**РОСЖЕЛДОР**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Ростовский государственный университет путей сообщения»**
**(ФГБОУ ВО РГУПС)**

---

А.В. Чернов, А.В. Тишина

# АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ

Учебно-методическое пособие
для выполнения лабораторных и практических работ

Часть 1

Микропроцессоры INTEL 80X86

Ростов-на-Дону
2017

УДК 681.31(75.8)

**Чернов А.В.**

Архитектура информационных систем: [Электронный ресурс] учебно-методическое пособие. В 3 ч. Ч. 1. Микропроцессоры INTEL 80X86 /А.В. Чернов, А.В. Тишина; ФГБОУ ВО РГУПС. – Ростов н/Д, 2017. – 50 с.

Учебно-методическое пособие предназначено для выполнения лабораторных и практических работ. Приводятся теоретические и методические рекомендации к выполнению работ по изучаемым темам. Рассмотрены принципы программирования и базовые приемы и на примере микропроцессоров семейства Intel 80x86. Приводится методика выполнения работы, контрольный пример, варианты заданий.

Предназначено для студентов направления подготовки 09.03.02 «Информационные системы и технологии» и 09.03.01 «Информатика и вычислительная техника» при изучении дисциплин «Архитектура информационных систем», «Архитектура вычислительных систем» и «Архитектура вычислительных и автоматизированных систем».

Одобрено к внесению в «Электронный университет» кафедрой «Вычислительная техника и автоматизированные системы управления».

**ВВЕДЕНИЕ**

Современный специалист в области создания программного обеспечения для вычислительной техники и автоматизированных систем должен обладать достаточными знаниями по использованию средств вычислительной техники в организации и управлении процессами разработки программного обеспечения. Низкоуровневое программирование позволяет четко усвоить принципы работы вычислительных машин и систем, а также их функциональных блоков, более рационально использовать их вычислительную мощность при разработке конкретного вида программного обеспечения, с учётом его особенностей.

**Целью** проведения лабораторных и практических работ является изучение студентами организации и принципов функционирования памяти, микропроцессора, организации ввода – вывода, а также приобретение навыков низкоуровневого программирования на языке ассемблера.

Для выполнения работ требуются IBM совместимый персональный компьютер, операционная система Windows XP и выше, программный пакет ассемблера TASM.

# Лабораторная работа 1

# ОРГАНИЗАЦИЯ МИКРОСИСТЕМ НА БАЗЕ МИКРОПРОЦЕССОРОВ I8086

## 1.1. Цель работы

Знакомство с принципами организации микросистем на базе МП i8086/80286 (далее МП86). Изучение архитектуры и программирования в машинных кодах МП86. Отработка навыков работы с турбоотладчиком TD.

## 1.2. Принципы организации микросистем на базе МП i8086 (К1810ВМ86)

На рис. 1.1 приведена типовая структура микропроцессорных систем и микрокомпьютеров на базе 16-битного микропроцессора i8086 или К1810ВМ86. Микросистема содержит центральный процессор на основе МП i8086, память, подсистему ввода-вывода и логику управления системной шиной, которая преобразует центральную магистраль Адрес/Данные (A/D) МП в раздельные шины адреса и данных. Такую структуру имеют 16-разрядные персональные компьютеры типа IBM PC/XT, Искра 1030, СМ1910, Mazovia CM1914 и др.

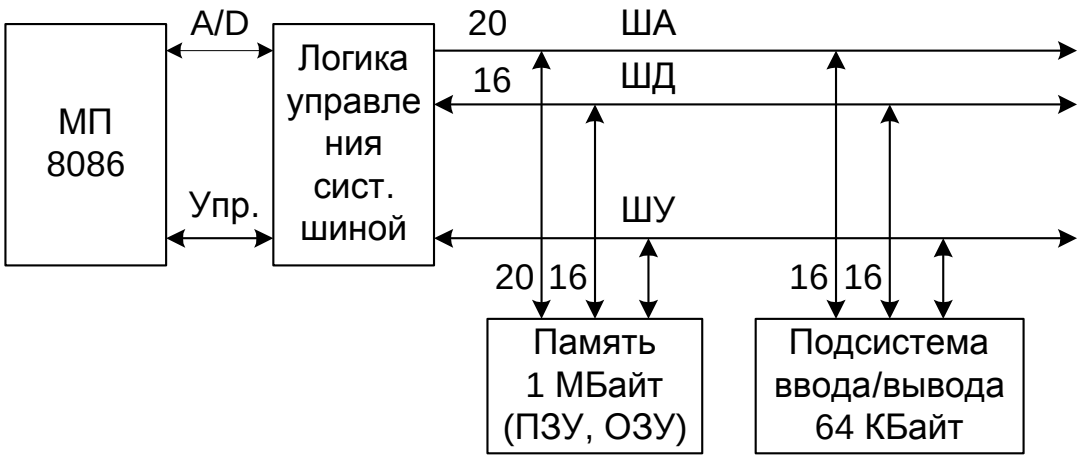


Рис. 1.1. Структура микросистемы на базе МП i8086

### 1.3. Структурная схема микропроцессора i8086 (рис. 1.2)

В МП i8086 основные этапы выполнения команды распределены внутри МП между двух сравнительно независимых устройств (рис. 1.2): между операционным (ОУ) и устройством сопряжения (УС).

ОУ содержит 16-битные регистры данных AX, BX, CX, DX, указатели памяти SP, BP, SI, DI, арифметико-логическое устройство АЛУ и регистр признаков F. Когда ОУ занято выполнением текущей команды, устройство сопряжения УС осуществляет опережающую выборку из памяти очередных команд. Команды хранятся во внутренней регистровой памяти, называемой очередью (буфером) команд. Очередь команд по существу выполняет функцию регистра команд процессора. Длина очереди составляет 6 байт.

