 1E        jmp short MyDataAddress
```

Довольно часто используют следующую форму записи короткого перехода:

```
EB 1E        jmp $+10
```

Здесь знак $ означает положение ip. Объясните, почему в коде команды стоит число 1Eh.

Мнемоника длинного перехода ассемблером не воспринимается, поэтому обычно длинный переход задаётся в виде последовательности байт:

```
DB 0EAh, 20h, 00, 00h, 00h,
```

где 0EAh — код команды длинного абсолютного перехода

Подпрограммы. При разработке программ нужно стараться сделать их как можно короче и компактнее. С этой целью часть программы, которая неоднократно повторяется, или программа, которая часто используется, могут быть оформлены в виде подпрограмм. *Подпрограмма* — последовательность команд, заканчивающаяся командой возврата, выполнение которой может быть вызвано из любого места программы любое количество раз.

Подпрограммы изначально появились как средство оптимизации программ по объёму занимаемой памяти и экономии времени программиста. В настоящее время, кроме этого, подпрограммы выполняют ещё функцию структуризации программы с

целью удобства её понимания и сопровождения, что особенно важно в языках высокого уровня.

При вызове подпрограммы, как и в случае выполнения команды перехода, управление передаётся команде, адрес которой содержится в команде вызова подпрограммы. Но, в отличие от команд перехода, вызов подпрограммы, после выполнения собственно подпрограммы, должен обеспечить выполнение команды, следующей за командой вызова подпрограммы. Поэтому любая подпрограмма должна заканчиваться командой возврата из подпрограммы. Для обеспечения механизма возврата в точку вызова используется специальное средство, называемое **стеком** (**stack**).

Стек — память, функционирующая по так называемому принципу LIFO (Last In, First Out — первым вошёл, последним вышел), и первоначально предназначенная для хранения адресов возврата из подпрограмм. Стеки бывают аппаратные и программные, в микропроцессоре i80386 используется программный стек, то есть стек размещается в ОЗУ. Стек адресуется при помощи пары регистров `ss:sp`. Регистр `ss` (stack segment) содержит адрес сегмента стека, `(e)sp` (stack pointer, указатель стека) — смещение относительно начала сегмента стека.

Процесс передачи управления подпрограмме, называется вызовом подпрограммы. Для вызова подпрограммы и возврата из нее в языке ассемблера используются команды `call` и `ret`, которые при выполнении используют стек. При необходимости подпрограмме можно передавать аргументы (*входные параметры*), в качестве которых могут выступать либо непосредственно данные, либо их адреса. Результаты работы программы, передаваемые по окончании её работы в основную программу, называются *выходными параметрами*.

Синтаксис команды `call` показан на рисунке 4.7. Необязательные модификаторы `near` и `far` указывают тип вызова: соответственно ближний (внутрисегментный), когда вызываемая подпрограмма находится в том же сегменте, что и вызывающая, или дальний (межсегментный), когда вызываемая подпрограмма находится в другом сегменте. Отсутствие модификатора указывает на ближний

вызов. Вместо символического имени подпрограммы в команде можно указать её адрес.

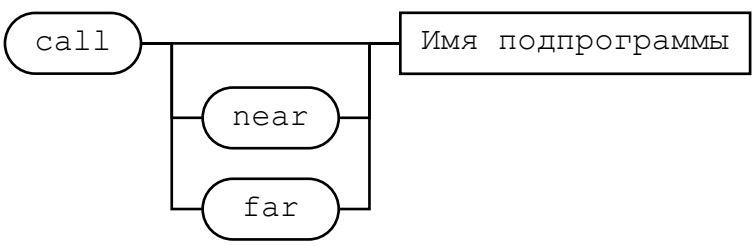


Рисунок 4.7 — Синтаксис команды `call`

В зависимости от типа вызова команда `call` адреса возврата сохраняются несколько по-разному. При ближнем вызове микропроцессор сохраняет в стеке содержимое регистра `ip`, которое, по сути, является адресом возврата (адресом следующей за `call` команды), после чего помещает в регистр `ip` адрес перехода. Что происходит при дальнем вызове, читателю предлагается узнать самостоятельно.

Команда `ret` (ближний возврат) берёт из стека последнее, помещённое туда слово, и помещает его в регистр `ip`, после этого выполнение программы продолжится с этого адреса. Как уже говорилось, любая подпрограмма должна заканчиваться командой возврата (`ret`, `retf`, `iret`, `ret N`). Самостоятельно выясните, в чем различие этих команд возврата.

Автоматическое сохранение и восстановление адреса основной программы при выполнении подпрограммы позволяет сделать подпрограммы вложенными (nested procedures), т. е. осуществлять вызовы одной подпрограммы из другой. Уровень вложенности определяется лишь размером стека.

Для работы со стеком можно использовать команды `push r` (записать в стек содержимое обозначенного регистра) и `pop r` (записать данные из стека в обозначенный регистр). Эти команды являются однобайтовыми [3]. Начиная с микропроцессора i80386 при работе со стеком можно использовать непосредственную адресацию.

Рассмотрим работу МП со стеком на примере команды `pushf`.

`pushf` — команда сохраняет младшее слово регистра флагов `eflags` в стеке, т. е. сохраняет 16-разрядный регистр флагов `flags`. Если рассматривать процесс выполнения команды, то можно отметить, что при 8-битной шине данных первым в

стек помещается старший байт регистра `flags` по адресу `sp – 1`, а потом младший байт по адресу `sp – 2`, где `sp` — адрес ячейки до обращения к стеку (т. е. до выполнения команды `pushf`). Такой формат размещения в стеке называется Little endian (или Intel формат). При работе с 16-битной шиной данных в стек будут сразу помещаться слова. Пример работы со стеком представлен на рисунке 4.8. После выполнения команды `sp` = 21FEh, — указывает на младший байт регистра `flags`.

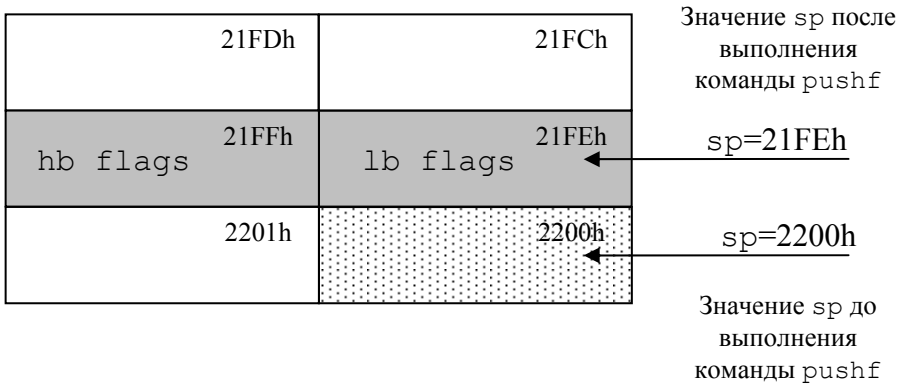


Рисунок 4.8 — Работа МП со стеком на примере команды `pushf`

При извлечении значения из стека (по команде `popf`), первым извлекается младший байт слова, а затем старший, а `sp` увеличивается на 2.

Таким образом, при записи данных в стек, значение `sp` уменьшается, т. е. стек растёт по направлению к младшим адресам. Если используется представление памяти с младшими адресами, расположенными вверху, то говорят, что стек растёт вверх.

Как следует из приведённых выше сведений, значение указателя стека `sp` всегда должно быть чётным.

Передача параметров. Часто основная программа должна передать вызываемой подпрограмме определённую информацию, данные, с которыми подпрограмма должна работать. И часто подпрограмма после завершения работы должна передать результат своего выполнения в вызвавшую её программу.

Передаваемые подпрограмме данные называют параметрами или, иначе, аргументами. Существует три общепринятых способа передачи параметров: через регистры, через общую область памяти и через стек [14].

Передача параметров в регистрах очень проста. Для этого нужно просто поместить значения-параметры в соответствующие регистры и вызвать подпрограмму. Каждая подпрограмма может иметь свои собственные потребности в параметрах, и чтобы избежать путаницы, лучше выработать некоторые соглашения передачи параметров и придерживаться их. Например, можно следовать правилу, согласно которому первый параметр-указатель всегда передается в регистре `bx`, второй — в `si` и т.д. Следует взять за правило особенно аккуратно комментировать каждую подпрограмму — какие параметры она получает, в каких регистрах они находятся.

В большинстве языков высокого уровня, включая Паскаль и Си, и в подпрограммах на Ассемблере, вызываемых из этих языков [14], передача параметров обычно осуществляется через стек. Передача параметров через стек несколько более сложна и отличается значительно меньшей гибкостью, чем передача через регистры. Передача параметров заключается в том, что вызывающая программа предварительно помещает в стек в определённом порядке параметры, требуемые для выполнения подпрограммы, а затем вызывает эту подпрограмму. Пример передачи параметров через стек представлен на рисунке 4.9 [10].

Вызванная подпрограмма непосредственно выполняет операции над данными, расположенными в стеке. Поскольку при работе со стеком микропроцессор автоматически использует регистры `ss` и `sp`, то при передаче параметров не рекомендуется записывать в эти регистры сведения о количестве передаваемых параметров. Обычно для этих целей используется регистр `(e)bp`, который по умолчанию также настроен для работы со стеком и в который записывается текущее значение указателя стека `(e)sp`. Выглядит это следующим образом:

```
pushbp
```

```
mov bp, sp
```

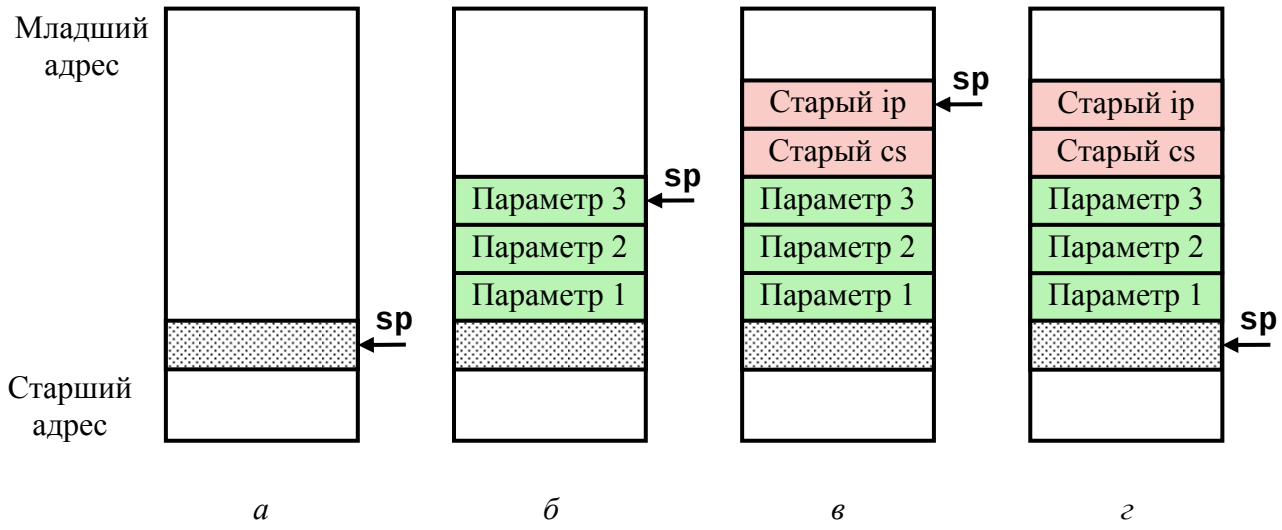


Рисунок 4.9 — Передача параметров через стек: *а* — стек до вызова подпрограммы; *б* — в стек поместили три параметра; *в* — стек после выполнения команды `call`; *г* — стек после возврата из подпрограммы командой `ret 3`

Теперь доступ к стеку можно осуществлять с помощью регистра `bp`.