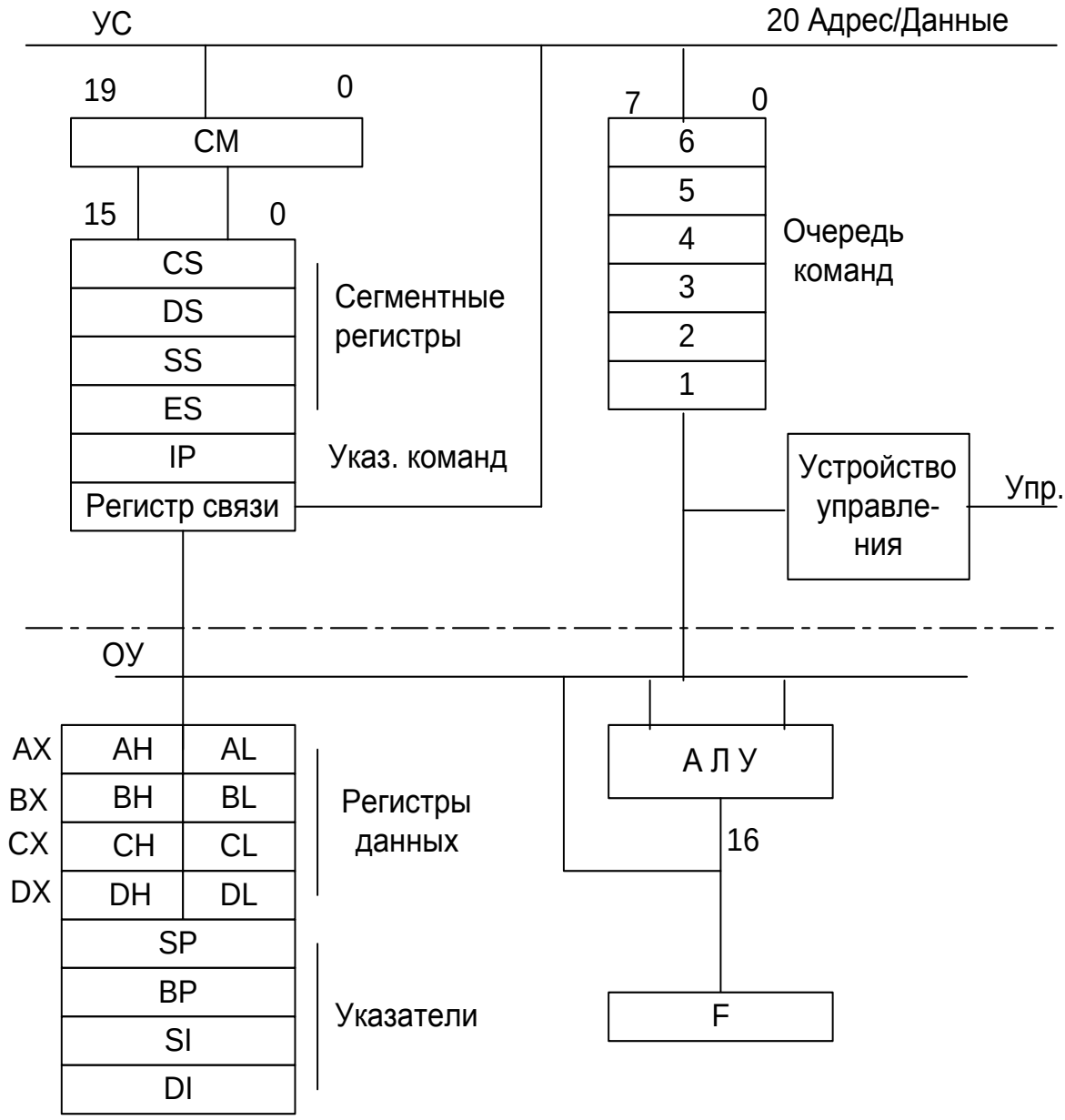


Рис. 1.2. Структурная схема МП i8086

В состав УС входят 16-битные сегментные регистры CS, DS, SS и ES и сумматор СМ, который формирует 20-битный физический адрес сегмента (базы) и смещения, называемого также эффективным (исполнительным) адресом ЕА. Это делается путём суммирования ЕА с содержимым сегментного регистра, сдвинутого относительно ЕА на 4 бита, как показано на рис. 1.3.

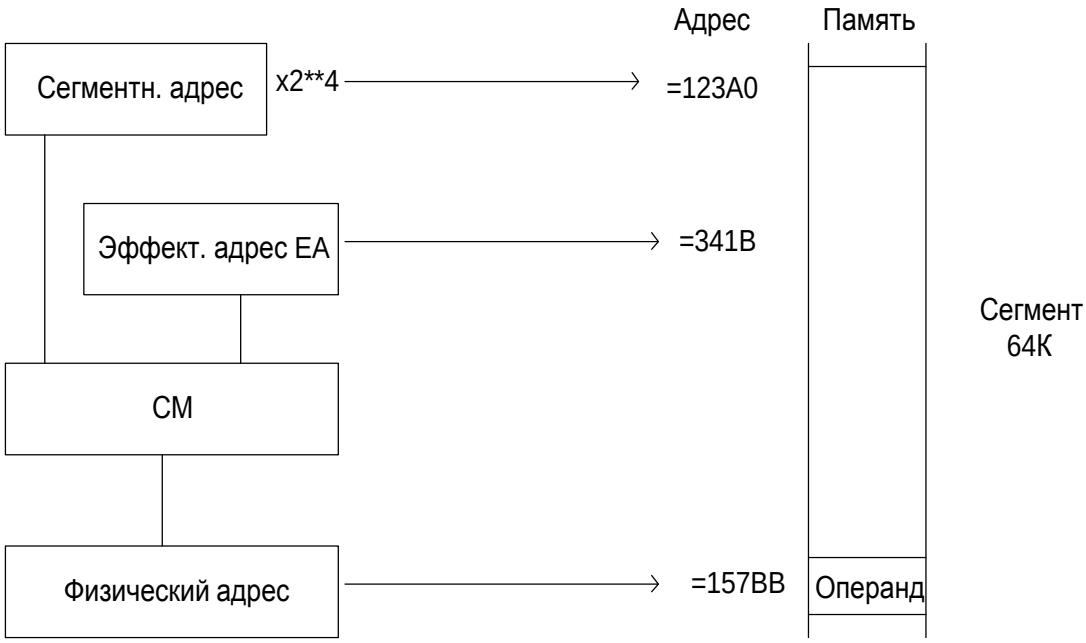


Рис. 1.3. Схема вычисления физического адреса

Если, например, содержимое сегментного регистра данных DS=123A, а указатель памяти SI=341B, то физический адрес операнда в памяти будет равен 157BB.