В конце подпрограммы обычно используют команду `ret N` (`ret` со смещением). При выполнении команда `ret N` дополнительно добавляет значение смещения N к указателю стека `sp`. Это позволяет игнорировать параметры, помещаемые в стек перед вызовом подпрограммы. Рассмотрим на примере, представленном на рисунке 4.9, выполнение команды возврата без смещения и со смещением. Если при выходе из подпрограммы мы будем использовать команду `ret`, то после выполнения этой команды из стека будет восстановлено значение `ip` и `cs`, а `sp` будет указывать на ячейку памяти содержащую параметр 3. Если данная подпрограмма вызывается неоднократно, то рано или поздно наступит переполнение стека. Чтобы этого избежать, необходимо использовать команду `ret N`. При выполнении этой команды из стека будет также восстановлено значение `ip` и `cs`, к этому моменту `sp` будет указывать на параметр 3; затем к значению `sp` будет прибавлено значение смещения, указанного в команде. В нашем случае, если параметры однобайтные, то значение смещения должно равняться трём. В

результате `sp` будет иметь исходное значение, которое было до загрузки параметров и вызова подпрограммы.

Для того чтобы использовать подпрограммы на Ассемблере в программах, написанных на языках высокого уровня, необходимо знать и корректно использовать правила (соглашения) передачи параметров через стек, принятые в соответствующем языке высокого уровня.

При передаче параметров, как через регистры, так и через стек, передаваться могут как непосредственно сами значения параметров, так и указатели на них. Обычно в роли указателя выступает адрес ячейки памяти, где и расположен сам параметр. При работе с указателями для получения доступа к значению параметра часто используются следующие методы адресации: косвенно-регистровая (indirect), относительная (base-displacement addressing), индексная (indexed addressing) и относительно-индексная (based indexed addressing).

4.2.2 Алгоритмы программ для выполнения лабораторной работы.

Простейший пример алгоритма вызова подпрограммы представлен на рисунке 4.10.

Алгоритм программы простой временной задержки приведён на рисунке 4.11. Общее время задержки вычисляется по формуле:

$$T_D = t_1 + (t_2 + t_3 + t_4)\cdot N + t_5, \qquad (4.1)$$

где N — число повторения цикла подпрограммы. Блок алгоритма «Нет операции» (тело цикла) может содержать команду `nop` или любые другие последовательности команд, выполнение которых не изменяет содержимое регистров МП. Этот блок необходим для увеличения времени выполнения цикла, а, следовательно, и общей задержки.

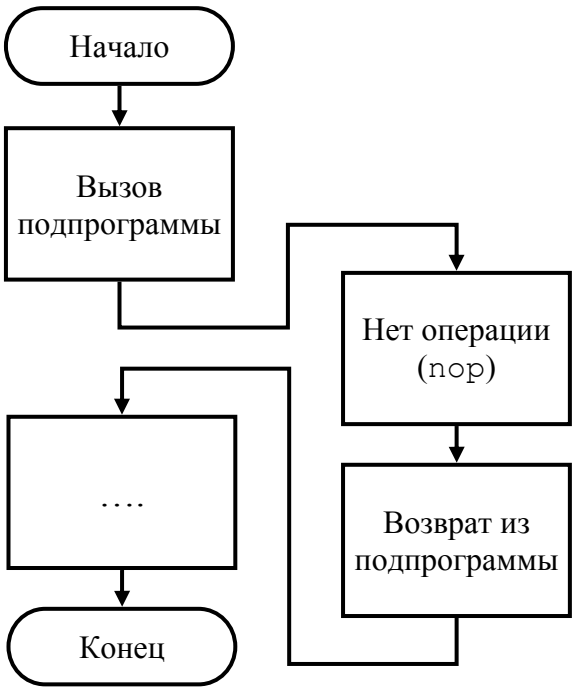


Рисунок 4.10 — Блок-схема алгоритма вызова подпрограммы.

Время $t_1 + t_5$ фиксировано и в цикл не входит. Минимальная задержка для приведённого алгоритма определяется при N = 01h и равна $T_{DMIN} = t_1 + t_2 + t_3 + t_4 + t_5$. Максимальная задержка имеет место при N = 00h и вычисляется по формуле $T_{DMAX} = t_1 + (t_2 + t_3 + t_4)\cdot 256 + t_5$.

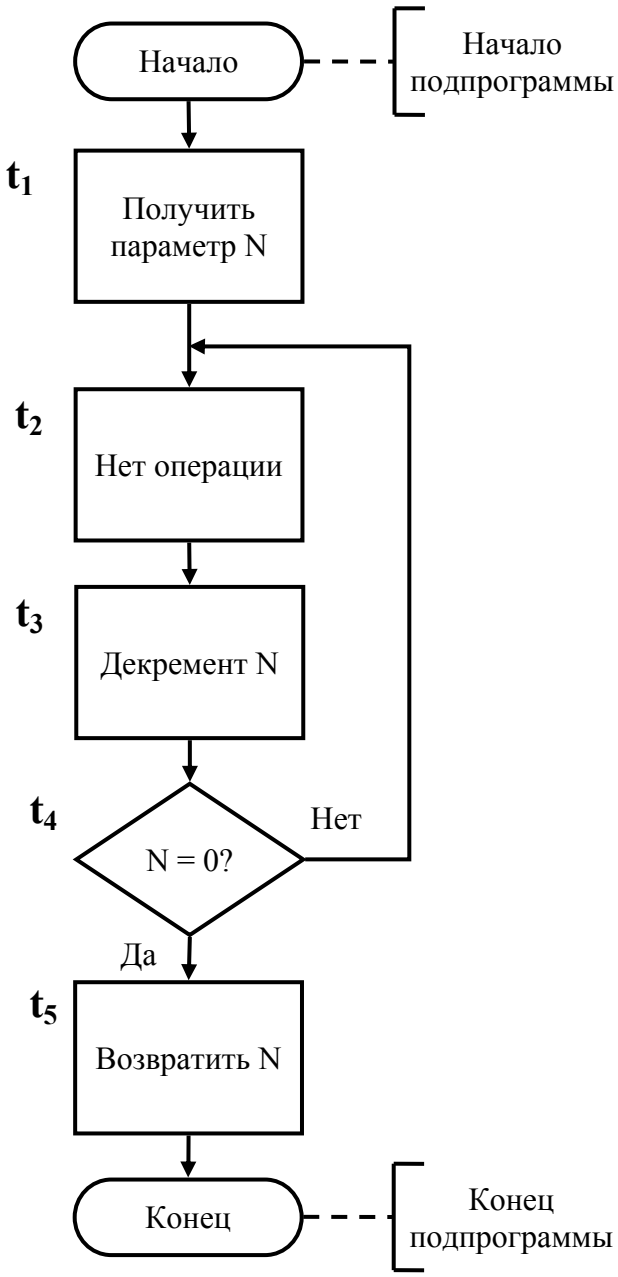


Рисунок 4.11 — Блок-схема алгоритма подпрограммы временной задержки

4.2.3 Подпрограммы учебного лабораторного стенда, выполняющие стандартные функции

Во втором пункте задания к работе потребуется вызвать подпрограммы учебного лабораторного стенда выполняющие стандартные функции. Подпрограммы, выполняющие стандартные функции, — это уже написанные

программы, хранящиеся в ОЗУ по определённым фиксированным адресам. Адрес и входные параметры функции представлены в таблице 4.3.

Таблица 4.3 — Подпрограммы, выполняющие стандартные функции

Адрес подпрограммы (CS:IP)	**Описание подпрограммы (функции)**	**Входные параметры подпрограммы**
0000:1010h	Функция вывода числа (от 0 до F) на 7-сегментный индикатор	Регистр `al` — выводимая цифра (от 0 до F), Регистр `cl` — разряд индикатора, в который выводится цифра.
0000:1060h	Функция подачи звукового сигнала	Регистр `bl` — коэффициент частоты звука.

Функция вывода числа на 7-сегментный индикатор выводит в одно или два младших знакоместа индикатора заданное число. Знакоместо индикатора и выводимое число задаются через входные параметры, передаваемые по значению. Параметры требуется определить перед вызовом функции, загрузив значением регистры `al` и `cl`. В регистр `al` загружается шестнадцатеричное число, которое требуется вывести на индикаторы (00h — 0Fh). Переводить число в код 7-сегментного индикатора не требуется, это делает сама подпрограмма вывода числа. В регистр `cl` загружается код, который определяет, с каким знакоместом индикатора мы будем работать. На рисунке 4.12 представлена нумерация знакомест индикатора. Если мы хотим вывести число в нулевое знакоместо, то нам необходимо загрузить в регистр `cl` число 02h, если же в первое знакоместо, то число 01h. Если хотим чтобы ни одно из этих знакоместо не светилось, тогда загружаем число 03h. Если хотим чтобы светилось сразу два знакоместа, тогда загружаем 00h. Схема подключения 7-сегментных индикаторов представлена в разделе 4.4.1. Кстати, из этого рисунка должно быть ясно, как связаны значения, передаваемые через регистр `cl` со светящимися знакоместами. Аналогично можно работать и с другими знакоместами 7-сегментных индикаторов

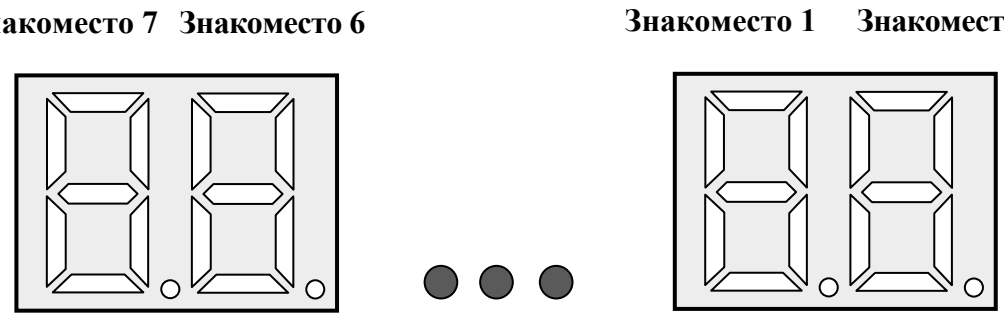


Рисунок 4.12 — Нумерация знакомест 7-сегментных индикаторов

Для вызова подпрограммы необходимо использовать команду `call far` (дальний вызов), требующую указания `cs` и `ip`. Фактические значения `cs` и `ip` представлены в таблице 4.3.