Пример 1.1

| | | |
|---|---|---|
| + | 1 2 3 A 0 | – начальный адрес сегмента данных DS·$2^4$ |
| | 3 4 1 B | – эффективный адрес в SI |
| | 1 5 7 B B | – физический адрес операнда в памяти |

### 1.4. Программная модель МП (рис. 1.4.)

В программной модели МП (рис. 1.4) можно выделить следующие 4 группы регистров.

1. Регистры данных

В зависимости от того, чем оперирует команда – словами или байтами, регистры данных можно рассматривать как четыре 16-битных (AX, CX, BX, DX) или как восемь 8-битных регистров (AL, AH, CL, CH, BL, BH, DL, DH). Символы L и H означают младшие и старшие байты 16-битных регистров. Каждый из этих регистров помимо общих выполняет и специализированные функции: AX, AL – аккумулятор; BX – базовый регистр (для косвенной адресации памяти); CX – счетчик в операциях с цепочками; DX – регистр данных или указатель при косвенной адресации регистров (портов) ввода/вывода.

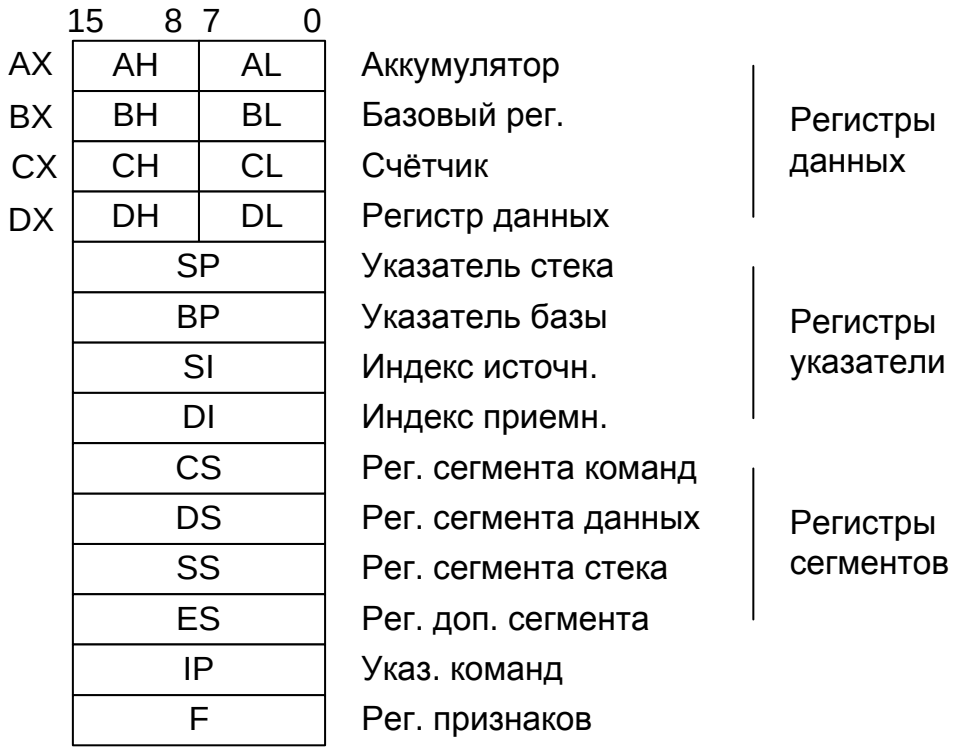


Рис. 1.4. Программная модель МП i8086

2. Регистры сегментов CS, DS, SS, ES

Как уже отмечалось ранее, память микросистемы на базе МП86 содержит сегменты памяти по 64 Кбайт. МП может иметь дело одновременно с четырьмя типами сегментов: кода (команд), стека, данных и дополнительного сегмента данных.

Сегментные регистры выполняют следующие функции.

*Регистр сегмента команд CS* указывает на сегмент, содержащий текущую выполняемую программу. Для вычисления адреса следующей (с учётом очереди команд) исполняемой команды к содержимому CS, умноженному на $2^4$, добавляется содержимое указателя команд IP.

*Регистр сегмента стека SS* указывает на текущий сегмент стека.

*Регистр сегмента данных DS* указывает на текущий сегмент данных, обычно содержащий используемые программой данные.

*Регистр дополнительного сегмента ES* указывает на текущий дополнительный сегмент, который используется для выполнения операций над строками.

3. Регистры указателей SP, BP и индексов DI, SI

Регистры предназначены для хранения внутрисегментных смещений и обеспечивают косвенную адресацию данных в пределах текущего сегмента. Эти же регистры могут участвовать в выполнении арифметических и логических операций над двухбайтными данными, т.е. используются при этом как регистры данных. Поэтому данную группу регистров и регистры

данных относят к регистрам общего назначения (РОН). Указатели стека SP и базы BP предназначены для доступа к данным в текущем сегменте стека. Индексные регистры DI (приёмника), SI (источника), а также регистр BX содержат смещение, которое по умолчанию относятся к сегменту данных (DS). В операциях с цепочками данных указатель DI по умолчанию относится к дополнительному сегменту (ES). Указатель команд IP адресует следующую команду программы (разумеется, с учётом очереди команд) в сегменте кодов (CS). Этот регистр ведёт себя как программный счётчик PC в 8-битном МП К580ВМ80 (i8080).

4. Регистр признаков F

Формат 16-битного регистра признаков F показан на рис. 1.5.