Следующая функция — функция подачи звукового сигнала. Эта функция берёт входной параметр из регистра `bl`, поэтому в него также необходимо записать определенное значение перед вызовом функции. При вызове функция выдает на звуковой излучатель меандр, частота которого зависит от входного параметра, передаваемого в регистре `bl`. Для вызова подпрограммы, аналогично, используется команда дальнего вызова `call far`.

4.2.4 Задание к работе № 2

Во всех заданиях N — номер варианта.

4.2.4.1 Задание для домашней подготовки

1 Составьте блок-схемы алгоритмов программ для выполнения пунктов 6 и 7 подраздела 4.2.4.2 Задание для выполнения в лаборатории.

2 Занесите в заготовку отчёта блок-схемы алгоритмов по всем пунктам Задания для выполнения в лаборатории.

3 Найдите коды команд и составьте программы по пунктам 1 – 7 задания для выполнения в лаборатории. Занесите полученные команды в заготовку отчёта.

4.2.4.2 Задание для выполнения в лаборатории

1 Продемонстрируйте выполнение команды безусловного перехода.
2 Продемонстрируйте выполнение команды условного перехода.
3 Отладьте и выполните программу в соответствии с алгоритмом, представленным на рисунке 4.10. Выполните свою программу по тактам. Проанализируете работу МП со стеком.
4 Вызовите стандартную функцию вывода числа на сегментный индикатор. Адрес и входные параметры функции представлены в таблице 4.3. Запустите программу в автоматическом режиме. Напомним, что при выполнении программ в автоматическом режиме, в конце программ необходимо ставать точку останова (команду `int 3`).
   а) выведите на сегментный индикатор номер вашего варианта в шестнадцатеричной системе счисления;
   б) выведите на сегментный индикатор номер в Dec формате равный N×6. Выводиться должно двухразрядное число;
   в) выведите на сегментный индикатор тот же номер в Hex формате.
5 Вызовите функцию подачи звукового сигнала, с входным параметром (коэффициентом частоты звука) равным N×11h.
6 Отладьте и выполните программу, соответствующую алгоритму, представленную на рисунке 4.11. Продемонстрируйте вызов своей подпрограммы с передачей параметра N, ее покомандное исполнение и завершение с возвратом параметра 2N в основную программу. Для чётных вариантов — передать параметр по значению, вернуть по указателю, для нечётных — наоборот.

7 Модифицируйте вызывающую программу и подпрограмму задержки так, чтобы подпрограмма получала параметр N через стек.

4.2.5 Контрольные вопросы

1 Дайте определение подпрограммы. Каково её назначение?
2 В чем отличие подпрограммы от макрокоманды?
3 Что такое стек, и каково его назначение?
4 Чем определяется уровень вложенности подпрограмм? Кратность вызова?
5 Приведите команды работы со стеком.
6 Какие ошибки могут возникнуть при работе со стеком?
7 С помощью каких команд можно задать или изменить область памяти, отведённую под стек?
8 Какими способами можно передать параметры подпрограмме на языке ассемблера?
9 Дайте сравнительный анализ способам передачи параметров подпрограмме.
10 Чем отличается передача параметров по указателю от передачи параметров по значению? Приведите примеры на ассемблере.
11 Объясните различие в формате и в выполнении команд `call` ближнего и дальнего вызова.
12 Объясните различие в формате и в выполнении команды `RET` ближнего и дальнего возврата.
13 Как выполняется команда `ret N`?
14 Чем отличаются команды `ret`, `retf`, `iret`, `ret n`?
15 Сколько байт занимает команда `jmp ptr16:16`, и что обозначает каждый байт?
16 Чем отличаются команды: `jmp rel16` и `jmp ptr16:16`. Привести примеры команд?

4.3 Лабораторная работа № 3. Арифметические действия

Цель работы: изучение особенностей выполнения команд группы преобразования данных, вычисление выражений.

4.3.1 Теоретические сведения

Команды преобразования данных. В микропроцессоре i80386EX к группе команд преобразования данных можно отнести арифметические и логические команды.

В подгруппу арифметических команд входят следующие основные команды (с учётом вариаций методов адресации и флагов): сложения (`add`), инкремента (`inc`), вычитания (`sub`), декремента (`dec`), сравнения (`cmp`), изменения знака (`neg`), команды коррекции ASCII кодов и кодов BCD (`aaa, aax, daa, dax`), умножения (`mul`), умножения со знаком (`imul`), деления (`div`) и деления со знаком (`idiv`).

Отметим наличие у МП i80386 команд, выполняющих арифметические команды со знаком. Например, команда `imul` выполняет операцию знакового целочисленного умножения [4]. Эта команда может оказывать влияние на флаги `CF` и `OF`. При этом состояние флагов `SF`, `ZF`, `AF` и `PF` не определено. Но если требуется выполнить беззнаковое умножение, то необходимо использовать команду `mul`.

К логическим командам относятся команды: сдвигов (`rol, ror, sal, sar, shr` и другие), логических операций «И» (`and`), «ИЛИ» (`or`), «Исключающее или» (`xor`) и «НЕ» (`not`) и другие.

Представление чисел в памяти ЭВМ. Существуют несколько основных способов представления чисел в памяти ЭВМ. Это форма с фиксированной точкой, форма с плавающей точкой и целые числа. Числа с фиксированной точкой, в отличие от чисел с плавающей точкой, имеют фиксированное количество бит, выделенных на целую (до точки) и дробную (после точки) части. Чаще всего используется формат в стандарте IEEE 754 [15]. Операции с плавающей точкой

выполняет специальный блок FPU (Floating Point Unit). Этот блок во всех моделях до i80486 выполнялся в виде отдельной микросхемы сопроцессора.

Мы же в данной лабораторной работе будем работать с целыми числами. Но целые числа могут быть как положительными, так и отрицательными. Для представления отрицательных чисел, в машинной арифметике используется дополнительный код. Двоичное шестнадцатиразрядное число можно представить как двоичное число со знаком, имеющее значение от -32768_{10} до $+32767_{10}$. Старший пятнадцатый разряд числа указывает знак.

Вычисление специальных функций. Для вычисления специальных функций ($\sin x$, $\cos x$, $\operatorname{tg} x$, $\ln x$, e^x, $\sqrt{x}$) применяются специальные алгоритмы. Функции $\sin x$, $\cos x$, $\ln x$ можно вычислить, воспользовавшись их разложением в ряд [16]:

$\sin x = x - x^3/3! + x^5/5! - x^7/7! + \ldots$ для любого x (x — в радианах),

$\cos x = 1 - x^2/2! + x^4/4! - x^6/6! + \ldots$ для любого x (x — в радианах),

$\ln (1+x) = x - x^2/2 + x^3/3 - x^4/4 + \ldots$ для всех $|x|<1$,

$e^x = 1 + x + x^2/2! + x^3/3! + x^4/4!$ для всех x.

Число членов ряда определяется из условия получения требуемой точности. Для вычисления функции $\sqrt{x}$ с точностью до целых чисел можно применить алгоритм, основанный на том, что квадрат числа можно определить сложением последовательности нечётных чисел:

1	1	=	1^2
2	1 + 3	=	2^2
3	1 + 3 + 5	=	3^2
4	1 + 3 + 5 + 7	=	4^2
5	1 + 3 + 5 + 7 + 9	=	5^2
6	1 + 3 + 5 + 7 + 9 + 11	=	6^2

Исходя из приведённого примера видно, что для получения квадрата числа N нужно последовательно сложить N нечётных чисел, начиная с 1.

Вычисление специальных функций по приведённым выражениям занимает длительное время и может не обеспечивать точность. Это обусловлено сравнительно небольшой длиной машинного слова и ограниченным быстродействием МП.

Поэтому в тех случаях, когда ставятся жёсткие требования по быстродействию и точности, возможно применение вычисления функций с помощью таблиц. Суть метода заключается в том, что где-то в памяти находится таблица с уже вычисленными значениями функции при определенных аргументах. МП лишь просматривает таблицу, и в соответствии с заданным аргументом, извлекает из неё определенное значение функции. При этом МП не нужно выполнять вычисление выражений по каким-то сложным алгоритмам, ему необходимо лишь выбирать из таблицы нужный элемент.

4.3.2 Задание к работе № 3

Во всех заданиях N — номер варианта.

4.3.2.1 Задание для домашней подготовки

1. Разработайте блок-схемы алгоритмов программ для пунктов 7 и 8 задания для выполнения в лаборатории.

2. Нарисуйте в заготовке отчёта по лабораторной работе блок-схемы алгоритмов программ по всем пунктам задания.

3. Найдите коды команд и запишите тексты всех программ, которые должны быть выполнены в лаборатории.

4.3.2.2 Задание для выполнения в лаборатории

1 Выполните операцию сложения двух чисел с получением результата двойной точности.

Первое число: 8F3Bh + N×35Fh.

Второе число: 816Ah.

2 Вычтите из числа, получившегося в предыдущей операции, число N×0F5Fh с получением результата двойной точности.

3 Выполните беззнаковое умножение двух чисел.

Первое число: $(N \times 11_{10})$.

Второе число: 120_{10}.

4 Выполните знаковое умножение двух чисел.

Первое число: — $(N \times 12_{10})$.

Второе число: — 128_{10}.

5 Выполните деление двух чисел.

Первое число: 1908_{10}.

Второе число: N_{10}.

6 Покажите, какие регистры используются для хранения результата операции.

7 Введите и отладьте программу для вычисления значения функции

$$f(x) = |(N - 5)| \times x^3 + N \times x^2 + |(N - 4)| \times x + N,$$

где x — аргумент функции (входной параметр, передаваемый программе), f — значение функции (результат вычислений программы).

8 Введите и отладьте программу для вычисления значения той же функции с помощью таблицы.

4.3.3 Контрольные вопросы

1 Дайте определение форме представления числа с плавающей точкой, с фиксированной точкой.

2 Дайте определение числа одинарной точности, двойной точности.

3 Чем отличаются прямой, обратный и дополнительный код? В каких целях используется та или иная форма представления числа?

4 Как формируется число в дополнительном коде?

5 Почему в микропроцессорах используется дополнительный код?

6 Какие арифметические операции может выполнять МП i80386?

7 Дайте сравнительный анализ методам вычисления функций: по специальным алгоритмам, по таблице.

8 В чем преимущество функций заданных в виде таблиц?

9 Разработайте алгоритм вычисления квадратного корня для чисел, принадлежащих ряду: 1, 4, 9, 16, 25, … (не используя табличный метод).

4.4 Лабораторная работа № 4. Работа с внешними устройствами

Цель работы: изучение принципов ввода и вывода информации, методов адресации к внешним устройствам, способов их подключения.

4.4.1 Теоретические сведения

Для работы с внешними устройствами в системе команд МП имеются специальные команды `in` и `out` (ввод-вывод с проекцией на ввод-вывод). При обращении к устройству ввода-вывода (при выполнении команд `in` и `out`) процессор выставляет на управляющую линию `M/#IO` сигнал низкого уровня, на адресную шину — адрес устройства ввода-вывода. Затем МП подаёт сигнал чтения или записи.