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

***Признаки (флажки):***

*OF – переполнение,*
*DF, IF, TF – признаки управления МП,*
*SF – знак, ZF – нуль,*
*AF – вспомогательный перенос,*
*PF – четность, CF – перенос.*

Рис. 1.5. Формат регистра признаков

Младший байт регистра полностью соответствует регистру признаков МП К580ВМ80. Кроме того, в старшем байте F содержится признак арифметического переполнения OF. В дополнение к этому в регистре признаков F фиксируются некоторые признаки, предназначенные для управления работой МП. DF – признак направления, указывающий на то, что обрабатываются цепочки: либо от меньших адресов к большим (DF=0), либо наоборот (DF=1). IF – признак прерывания: при IF=1 прерывание разрешено. TF – признак трассировки, разрешающий при TF=1 выполнять программу по командам (пошаговый режим).

### 1.5 Адресная организация памяти, представление данных

МП i8086 обеспечивает адресацию памяти ёмкостью $2^{20}$=1 Мбайт. На программном уровне память представляют как линейную последовательность из $2^{20}$ байт =1 Мбайт (рис. 1.6).

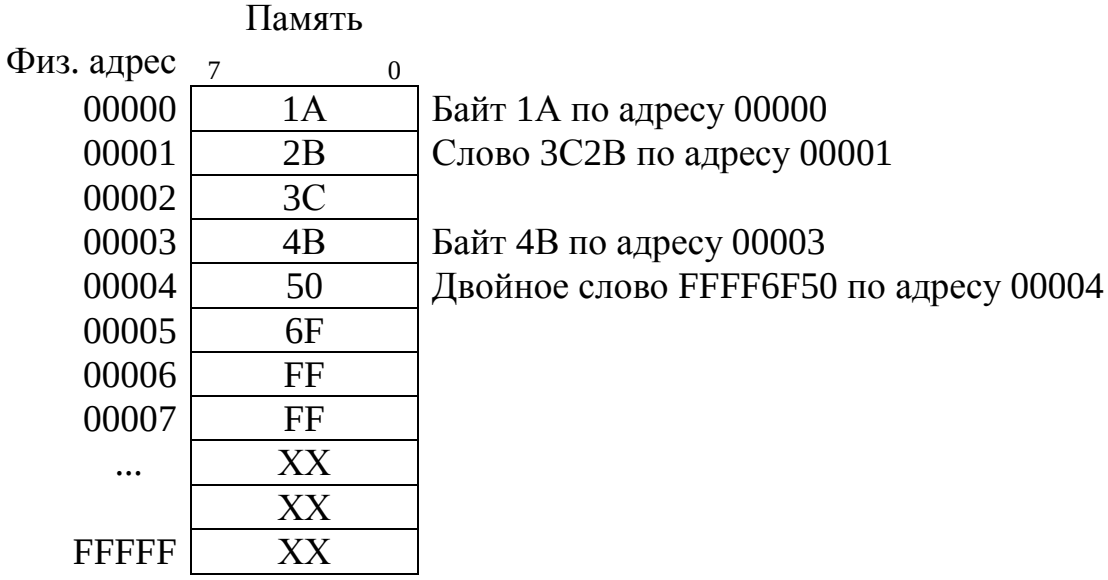


Рис. 1.6. Адресная организация памяти

Два смежных байта образуют слово. Слово может храниться по четному или нечетному адресу. В первом случае слово передается за один цикл шины МП, а во втором – за два.

Следовательно, для достижения наивысшей производительности МП необходимо слова размещать по четным адресам памяти. При этом адресом слова считается адрес его младшего байта, а старший байт размещается по более старшему адресу. Принцип “Младшее по младшему адресу” сохраняется и для представления других единиц данных: многобайтных команд, двойных слов и т.д.

Отметим, что выравнивание команд по четным адресам практически не влияет на производительность МП, т. к. он выбирает их в очередь команд с опережением. Двойное слова содержит 4 байта, а квадрослово – 8.

### 1.6. Примеры форматов команд МП i8086

Примеры форматов команд с регистровой (1, 5, 6) , непосредственной (2, 7, 8) и прямой (3, 4) адресацией приведены на рис. 1.7.

В обобщенном представлении команд, например ADD reg1,reg2, после мнемоники команды указывается приемник, а затем через запятую – источник операнда. В байтах data L, data H команды указываются младшая и старшая части (если W=1) константы соответственно.

Во всех командах содержится однобитное поле W: если W=1, то команда оперирует словом, если W=0, то – байтом.

В поле reg (или reg1, reg2, r/m) команд указываются трехбитные адреса РОНов МП. В табл. 1.1 приведены адреса регистров МП для различных значений W.

1. Передача регистр – регистр MOV reg1,reg2

| 1-й байт | 2-й байт |
|---|---|
| 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
| 1 0 0 0 1 0 1 w | 1 1 reg1 reg2 |

2. Передача константы в регистр MOV reg,data

| | | | |
|---|---|---|---|
| 1 0 1 1 w reg | Data L | Data H | ,если w=1 |

3. Передача из памяти в аккумулятор MOV acc,[EA]

| | | |
|---|---|---|
| 1 0 1 0 0 0 0 w | Мл. часть EA | Ст. часть EA |

4. Передача из аккумулятора в память MOV [EA],acc

| | | |
|---|---|---|
| 1 0 1 0 0 0 1 w | Мл. часть EA | Ст. часть EA |

5. Сложить регистр – регистр ADD reg1,reg2

| | |
|---|---|
| 0 0 0 0 0 0 1 w | 1 1 reg1 reg2 |

6. Вычесть регистр – регистр SUB reg1,reg2

| | |
|---|---|
| 0 0 1 0 1 0 1 w | 1 1 reg1 reg2 |

7. Сложить константу с аккумулятором ADD acc,data

| | | | |
|---|---|---|---|
| 0 0 0 0 0 1 0 w | Data L | Data H | ,если w=1 |

8. Вычесть константу из аккумулятора SUB acc,data

| | | | |
|---|---|---|---|
| 0 0 1 0 1 1 0 w | Data L | Data H | ,если w=1 |

Рис 1.7. Примеры форматов команд с регистровой, непосредственной и прямой адресацией

Таблица 1.1. Адреса регистров МП

| Адрес регистра reg, r/m | Регистр | |
|---|---|---|
| | W=1 | W=0 |
| 000 | AX | AL |
| 001 | CX | CL |
| 010 | DX | DL |
| 011 | BX | BL |
| 100 | SP | AH |
| 101 | BP | CH |
| 110 | SI | DH |
| 111 | DI | BH |

Рассмотрим примеры кодирования команд.

**Пример 1.2.** Команды передачи

| Код команды | Мнемоника | Операция |
|---|---|---|
| 10001011<br>11000011 | MOV AX,BX | AX←BX |
| 10001010<br>11100111 | MOV AH,BH | AH←BH |
| BA<br>A7<br>02 | MOV DX,02A7H | DX←02A7H |
| A1<br>27<br>10 | MOV AX,[1027H] | AX← [1027H] |

### 1.7. Пример разработки программы в машинных кодах

При программировании на машинном языке адреса, коды команд и данные записываются в 16-ричной системе счисления.

Разработаем программу для вычисления выражения

$$M=K+N-R+120, \qquad (1.1)$$

в котором все операнды и результат – двухбайтные данные. K, R, M размещены в сегменте данных, а N – в регистре DX. Числа со знаком представлены в дополнительном коде (ДК). Например, ДК чисел K=+60, R=-10, N=+30 соответственно равны $[K]_{ДК}$=0060, $[R]_{ДК}$=FFF0, $[N]_{ДК}$=0030. Результат M=+1C0, $[M]_{ДК}$=01C0.

*Правило образования ДК*.

ДК положительного числа есть само число, представленное в формате байта или слова. ДК отрицательного числа есть инверсия (not) числа плюс единица, например в формате слова $[-30]_{ДК}$=not(0030)+1=FFCF+1=FFD0. Инверсия 16-ричной цифры равна: not($\alpha$)=F – $\alpha$.

В табл. 1.2 приведена программа вычисления выражения (1.1). Начальный эффективный адрес программы равен 100. Этому адресу соответствует физический адрес CS·$2^4$+100. Далее его будем обозначать как CS:0100. Эффективные адреса данных лежат в диапазоне 0500-0505, а физические – в диапазоне DS:0500 - DS:0505.

Таблица 1.2. Программа вычисления выражения M=K+N-R+120

| Адрес | Код | Мнемоника | Операция |
|---|---|---|---|
| CS:0100 | A1 | MOV AX,[0502H] | AX←R |
| 0101 | 02 | | |
| 0102 | 05 | | |
| 0103 | 2B | SUB DX,AX | DX←N-R |
| 0104 | D0 | | |
| 0105 | A1 | MOV AX,[0500H] | AX←K |
| 0106 | 00 | | |
| 0107 | 05 | | |
| 0108 | 03 | ADD AX,DX | AX←K+N-R |
| 0109 | C2 | | |
| 010A | 05 | ADD AX,0120H | AX←AX+120 |
| 010B | 20 | | |
| 010C | 01 | | |
| 010D | A3 | MOV [0504H],AX | [0504]←M |
| 010E | 04 | | |
| 010F | 05 | | |
| 0110 | 90 | NOP | Пустая операция |
| Адреса данных | Данные в ДК | | |
| DS:0500 | 60 | K=0060 | Операнд K=+60 |
| 0501 | 00 | | |
| 0502 | F0 | R=-0010 | Операнд R=-10 |
| 0503 | FF | | |
| 0504 | XX | M=XXXX | Результат M |
| 0505 | XX | | |
| РОН | Данные | | |
| DX | 0030 | N=0030 | Операнд N=+30 |

### 1.8. Отладчик TD

Отладчик TD – это системная программа, находящаяся на диске ПЭВМ. Отладчик позволяет управлять процессом исполнения пользовательской программы. Команды TD, вводимые с клавиатуры ПЭВМ, позволяют выводить на экран и изменять содержимое памяти и регистров МП, исполнять программу по шагам (командам) и др. Команды отладчика будут рассмотрены при выполнении работ.

### 1.9. Варианты заданий

Таблица 1.3. Варианты заданий

| № варианта задания | Размещение операндов | | | | Длина операнда | Нач. адр. прогр. |
|---|---|---|---|---|---|---|
| | M | K | R | N | | |
| 1.M=K+R+2N-3 | Память | | | BH | Байт | 100 |
| 2.M=K-R-N+50 | Память | | DX | Пам. | Слово | 100 |
| 3.M=2K-N+R-6 | Пам. | CL | Память | | Байт | 100 |
| 4.M=K+3N-R+20 | Память | | | DX | Слово | 100 |
| 5.M=K-R-N+8 | Память | | CL | Пам. | Байт | 100 |
| 6.M=2K-R+N+18 | Пам. | DX | Память | | Слово | 100 |
| 7.M=2K-2R+N-10 | Память | | DL | Пам. | Байт | 100 |
| 8.M=120-K-R+N | Пам. | CX | Память | | Слово | 100 |
| 9.M=20+K-2R+N | Память | | | CH | Байт | 100 |
| 10.M=500+K-2R+N | Пам. | BX | Память | | Слово | 100 |

### 1.10. Порядок выполнения работы

Ниже изучаются команды программы Norton Commander и отладчика TD персональной ЭВМ, с помощью которых осуществляются ввод и отладка программ в машинных кодах микропроцессора МП86. В качестве примера используется программа вычисления выражения M=K-R+N+120, приведенная в табл. 1.2.

**Создание исходного файла программы**

1. Для создания в системе Norton Commander (NC) нового файла программы последовательно выполните пп. 2-4, при корректировке файла – пп. 5-7. Вызов системы производится командой nc.
2. При нажатой клавише Shift нажмите клавишу F4 – программа NC выдаст запрос. Введите в нем имя файла и его расширение txt и нажмите Enter (↵). Например, petrov.txt↵. Еще раз нажмите Enter и попадете в NC.
3. С помощью команд встроенного редактора NC наберите программу (табл. 1.2) в машинных кодах:

```
a1 0205
2bd0
a1 0005
03c2
05 2001
a3 0405
90
```

4. Чтобы сохранить созданный файл на диске, нажмите F2, а затем F10 – вы выйдете из редактора.