Схема выбора адреса. Схема выбора адреса представлена на рисунке 4.13. Схема позволяет назначить конкретный адрес устройствам ввода-вывода: регистру DIP-переключателя и регистру светодиодов. Одна из целей этой схемы — показать классический приём подключения внешних устройств. Обычно для выбора внешних устройств используется дешифратор. При подаче на его вход адреса внешнего устройства только на одном из выходов дешифратора, номер которого соответствует поданному на вход адресу, появится управляющий сигнал.

Для того чтобы была возможность задать устройству ввода-вывода 16 различных адресов, необходимо использовать дешифратор 1 из 16. При этом базовым адресом должен быть адрес 0B0h. Таким образом, диапазон адресов, к которым можно адресоваться с помощью дешифратора и перемычек, равен 0B0h — 0BFh.

В связи с особенностями стенда операцию ввода следует осуществлять только по чётным адресам!!!

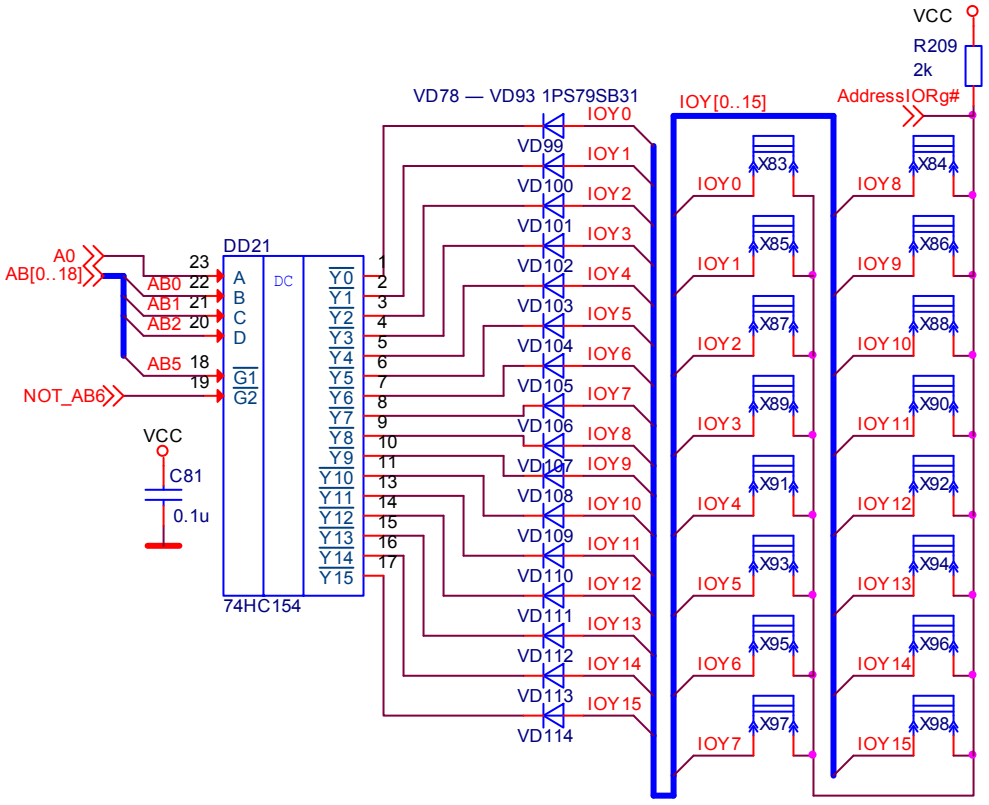


Рисунок 4.13 — Схема выбора адреса внешнего устройства

В результате получаем, что для регистра DIP-переключателя и регистра светодиодов зарезервировано 16 адресов. Причем регистрам можно назначать одинаковые адреса. Это не приведет к конфликту устройств на системной шине, потому что обращение к регистру DIP-переключателя происходит только по команде `IN`, а обращение к регистру светодиодов только по команде `OUT`.

При обмене данными с одним из этих устройств, по команде ввода-вывода, микропроцессор должен выставлять на адресную шину адрес этого устройства, назначенный с помощью схемы выбора адреса. Как только на выходе `AddressIORg#` появляется «0», то это значит, что выбрано либо устройство ввода — регистр DIP-переключателя (активируется при подаче сигнала RD#), либо выбрано устройство вывода — регистр светодиодов (активируется при подаче сигнала WR#).

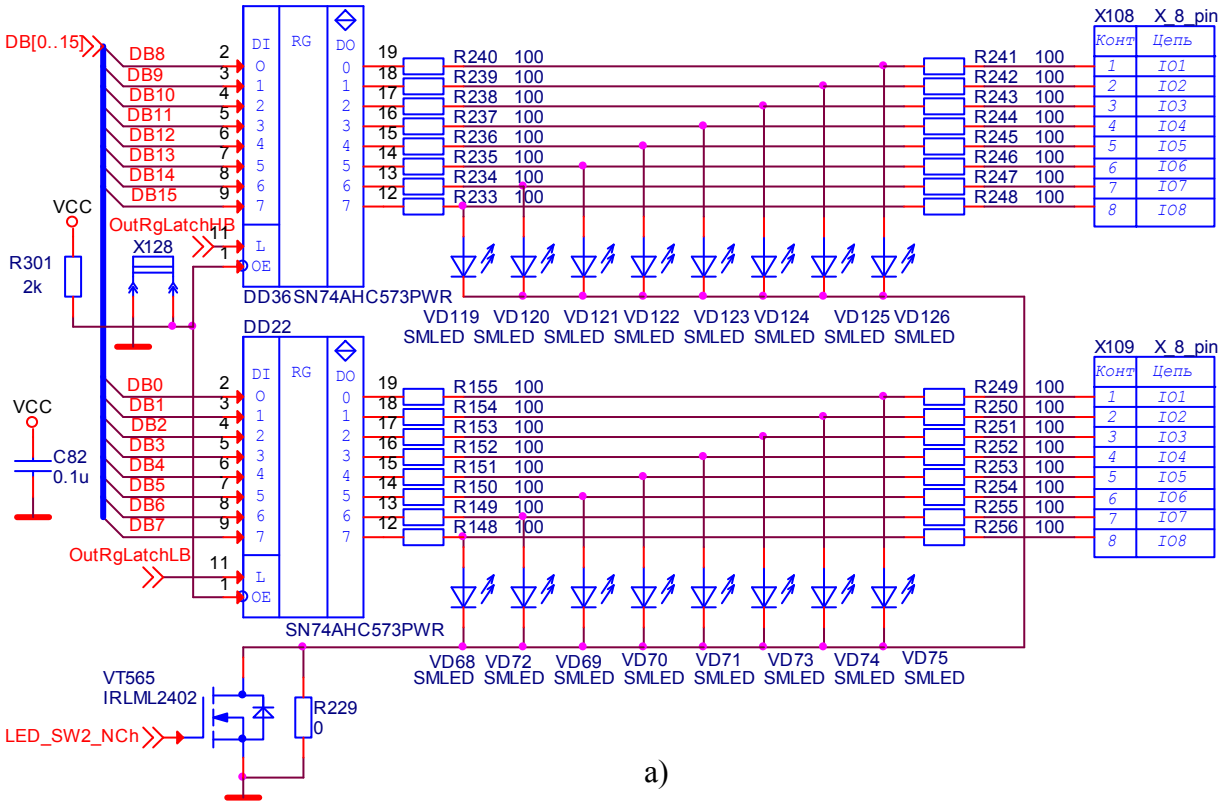
Схема подключения DIP-переключателя и светодиодов. На рисунке 4.14 представлена схема подключения регистра DIP-переключателя и регистра светодиодов.

При выполнении команды ввода с регистра DIP-переключателя, сигнал `RD#` микропроцессора вызовет появление сигнала активного уровня на линии `#InRgOE` регистра. Будет произведено чтение регистра DIP-переключателя. При выполнении команды вывода в регистр светодиодов, сигнал `#WR` микропроцессора вызовет появление сигнала активного уровня на линии `OutRgLatch` регистра. Будет произведена запись в регистр светодиодов. Для выполнения операции обмена данными с внешним устройством адрес устройства ввода-вывода, указанный в команде, должен соответствовать адресу устройства, заданному с помощью схемы выбора адреса.

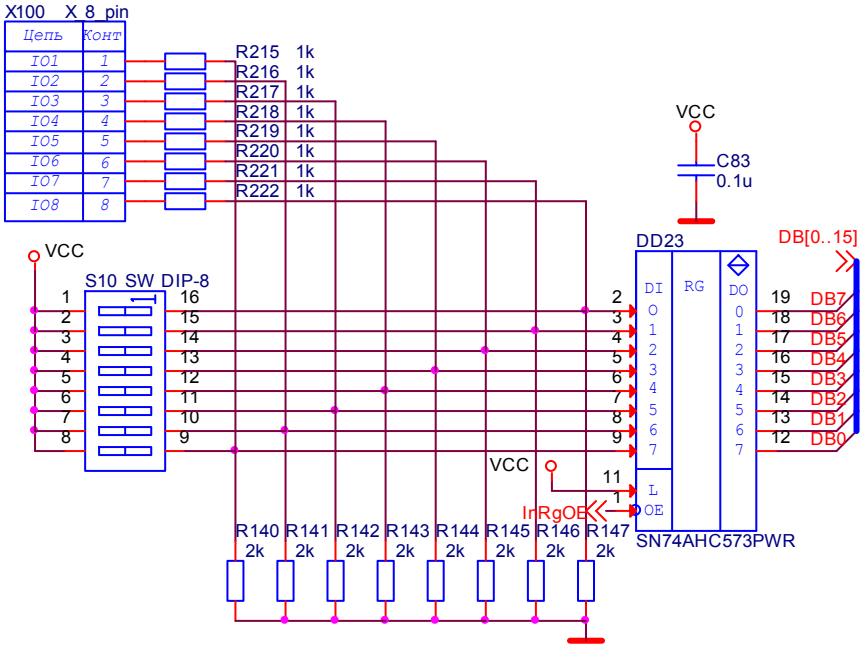
Схема подключения 7-сегментного индикатора. В качестве устройства вывода информации, удобного для восприятия, часто используется 7-сегментный индикатор. Рассмотрим подключение 7-сегментного индикатора. Внешний вид и схема 7-сегментной светодиодной матрицы представлена на рисунке 4.15.

В учебном лабораторном стенде используются 7-сегментные индикаторы в количестве восьми знакомест, имеющих 7 сегментов для отображения цифр и букв и 1 сегмент для отображения точки. Каждый сегмент знакоместа подсвечивается отдельным светодиодом. Индикаторы бывают разные, и светодиоды каждого знакоместа могут включаться либо по схеме с общим катодом либо по схеме с общим анодом. В учебном лабораторном стенде используется 7-сегментный индикатор, в котором светодиоды каждого знакоместа соединены по схеме с общим анодом. Соединение с общим анодом означает то, что все аноды светодиодов одного знакоместа индикатора соединены между собой. Соответственно чтобы светодиоды светились подключать анод 7-сегментного индикатора необходимо к напряжению питания (см. рисунок 4.15). Бывают индикаторы с общим катодом, в этом случае все катоды светодиодов одного знакоместа соединены между собой. При этом подключать катод 7-сегментного индикатора необходимо к «земле». Индикатор

может отображать цифры от 0 до 9, а также некоторые буквы. Буквенное обозначение сегментов представлено на рисунке 4.15.

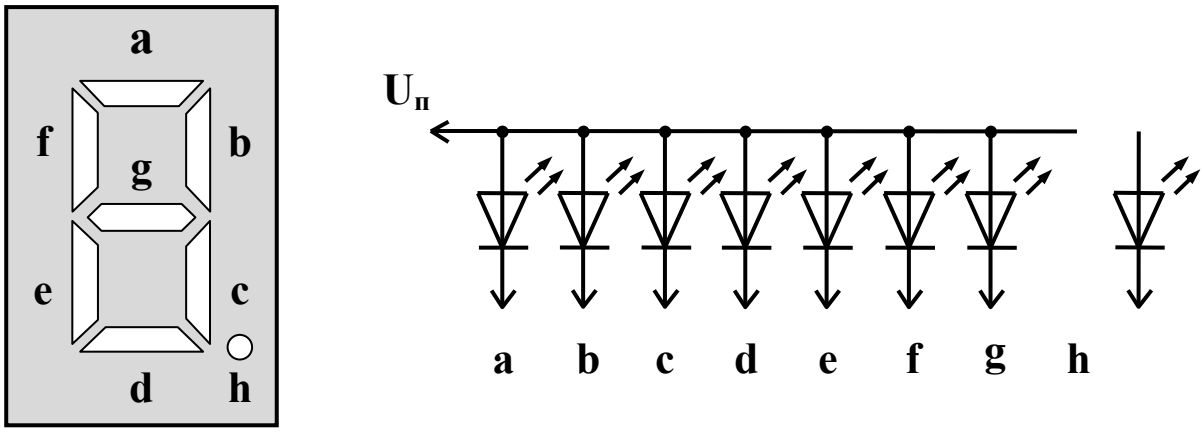


а)



б)

Рисунок 4.14 — Схема подключения регистров светодиодов а) и регистра DIP-переключателя б)



а *б*

Рисунок 4.15 — Внешний вид (*а*) и схема 7-сегментной светодиодной матрицы (*б*)

На рисунке 4.16 представлена схема подключения 7-сегментных индикаторов.

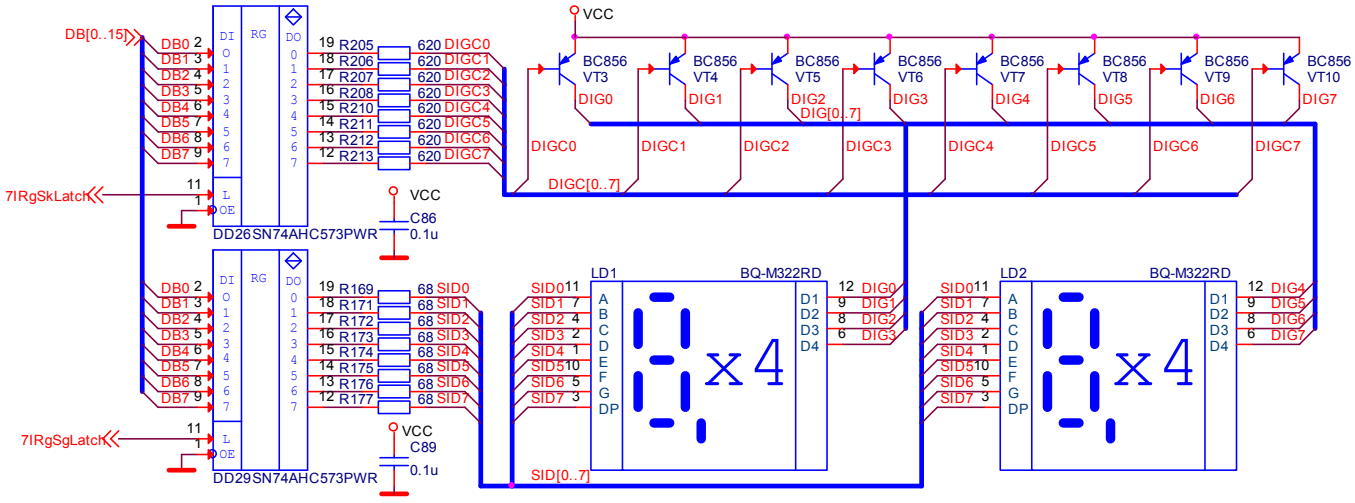


Рисунок 4.16 — Схема подключения 7-сегментных индикаторов

Регистры в схеме имеют фиксированные адреса. 200h — адрес регистра, выбирающего соответствующий индикатор (разряд) (DD28). 202h — адрес регистра, задающего 7-сегментный код символа (DD29). При выполнении команд вывода данных по адресам 200h и 202h, по сигналу МП `#WR`, будут формироваться активные уровни сигналов `7IRgSkLatch` и `7IRgSgLatch` соответственно.

Схема подключения клавиатуры. На рисунке 4.17 представлена схема подключения клавиатуры. Светодиоды, подключенные к выходу регистра DD10 необходимы для отладки программ работы с клавиатурой. Сканирование клавиатуры выполняется нулём. Это значит, что в регистр DD10 по каждому из 4-х первых его входов по очереди защёлкивается уровень логического нуля («0»). Этот «0» появляется на выходах регистра DD10. Для того чтобы выполнить сканирование первого ряда, состоящего из четырёх кнопок SA1, SA2, SA3, SA4, необходимо защёлкнуть в регистр DD10 число 0Eh (на первый вход DD10 подать «0») и прочитать состояние регистра D11. Эта операция называется операцией сканирования линии. Если линия в данный момент сканируется, то она находится под потенциалом логического нуля, и светодиод, подключенный к этой линии (соответствующему выводу DD10), не светится. Регистры клавиатуры имеют фиксированные адреса. 204h — адрес регистра, выдающего на системную шину данных результат сканирования клавиатуры (DD11). 206h — адрес регистра, получающего сканирующий сигнал от МП (DD10). При выполнении команд ввода-вывода по этим адресам, по сигналу `#RD` или `#WR`, будут формироваться активные уровни сигналов `KbInOE` или `KbOutLatCH` соответственно.

На рисунке 4.18 показано фактическое расположение внешних устройств на стенде. Нижнее положение перемычки на разъёмах для выбора адреса внешних устройств соответствует подаче уровня логического нуля на устройства ввода-вывода.

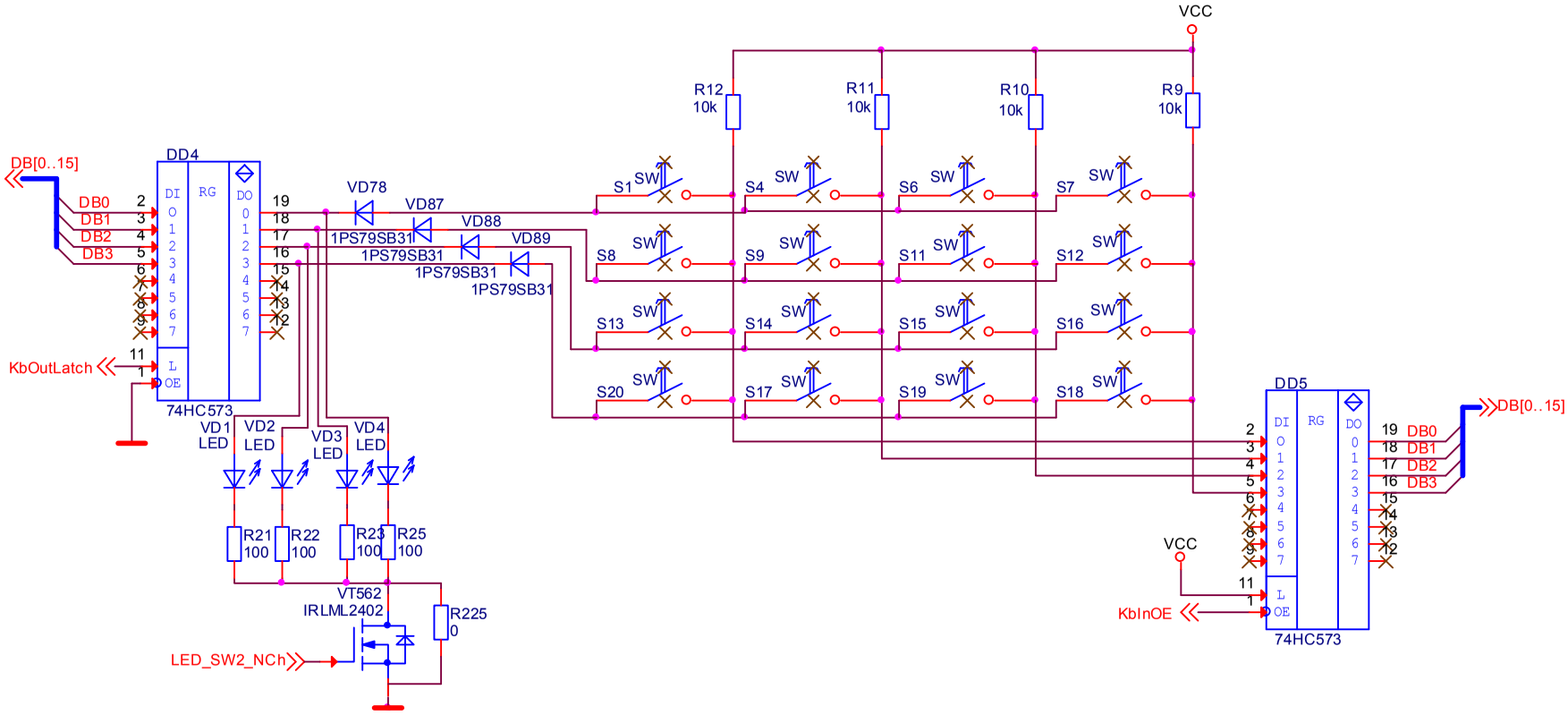


Рисунок 4.17 — Схема подключения клавиатуры

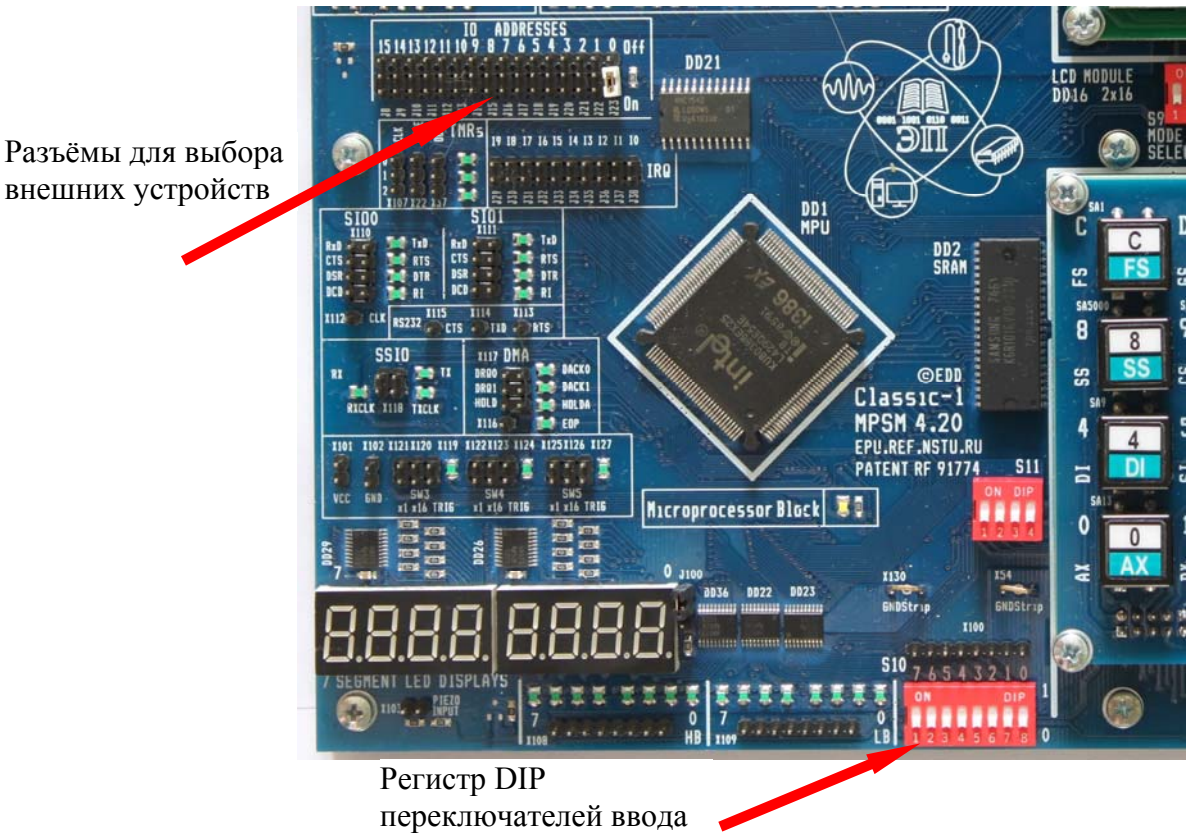


Рисунок 4.18 — Расположение периферийных устройств на стенде

4.4.2 Задание к работе № 4

Во всех заданиях N — номер вашего варианта.