5. Если потребуется скорректировать или дополнить известный файл, то в панели системы NC курсором выделите имя этого файла и нажмите F4 – вы попадете в редактор с вызванным файлом.

6. С помощью команд редактора NC скорректируйте исходный файл. Например, перед машинной командой 90 введите еще несколько кодов 90 (пустая операция NOP).

7. Чтобы сохранить созданный файл на диске, нажмите F2, а затем F10 – вы выйдите из редактора.

Выделенный в панели NC файл можно распечатать на принтере, если нажать F5 – копировать, а затем при ответе на запрос ввести PRN (принтер) и нажать ↵.

**Преобразование исходного модуля программы в машинных кодах в исполняемый модуль типа COM**

8. С помощью команды

trans <имя файла>.txt↵

запустите программу трансляции TRANS, которая преобразует коды команд МП86, представленные в виде ASCII-символов, в последовательность двоичных кодов команд, образующих COM-файл программы.

**Выполнение и отладка исполняемого модуля программы с помощью турбоотладчика TD**

9. С помощью команды

td <имя файла>.com↵

запустите отладчик TD для работы с созданным COM-файлом программы. Отладчик загрузит в память исполняемый модуль типа COM с адреса 100h, причем коды программы и данных разместятся в одном сегменте кода емкостью 64Кбайт. После загрузки отладчик выдаст на экран монитора окно процессора CPU.

10. Нажав ENTER, снимите марку отладчика. На экране изобразится окно отладчика CPU, состоящее из 5 подокон: кодового сегмента, содержащего коды программы; регистров микропроцессора; регистров флажков; сегмента стека и сегмента данных. Клавишей F5 увеличите/уменьшите окно CPU. Двойной рамкой и находящимся в подокне маркером выделено активное подокно (или окно). Переход из одного подокна в другое производится нажатием клавиши Tab или Shift-Tab. Можно перейти в подокно, нажимая Shift и одну из клавиш перемещения курсора.

Находясь в подокне, можно, нажав Alt-F10, войти в локальное подменю и с помощью его команд изменить содержимое регистров памяти. Нажав F10, можно войти в главное меню отладчика и воспользоваться его командами для управления выполнением программы. Выход из меню производится клавишей Esc.

Ниже рассматриваются основные команды отладчика для отладки программ с помощью окна процессора CPU.

11. Нажав клавишу Tab, перейдите в подокно регистров. Подведите маркер к регистру DX, введите код 30 и нажмите ENTER . Тем самым вы ввели в DX операнд N=0030h. (При вводе 16-ричных кодов необходимо, чтобы 1-й символ начинался с цифры, например 0FFFF.)

12. Перейдите в подокно сегмента данных. Нажав Alt-F10, вызовите для данного подокна локальное меню. Выберите в нем команду GOTO и нажмите ENTER. Далее введите с клавиатуры начальный адрес данных ds:500↵. Убедитесь в том, что маркер выделяет в подокне памяти байт с адресом 500, а затем введите с клавиатуры коды 60 00 0F0 0FF↵. Тем самым вы ввели в память операнды K=+60 и R=-10.

Можно вводить данные в память в формате слова. Для этого с помощью команды Ctrl-D войдите в подменю локального меню и задайте в нем формат WORD – слово. Затем введите с клавиатуры необходимый операнд, например 60 по адресу 500.

Отметим, что Ctrl-D – это активная клавиша команд Alt-F10; Display as. Далее при работе с окнами отладчика будем в основном использовать активные клавиши, нажимая Ctrl в сочетании с первой буквой команды локального меню.

13. Перейдите в подокно кода. В нем изображаются дизассемблированные команды выполняемой программы, причем текущая команда помечается стрелкой. Для управления выполнением программы используют следующие основные команды, вызываемые активными клавишами:

| | |
|---|---|
| F7 | Выполнение одной команды |
| F8 | Выполнение одной команды с пропуском вызовов |
| F9 | Запуск программы в автоматическом режиме |
| F4 | Выполнение команд до точки останова |
| Ctrl-F2 | Установка программы в исходное состояние |
| F2 | Установка/отмена точки останова |

Нажимая последовательно F7, выполните программу по шагам до команды NOP. При этом следите за содержимым регистров МП и памяти, убедитесь в правильности полученного результата.

14. (Данный пункт выполнять необязательно). Нажав Ctrl-F2, установите программу в исходное состояние. Обратите внимание, что указатель отмечает первую команду программы. Перезагрузите исходные данные (см. пп. 11-12). С помощью маркера выделите 3-ю команду программы и нажмите F2 – таким образом вы задали точку останова. Аналогичным образом установите точку останова на команде NOP. Для выполнения программы до 1-й точки останова нажмите F9 (или F4). Далее выполните программу по шагам, последовательно нажимая F7.

15. Нажав Ctrl-F2, установите программу снова в исходное состояние. Перезагрузите исходные данные (см. пп. 11-12). С помощью маркера и команды F2 отметьте точки останова. Маркером выделите команду программы ADD AX,0120 и замените ее с клавиатуры на команду ADD AX,0420 и нажмите ENTER. Тем самым вы установили так называемую “заплату”. Выполните модифицированную программу в пошаговом режиме.

**Работа с главным меню и окнами отладчика**

Ранее были рассмотрены основные команды отладчика, вызываемые активными клавишами. В общем случае управление отладкой можно производить через главное меню отладчика. Это меню содержит позиции, указанные в верхней части экрана: File (файлы), View (просмотр), Run (выполнение) и другие. Вход в главное меню производится с помощью клавиши F10 и выбора необходимой позиции. Вызов новых окон, например памяти (DUMP) или регистров (REGISTERS), производится через позицию View.

16. Выведите на экран окно памяти, для чего выполните команду F10; View; Dump; ENTER. Работа в этом окне аналогична подокну данных окна CPU.