4.4.2.1 Задание для домашней подготовки

1. Разработайте блок-схемы алгоритмов программ для всех пунктов задания.

2. Нарисуйте в заготовке отчёта по лабораторной работе блок-схемы алгоритмов программ по всем пунктам задания.

3. Найдите коды команд и запишите тексты всех программ, которые должны быть выполнены в лаборатории.

4.4.2.2 Задание для домашней подготовки

1 Выведите в младший разряд 7-сегментого индикатора номер вашего варианта, а в старший разряд номер варианта вашего соседа. Выполните разработанную программу по тактам. Проанализируйте выполнение команды вывода out.

2 Разработайте программу-цикл, которая читает состояние регистра DIP-переключателя, и выводит его в регистр светодиодов. С помощью схемы выбора адреса задайте регистру DIP-переключателя адрес 0FEh+2N, а регистру светодиодов адрес 100h+2N. Проверьте работу программы в автоматическом режиме.

3 Разработайте программу-счётчик, выводящую на 7-сегментный индикатор последовательно цифры от 0 до 99 в десятичной системе счисления.

4 Модифицируйте программу пункта 3 так, чтобы частота счёта задавалась с DIP-переключателя. Задайте адрес регистра DIP-переключателя 100h+2N.

5 Разработайте программу, которая осуществляет сканирование цифровой клавиатуры и вывод на 7-сегментный индикатор номера нажатой клавиши. Распределение номеров для клавиш по вариантам представлено на рисунке 4.18.

4.4.3 Контрольные вопросы

1 В чём отличие ввода-вывода с проекцией на память от ввода-вывода с проекцией на ввод-вывод?

2 Может ли МП 80386 осуществлять ввод-вывод с проекцией на память?

3 Зачем микропроцессору управляющая линия `M/#IO`?

4 Какие способы адресации используются в командах ввода-вывода МП i80386?

5 Объясните работу схемы подключения клавиатуры.

0	**4**	**8**	**C**
1	**5**	**9**	**D**
2	**6**	**A**	**E**
3	**7**	**B**	**F**

N = 1, 5, 9, D

3	**7**	**B**	**F**
2	**6**	**A**	**E**
1	**5**	**9**	**D**
0	**4**	**8**	**C**

N = 2, 6, A, E

3	**2**	**1**	**0**
7	**6**	**5**	**4**
B	**A**	**9**	**8**
F	**E**	**D**	**C**

N = 3, 7, B, F

0	**1**	**2**	**3**
4	**5**	**6**	**7**
8	**9**	**A**	**B**
C	**D**	**E**	**F**

N = 4, 8, C, 10h

Рисунок 4.18 — Распределение номеров для клавиш.

6 Объясните работу схемы выбора адреса внешнего устройства.

7 Объясните назначение всех резисторов в схеме подключения клавиатуры.

8 Как оптимально модернизировать схему подключения 7-сегментных индикаторов, так чтобы можно было подключить еще 30 индикаторов?

9 Каков максимальный размер пространства адресов устройств ввода-вывода, к которому может обратиться микропроцессор?

10 К какому пространству адресов устройств ввода-вывода может обратиться МП с использованием прямого метода адресации?

11 В какое максимальное пространство физических адресов может быть выполнен ввод-вывод, если ввод-вывод осуществляется с проекцией на память?

Список литературы

1 Intel. Номера процессоров Intel. — [Web-документ], н. д. / 2008. — URL: [http://www.intel.com/products/processor_number/rus/index.htm] (12.11.2008).

2 Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture November 2006 Order Number: 253668-022US.

3 Intel386 EX Embedded Microprocessor User's Manual. — [Электронный документ], 1996 Order Number: 272485-002. — На диске: \Microprocessor\Docs\27248502.pdf].

4 Брамм П., Брамм Д. Микропроцессор 80386 и его программирование: пер. с англ. — Москва: Мир, 1990. — 448 с.

5 Википедия. Intel 8086. — [Web-документ], 10 августа 2009. — URL: [http://ru.wikipedia.org/wiki/8086].

6 Паппас К., Марри У. Микропроцессор 80386. Справочник: пер. с англ. / под ред. В. В. Василькова. — Москва: Радио и связь, 1993. — 318 с.

7 Рафикузаман М. Микропроцессоры и машинное проектирование микропроцессорных систем: пер. с англ. В 2 кн. Кн. 2. — М.: Мир, 1988. — 288 с.

8 Смит Б. Э., Джонсон М. Т. Архитектура и программирование микропроцессора Intel 80386: пер. с англ./ Под ред.А. С. Карнаухова. — Москва: Конкорд, 1992. — 334 с.

9 Л. Дао. Программирование микропроцессора 8088: пер. с англ./ Под ред. М. М. Гельмана. — Москва: Мир, 1988. — 357 с.

10 Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, http://www.intel.com/Assets/PDF/manual/253666.pdf (05.12.2010).

11 Лю Ю-Чжен, Гибсон Г. Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем. Пер. с англ. — М.: Радио и связь, 1987. — 512.; илл.

12 Микропроцессорный комплект К1810: Структура, программирование, применение: Справочная книга/ Ю.М. Казаринов, В.Н. Номоконов, Г.С.

Подклетнов, В.Ф. Филиппов; Под ред. Ю.М. Казаринова. – М.: Высш. Шк., 1990. – 269 с.

13 К. Касперски. Cod:net. Техника и философия хакерских атак. — [web-документ]. — URL: [http://www.codenet.ru/progr/other/hack_solon8.php] (2.10.2008).

14 Cod:net. Справочник по системе программирования Турбо Ассемблер. — [web-документ]. — URL: [http://www.codenet.ru/progr/asm/tasm/41.php] (28.10.2008).

15 754-2008 IEEE Standard for Floating-Point Arithmetic. — [web-документ]. — URL: [http http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?reload=true&arnumber=4610935] (25.04.2010).

16 Микропроцессоры. Кн. 3/ Под ред. Л. Н. Преснухина. — Минск: Высшая школа, 1987. — 350 с.

Приложение А
(справочное)
Система команд i80386

код	x0	x1	x2	x3	x4	x5	x6	x7
0x	ADD r/m,r8	ADD r/m,r16	ADD r8,r/m	ADD r16,r/m	ADD AL,im8	ADD AX,im16	PUSH ES	POP ES
1x	ADC r/m,r8	ADC r/m,r16	ADC r8,r/m	ADC r16,r/m	ADC AL,im8	ADC AX,im16	PUSH SS	POP SS
2x	AND r/m,r8	AND r/m,r16	AND r8,r/m	AND r16,r/m	AND AL,im8	AND AX,im16	SEG ES	DAA
3x	XOR r/m,r8	XOR r/m,r16	XOR r8,r/m	XOR r16,r/m	XOR AL,im8	XOR AX,im16	SEG SS	AAA
4x	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5x	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6x	PUSHA	POPA	BOUND	ARPL	SEG FS	SEG GS	opSize prefix	addrSize prefix
7x	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA
8x	ArOp1 r/m,im8	ArOp1 r/m,im16	ArOp2 r/m8,im8	ArOp2 r/m16,im 8	TEST r/m,r8	TEST r/m,r16	XCHG r8,r/m	XCHG r16,r/m
9x	NOP	XCHG AX,CX	XCHG AX,DX	XCHG AX,BX	XCHG AX,SP	XCHG AX,BP	XCHG AX,SI	XCHG AX,DI
Ax	MOV AL,mem8	MOV AX,mem16	MOV mem8,AL	MOV mem16,AX	MOVSB	MOVSW	CMPSB	CMPSW
Bx	MOV AL,im8	MOV CL,im8	MOV DL,im8	MOV BL,im8	MOV AH,im8	MOV CH,im8	MOV DH,im8	MOV BH,im8
Cx	ShfOp r/m8,im	ShfOp r/m16,im	RET near im16	RET near	LES r16,mem	LDS r16,mem	MOV mem,im8	MOV mem,im16
Dx	ShfOp r/m8,1	ShfOp r/m16,1	ShfOp r/m8,CL	ShfOp r/m16,CL	AAM	AAD		XLAT
Ex	LOOPNE/ LOOPNZ	LOOPE/ LOOPZ	LOOP	JCXZ JECXZ	IN AL,port8	IN AX,port8	OUT AL,port8	OUT AX,port8
Fx	LOCK		REP/ REPNE	REPZ/ REPE	HALT	CMC	Grp1 r/m8	Grp1 r/m16
код	**x0**	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**

код	x8	x9	xA	xB	xC	xD	xE	xF
0x	OR r/m,r8	OR r/m,r16	OR r8,r/m	OR r16,r/m	OR AL,im8	OR AX,im16	PUSH CS	Extnsn OpCode
1x	SBB r/m,r8	SBB r/m,r16	SBB r8,r/m	SBB r16,r/m	SBB AL,im8	SBB AX,im16	PUSH DS	POP DS
2x	SUB r/m,r8	SUB r/m,r16	SUB r8,r/m	SUB r16,r/m	SUB AL,im8	SUB AX,im16	SEG CS	DAS
3x	CMP r/m,r8	CMP r/m,r16	CMP r8,r/m	CMP r16,r/m	CMP AL,im8	CMP AX,im16	SEG DS	AAS
4x	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5x	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6x	PUSH im16	IMUL r/m,im16	PUSH im8	IMUL r/m,im8	INSB	INSW	OUTSB	OUTSW
7x	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNG	JNL/ JGE	JLE/ JNG	JNLE/ JG
8x	MOV r/m,r8	MOV r/m,r16	MOV r8,r/m	MOV r16,r/m	MOV r/m,seg	LEA r16,mem	MOV seg,r/m	POP r/m
9x	CBW	CWD	CALL far	WAIT	PUSHF	POPF	SAHF	LAHF
Ax	TEST AL,mem8	TEST AX,mem16	STOSB	STOSW	LODSB	LODSW	SCASB	SCASW
Bx	MOV AX,im16	MOV CX,im16	MOV DX,im16	MOV BX,im16	MOV SP,im16	MOV BP,im16	MOV SI,im16	MOV DI,im16
Cx	ENTER im16,im8	LEAVE	RET far im16	RET far	INT3	INT im8	INTO	IRET
Dx	ESC 0 387/486	ESC 1 387/486	ESC 2 387/486	ESC 3 387/486	ESC 4 387/486	ESC 5 387/486	ESC 6 387/486	ESC 7 387/486
Ex	CALL *near*	JMP near	JMP far	JMP short	IN AL,DX	IN AX,DX	OUT AL,DX	OUT AX,DX
Fx	CLC	STC	CLI	STI	CLD	STD	Grp2 r/m8	Grp3 r/m16
код	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**

Команды, содержащие код операции в битах 3-5 второго байта, в которых обычно указывается режим адресации:

	md 000 rm	**md 001 rm**	**md 010 rm**	**md 011 rm**	**md 100 rm**	**md 101 rm**	**md 110 rm**	**md 111 rm**
ArOp1	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
ArOp2	ADD		ADC	SBB		SUB		CMP
ShftOp	ROL	ROR	RCL	RCR	SHL/SAL	SHR		RAR
Grp1	TEST		NOT	NEG	MUL	IMUL	DIV	IDIV
Grp2	INC	DEC	CALL near	CALL far	JMPnr	JMP far	PUSH	
Grp3	INC	DEC						

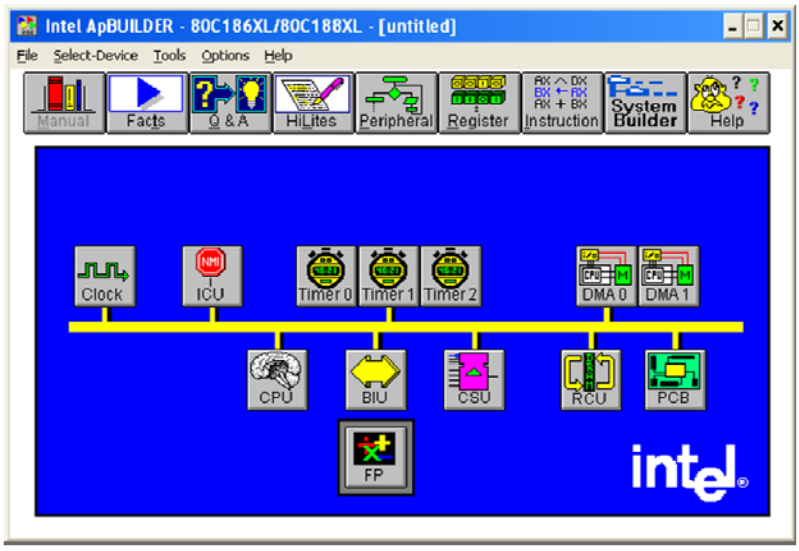
Приложение Б
(справочное)
Описание работы с программой ApBuilder

Свободно распространяемая программа ApBuilder разработана фирмой Intel и предназначена для облегчения разработки программного обеспечения для электронных приборов фирмы Intel, которые позиционируются фирмой как микроконтроллеры. Программа ApBuilder является одной из первых в своём классе программ, позволяющих создавать программное обеспечение для микроконтроллеров с использованием методов визуального программирования. В настоящее время такие программы обычно называют конфигураторами.

Вид рабочего окна программы показан на рисунке Б.1, *а*. Окно можно разделить на три зоны. Вверху располагается панель инструментов, ниже расположена панель быстрого доступа и основное место занимает символическое представление архитектуры выбранного микроконтроллера. Как следует из названия окна, на этом рисунке представлена архитектура микроконтроллера i80C186XL/80C188XL.

Кратко рассмотрим назначение кнопок на панели инструментов. Кнопка File (Файл) обеспечивает доступ к стандартным функциям работы с файлами. Кнопка Select-Device (Выбор устройства) позволяет выбрать микроконтроллер, с которым предстоит работа. С помощью кнопки Tools (Инструменты) можно выбрать один из инструментов, иконки которых приведены на панели быстрого доступа, и о которых будет сказано позднее. Настройка программы ApBuilder производится с помощью кнопки Options (Опции). Набор возможностей, предоставляемых программой, показан на рисунке Б.1, *б*. Таким образом, пользователь может выбрать шрифт (Select Font…), указать предпочитаемый текстовый редактор (Configure Editor…), добавить в панель инструментов необходимые вам программы (Configure User Tools…) и указать полный путь к руководствам пользователя и справочным листкам для данного микроконтроллера (Configure Manual and Data Sheets…). И, наконец, кнопка Help (Помощь) позволяет пользователю получить справочную информацию (естественно, на английском языке) по использованию программы ApBuilder. К

а



б

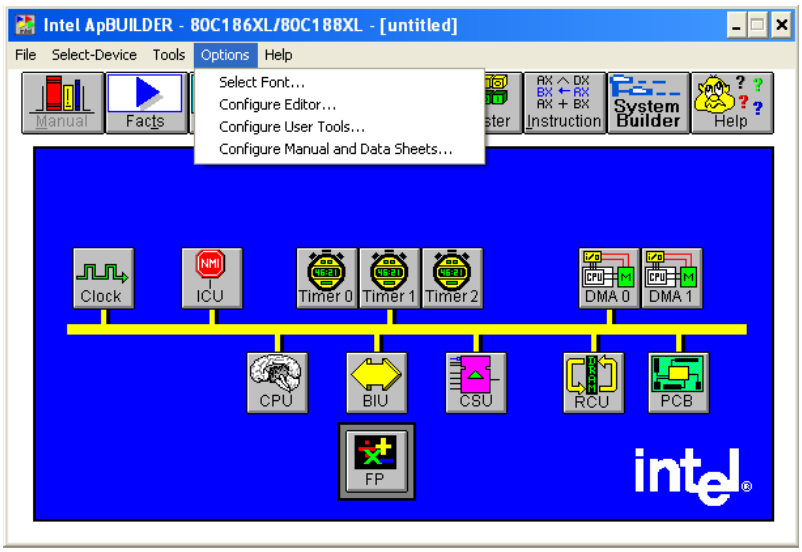


Рисунок Б.1 — Вид рабочего окна: *а* — после инсталляции; *б* — при выборе инструмента Options

сожалению, в своё время фирма Intel отказалась от поддержки развития микроконтроллеров в пользу микропроцессоров общего назначения, поэтому новых версий ApBuilder не существует. Мы используем версию 2.21 от 1997 года. Интересно, что используемая версия программы была создана до широкого использования сети Internet, и из справочной информации вы можете узнать, что обновление программы, в первую очередь, возможно через BBS (а не через Internet, как это сейчас принято). Можно быть уверенным, что никто из студентов, читающих эти методические указания, уже не знает, что такое BBS! (Любознательные могут получить ответ с помощью ABBYY Lingvo x3.)

Для работы с конкретным микроконтроллером следует с помощью кнопки Select-Device на панели инструментов выбрать интересующий нас тип микроконтроллера — Intel386(TM)EX. Эти действия иллюстрируются рисунком Б.2.

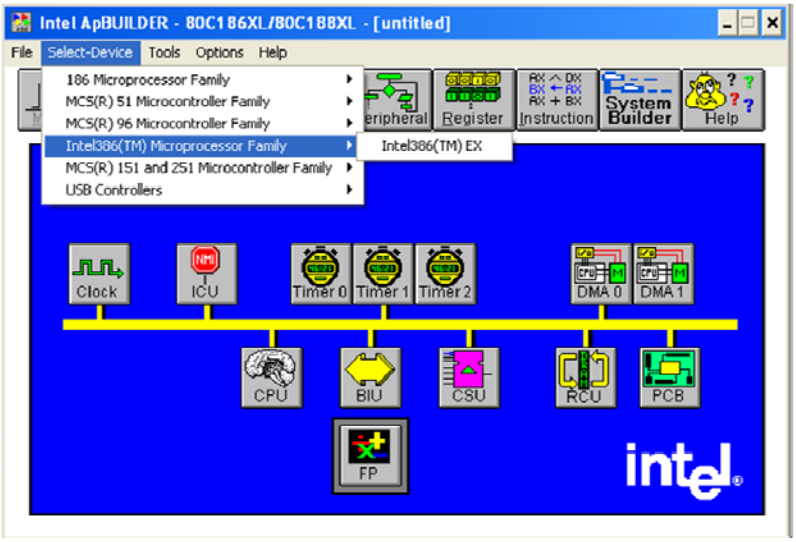


Рисунок Б.2 — Выбор микроконтроллеров семейства Intel386(TM)

В результате мы получим рабочее окно, соответствующее микросхемам семейства Intel386(TM)EX (рисунок Б.3). Как следует из этого рисунка, в микросхеме имеется большое количество различных модулей, и она является достаточно развитым микроконтроллером. При выполнении данного цикла

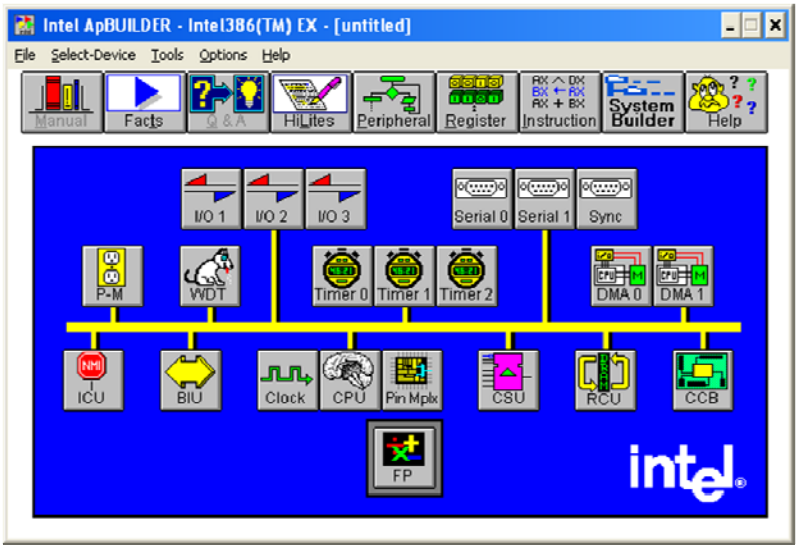


Рисунок Б.3 — Вид рабочего окна ApBuilder для микроконтроллеров семейства Intel386(TM)

лабораторных работ, посвящённого изучению основ микропроцессорной техники, мы будем использовать только модуль центрального процессора (CPU), а остальные модули будут изучаться при выполнении лабораторных работ по дисциплине «Микропроцессорные системы и персональные компьютеры».