17. Нажав Ctrl-F5, установите режим перемещения и изменения размеров текущего окна. Нажимая клавиши перемещения курсора, переместите окно. Этими же клавишами можно изменить размеры окна, если держать нажатой клавишу Shift. Завершается этот режим нажатием клавиши ENTER.

18. Командой F10; View; Registers на экран окно регистров Registers.

19. Нажимая последовательно F6, можно перейти из одного окна в другое. Двойной рамкой или находящимся в ней маркером выделяется активное окно. Командой Alt-F3 текущее окно удаляется. Ошибочно удаленное окно можно восстановить командой Alt-F6. Удалите все окна, кроме окна CPU.

20. Для выхода из отладчика нажмите Alt-X.

21. Разработайте для заданного варианта программу вычисления выражения в машинных кодах МП86 и по аналогии с пп. 1-20 введите ее в ЭВМ и выполните. Исходные данные и результат представьте в ДК.

### 1.11. Содержание отчета

1. Программа вычисления заданного выражения с представлением исходных данных в ДК. Распределение ячеек памяти для данных и программы.
2. Трасса отлаженной программы.

## Лабораторная работа 2

## ФОРМАТЫ КОМАНД И РЕЖИМЫ АДРЕСАЦИИ МП i8086

### 2.1. Цель работы

Изучение форматов команд и режимов адресации МП i8086, отработка навыков работы с отладчиком TD.

### 2.2. Форматы команд МП i8086

Полный перечень команд дается в приложении.

Длина команд МП i8086 варьируется от одного до шести байт. В первых (одном или двух) байтах команды находятся код операции КОП и указание режима адресации. На рис. 2.1 приведен наиболее общий формат двухоперандной команды (3-й и 4-й байты необязательны). Второй байт команды называют еще постбайтом.

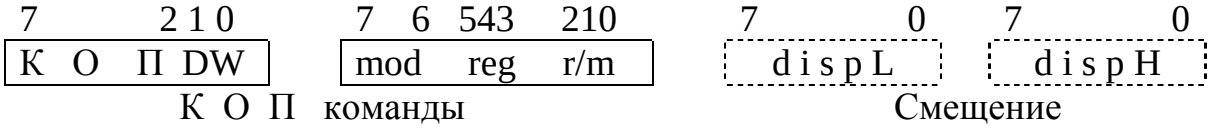


Рис. 2.1. Формат двухоперандной команды

В КОП имеются следующие однобайтные поля.

Бит W определяет операцию с байтом (W=0) или словом (W=1).

Бит D показывает, чем является регистр reg: операндом-источником (D=0) или операндом-получателем (D=1).

Если на код операции команды отводится два байта (например, рис. 2.1), то во втором байте имеются двухбитное поле режима адресации mod и трехбитное поле r/m. В зависимости от значения mod поле r/m определяет операнд в РОНе МП, если mod =11, или – в памяти, если mod # 11.

Для mod=11 адрес регистра МП, в котором размещен операнд, указывается полем r/m в соответствии с табл. 1.1.

Если mod=11, а D=1, то общий формат двухоперандной команды (рис.2.1) трансформируется в двухбайтную команду типа “регистр – регистр”. На рис. 1.7 приведены форматы таких команд: это команды MOV reg1,reg2; ADD reg1,reg2; SUB reg1,reg2. В них обозначены: reg1=reg; reg2=r/m. В качестве регистра-приемника выбран регистр reg, поэтому D=1.

Если mod#1, то эффективный адрес EA операнда в памяти вычисляется в соответствии с табл. 2.1.

Таблица 2.1. Способы вычисления адреса EA операнда в памяти

| mod<br>r/m | 00 | 01 | 10 |
|---|---|---|---|
| 000 | (BX)+(SI) | (BX)+(SI)+disp L | (BX)+(SI)+disp H,L |
| 001 | (BX)+(DI) | (BX)+(DI)+disp L | (BX)+(DI)+disp H,L |
| 010 | (BP)+(SI) | (BP)+(SI)+disp L | (BP)+(SI)+disp H,L |
| 011 | (BP)+(DI) | (BP)+(DI)+disp L | (BP)+(DI)+disp H,L |
| 100 | (SI) | (SI)+disp L | (SI)+disp H,L |
| 101 | (DI) | (DI)+disp L | (DI)+disp H,L |
| 110 | disp H,L | (BP)+disp L | (BP)+disp H,L |
| 111 | (BX) | (BX)+disp L | (BX)+disp H,L |

Отметим, что для mod=00 и r/m=110 смещением disp H.L задается 16-битный эффективный адрес EA операнда в памяти.

Если mod=01, то третий байт команды содержит 8-битное смещение dispL, которое при вычислении EA автоматически расширяется со знаком до 16 бит. Если mod=10, то 3-й и 4-й байты команды содержат 16-битное смещение disp H,L.

В однооперандных командах с постбайтными режимами адресации отсутствует однобитное поле D, а на месте поля reg размещается расширение поля КОП.

Формат двухоперандной команды с непосредственным операндом показан на рис. 2.2.

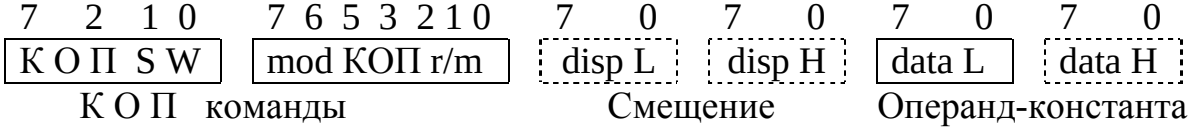


Рис. 2.2. Формат двухоперандной команды с непосредственным операндом

Назначение полей mod и r/m в этой команде сохраняется таким же, как и для команды, приведенной на рис. 2.1

Бит S совместно с битом W (поле S:W) определяют размер непосредственного операнда:

S:W= ⎧ X0 – один байт данных data L,
⎨ 01 – два байта (слово) данных data H, data L,
⎩ 11 – один байт данных data L, который расширяется до16 бит со знаком

Операнд-константа является всегда источником.