Рассмотрим назначение кнопок, расположенных на панели быстрого доступа. Самая левая кнопка Manual (Руководство). По умолчанию эта кнопка не активна (подпись выполнена серым, а не чёрным, цветом). Для её активации нужно с помощью кнопки панели инструментов Options → Configure Manual and Data Sheets указать полный путь к документу, в котором находится описание выбранного микроконтроллера (например, файл 27248502.pdf «Intel386™ EX Embedded Microprocessor User's Manual», расположенный на диске в папке Microprocessor\Docs.

Следующая кнопка — Facts. Она позволяет получить краткую справку о продукции фирмы Intel и о возможностях поддержки и обслуживания этой продукции. Facts имеет три уровня: Краткая информация, Сравнение продуктов и Сервис, средства поддержки и сведения о получении справочной литературы.

Название третьей кнопки Q&A является аббревиатурой слов Questions and Answers (Вопросы и ответы), и на самом деле эта кнопка дает возможность увидеть наиболее часто задаваемые вопросы и правильные ответы на них. Для микропроцессоров семейства Intel386(TM)EX этот инструмент фирма Intel так и не успела подготовить.

Кнопка HiLites открывает доступ к сжатой информации о выбранном блоке микроконтроллера. Она действует после выбора конкретного блока с помощью щелчка левой клавиши мыши и последующего нажатия левой клавиши мыши на иконке HiLites.

Как видно, перечисленные инструменты носят справочно-информационный характер. Следующие три кнопки являются основными средствами автоматизации разработки программ. Обратите внимание, что во всех диалоговых окнах, соответствующих этим инструментам, есть кнопки Manual, предоставляющие

контекстно-чувствительную справочную информацию (если вы уже произвели конфигурацию этой опции через Options → Configure Manual and Data Sheets).

Peripheral (Периферия) — позволяет установить нужную конфигурацию периферийного устройства, просмотреть код, определяющий эту конфигурацию (Showcode) и скопировать код в буфер обмена Windows с помощью кнопки Copy.

Register (Регистр) — выполняет те же функции для всех регистров — как для регистров общего назначения, так и для SFR.

Кнопка Instruction (Инструкция, команда) обеспечивает получение исчерпывающей информации о командах выбранного устройства (особенно с учётом Manual), вплоть до времени выполнения команды при выбранной частоте тактового генератора. Рассмотрим подробнее работу с этой кнопкой.

Окно Instruction Editor (Редактор команд), появляющееся при нажатии кнопки Instruction и выборе команды Add из выпадающего списка команд Instruction, показано на рисунке Б.4. Левее выпадающего списка Instruction расположен выпадающий список Format (Формат), с помощью которого можно увидеть все

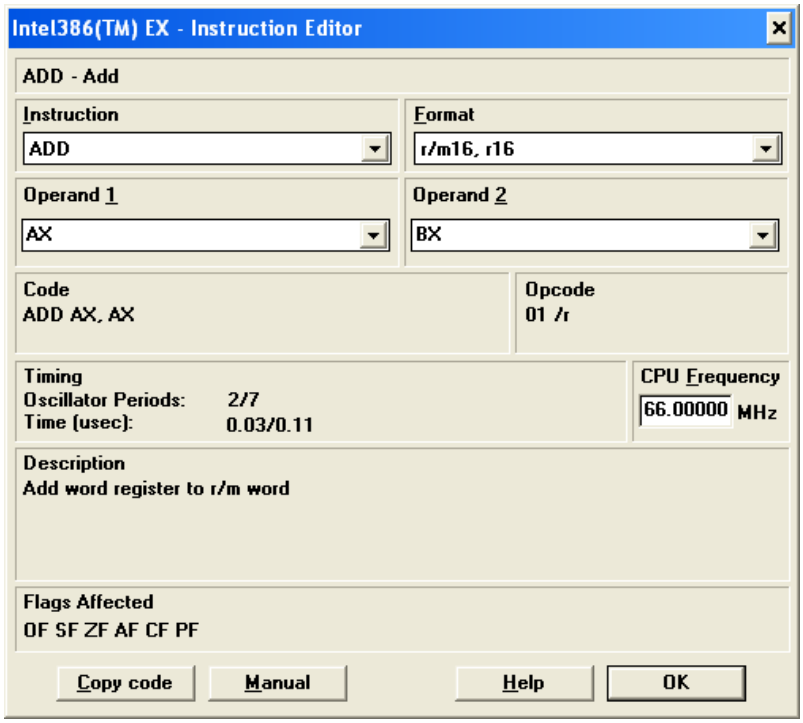


Рисунок Б.4 — Вид окна редактора команд для микроконтроллеров семейства Intel386(TM)

форматы для выбранной команды. Для команды Add это — набор возможных операндов, в выпадающем списке будет представлено 14 вариантов. На рисунке Б.4 показано окно при выборе варианта r/m16, r16 (формат r/m рассматривается в данной методичке). В выбранном формате в качестве операнда-источника и операнда-приёмника также возможны различные варианты, поэтому появляются выпадающие списки для операнда-приёмника (Operand 1) и операнда-источника (Operand 2).

Ниже располагается поле, в котором приведена мнемоника типа команды (`ADD AX, AX`), правее которого находится код операции для данного типа команд (01 /r).

Ещё ниже приведены данные о времени выполнения команд (Timing) при частоте, выбираемой из выпадающего списка CPU Frequency (Частота ЦПУ). Время выполнения для данного типа команд указывается как в виде количества тактов, таки и мкс, через дробь, где в числителе указывается минимальное значение, а в знаменателе — максимальное значение для данного типа команд.

Далее располагается краткое описание команды, ещё ниже — список флагов, на которые влияет выполняемая команда.

С помощью этой кнопки можно получить достаточно подробную информацию о формате команд i80386EX. Для этого в окне Instruction Editor следует нажать кнопку Manual. В появившемся окне Intel386TM Microprocessor Instruction Set представлена возможность получить справку об общем формате команд (Instruction Format), о формате постбайта ModR/M и о формате байте SIB. Также там доступна справка по командам, разделённым на 18 групп по алфавитному признаку названий команд.

Для составления программы нужно выбирать требуемую команду, затем копировать её в буфер Windows, нажимая кнопку Copy code (Копировать код), а затем вставлять в программу, которая записывается в выбранном вами текстовом редакторе.

Возвратимся к рассмотрению кнопок инструментов на панели быстрого доступа. Следующая кнопка System Builder (Построение системы) предназначена

для программирования микроконтроллера как системы. Эта возможность реализована только для микроконтроллеров семейства i80151 и i80251.

Наконец, последняя кнопка, Help, даёт возможность получить справочную информацию по работе с программой ApBuilder.

Приложение В
(справочное)
Описание работы с программой TASM5

Для получения двоичных кодов своих программ будем пользоваться транслятором TASM5. Этот транслятор является компилятором и общий принцип его работы следующий.

Пишем программу на ассемблере в текстовом редакторе, сохраняем в файл с расширением asm. Компилируем этот файл с помощью транслятора TASM5. В результате транслятор генерирует листинг с расширением lst и объектный файл с расширением obj. Двоичные коды следует брать из листинга. В листинге двоичные коды каждой команды группируются в одну строчку, отображается адрес команды в памяти, комментарии.

Теперь всё по порядку. Первым делом устанавливаем на персональный компьютер компилятор TASM5 из дистрибутива, находящегося на прилагаемом диске. [8]. Далее на доступном логическом диске создаем свою рабочую директорию, например Drive:\TASM5_DATA\100320IVN_lab1. В неё мы будем сохранять все свои файлы проекта. Затем из папки Drive:\TASM\BIN\ установленной программы в свою рабочую директорию копируем файл TASM.exe (где Drive — имя диска на который была произведена установка транслятора).

Открываем текстовый редактор, например Блокнот (или специализированное приложение, например, Eclipse), и пишем программу на ассемблере. Сохраняем текст программы в свою рабочую директорию в файл с расширением asm. Например, сохраним файл как IVN_lab1.asm. Переходим к компиляции своей программы. Запускаем командную строку Windows, или какой-либо консольный файловый менеджер (например, FAR Manager, Volkov Commander, DN/2, которые находятся на прилагаемом диске). Заходим в свою рабочую директорию, для этого набираем в командной строке команду cd Drive:\TASM5_DATA\100320IVN_lab1 и нажимаем Enter. Если пользуемся консольным файловым менеджером, то войти в свою директорию можно также с помощью клавиш навигации по директории или мышки.

Для получения листинга программы необходимо выполнить компиляцию с ключом /la. Для выполнения компиляции в командной строке вводим следующую команду и нажимаем Enter.

```
tasm /la IVN_lab1.asm
```

В команде указываем компилятор (tasm), вводим ключ (/la), указываем файл на языке ассемблера который нужно скомпилировать. Если компиляция прошла успешно, и ошибок нет, то транслятором будет сгенерирован листинг, который уже можно использовать для просмотра, анализа и извлечения двоичных кодов своей программы. В консольном файловом менеджере FAR Manager для просмотра результата компиляции можно воспользоваться комбинацией клавиш Ctrl+O.

Для того чтобы каждый раз вручную не вводить команду компиляции обязательно при выполнении лабораторных работ можно написать пакетный файл (bat-файл), содержащий команду вызова компилятора с необходимыми параметрами. В случае использования bat-файла нет необходимости копировать файл TASM.exe в рабочую директорию, следует лишь указать в пакетном файле путь к exe-файлу (Drive:\TASM\BIN\).

Ниже приведён пример пакетного файла, для случая когда транслятор установлен на диск C: и рабочая директория имеет путь C:\data_tasm.

```
@goto start
-----------------------------------------------------------------
Этот пакетный файл предназначен для автоматизации рутинных операций,
выполняемых студентами на лабораторных работах по основам МП техники
Пакетный файл написан 25/09/2009
-----------------------------------------------------------------
:start
@cd c:\data_tasm
@echo  assembling….
@pause
c:\tasm\bin\tasm /la IVN_lab1.asm, ,
```

Для того чтобы создать пакетный файл открываем текстовый редактор (блокнот), вводим команды, сохраняем файл с расширением bat в свою рабочую директорию.

Для проверки работоспособности в Turbo Debugger фирмы Borland можно после ассемблирования произвести линковку, добавив в созданный *.bat файл такую строчку:

```
c:\tasm\bin\tlink /t IVN_lab1.obj, ,
```

В результате (если не было ошибок) должен появиться файл IVN_lab1.com.

Далее скопируйте в рабочую директорию файлы TD.EXE и RTM.EXE из папки C:\tasm\bin.

Теперь для отладки удобно создать ещё один *.bat файл, содержащий строчку: `td.exe IVN_lab1.com.`

После запуска вновь созданного *.bat файла вы окажетесь в рабочем окне отладчика Borland и сможете проверить выполнение команд, содержащихся в файле IVN_lab1.asm.

В конце лабораторной работы сохраняем финальные или промежуточные версии написанных программ в своем разделе на сервере.

После окончания работы файлы TD.EXE и RTM.EXE можно удалить из рабочей директории.