Микропроцессор i8086 имеет в своем составе команды специального формата, которые позволяют сократить постбайт в часто используемых командах. Это команды пересылки в регистр и операций с аккумулятором

AX (или AL) и непосредственным операндом. На рис.1.7 приведены специальные (укороченные) форматы команд:

```
MOV  reg,data ;    MOV  acc,[EA] ;   ADD  acc,data ;
MOV  [EA],acc ;    SUB  acc,data
```

Следует отметить, что при трансляции ассемблерных команд ассемблер всегда выбирает более короткий формат транслируемой команды.

### 2.3. Режимы адресации

Большинство команд МП86, за исключением команд передачи цепочки байт, имеет тип “регистр – регистр”, “регистр – память” и “память – регистр”. Это означает, что его команда адресует максимум два операнда и что не допускается одновременная адресация двух ячеек памяти.

Как видно из табл. 2.1, МП имеются все основные режимы адресации, характерные для современных процессоров: регистровая, непосредственная, прямая (абсолютная), косвенная регистровая, базовая, индексная и некоторые их модификации.

1. РЕГИСТРОВАЯ АДРЕСАЦИЯ. Этот способ адресации кодируется в 1-м байте или постбайте команды и определяет, что операнд находится в одном из регистров МП.

2. НЕПОСРЕДСТВЕННАЯ АДРЕСАЦИЯ. Операнд содержится в самой команде и имеет длину 8 или 16 бит. С помощью команды (рис. 2.2) можно загрузить константу как в РОН, так и в память. Вместе с тем следует отметить, что МП86 не имеет команды загрузки непосредственного операнда в сегментные регистры.

3. ПРЯМАЯ АДРЕСАЦИЯ. В команде содержится двухбайтный эффективный адрес ЕА операнда, размещенный вслед за 1-м байтом команды (специальный укороченный формат) или за постбайтом, если mod=00, а r/m=110.

На рис. 1.7 приведен ряд примеров кодирования команд с непосредственной и прямой адресацией данных для специальных (укороченных) форматов. Ниже, в табл. 2.2, даны примеры кодирования команд со стандартными форматами, приведенными на рис. 2.1 и 2.2, для регистровой, непосредственной и прямой адресации. Последовательность этих команд образует программу вычисления выражения M=K+N–R +120 (см. п. 1.7 лабораторной работы 1).

Таблица 2.2. Вариант программы вычисления выражения M=K+N–R+120

| Адрес команды | Код команды | Мнемоника | Комментарий |
|---|---|---|---|
| CS: 0100 | 2B | SUB DX,[0502H] | DX←N – R |
| 0101 | 16 | | |
| 0102 | 02 | | |
| 0103 | 05 | | |
| 0104 | 03 | ADD DX,[0500H] | DX←N – R + K |
| 0105 | 16 | | |
| 0106 | 00 | | |
| 0107 | 05 | | |
| 0108 | 81 | ADD DX,0120H | DX←N – R + K +120 |
| 0109 | C2 | | |
| 010A | 20 | | |
| 010B | 01 | | |
| 010C | 89 | MOV [0504H],DX | M←DX |
| 010D | 16 | | |
| 010E | 04 | | |
| 010F | 05 | | |
| 0110 | 90 | NOP | |

Как видно, данная программа содержит меньшее число команд, однако ее длина в числе байт одинакова с программой, приведенной в табл. 1.2 и реализующей ту же задачу (см. лаб. работу 1).

4. КОСВЕННАЯ РЕГИСТРОВАЯ АДРЕСАЦИЯ. Режим регистровой косвенной адресации задается в постбайте с помощью поля mod=00 и поля r/m=100, 101 или 111, определяющего адрес регистров SI, DI или BX, где хранится 16-битный эффективный адрес операнда.

**Пример 2.1.** Команда MOV DX,[BX] пересылки в регистр DX содержимого ячейки сегмента данных, адрес которой указан в регистре BX

| Код команды | Мнемоника | Операция |
|---|---|---|
| КОП DW<br>100010 1 1<br>00 010 111<br>mod DX r/m | MOV DX,[BX] | DX←M[BX] |

**Пример 2.2.** Команда MOV [SI],AX

| | | |
|---|---|---|
| 10001001<br>00000100 | MOV [SI],AX | M[SI]←AX |

5. БАЗОВАЯ АДРЕСАЦИЯ. При базовой адресации (называемой также адресацией по базе или адресацией типа “база + смещение”) эффективный адрес операнда является суммой значений смещения и содержимого регистров BX или BP:

$$EA = \begin{Bmatrix} (BX) \\ (BP) \end{Bmatrix} + \begin{Bmatrix} \text{8-битное смещение disp L} \\ \text{16-битное смещение disp H,L} \end{Bmatrix}$$

Напомним, что регистр базы BP позволяет обращаться по умолчанию только к сегменту стека.

Базовый регистр указывает на начало структуры, а требуемый элемент адресуется с помощью смещения (расстояния) от базы.

Следующие команды эквивалентны:

MOV AH,[BX+4]; MOV AH,4[BX]; MOV AH,[BX]+4.

Любая из них реализует передачу в AH байта памяти с адресом EA, равным сумме содержимого BX со смещением 4: AH←M[BX] +4

6. ИНДЕКСНАЯ АДРЕСАЦИЯ. Эффективный адрес EA вычисляется как сумма:

$$EA = \begin{Bmatrix} (DI) \\ (SI) \end{Bmatrix} + \begin{Bmatrix} \text{8-битное смещение disp L} \\ \text{16-битное смещение disp H,L} \end{Bmatrix}$$

Смещение определяет фиксированный начальный адрес массива, а значение в DI (или SI) – номер элемента в массиве.

Следующие команды эквивалентны:

MOV BL,[10+SI]; MOV BL,10[SI]; MOV BL,10+[SI].

По существу режимы базовой и индексной адресации МП К1810 одинаковы. Они задаются в постбайте полем mod=01, 10 для r/m=110, 111 (базовая адресация) и для r/m=100, 101 (индексная адресация).

**Пример 2.3.** Команда MOV BH,[10H+DI]

| Код команды | Мнемоника | Операция |
|---|---|---|
| КОП DW<br>100010 1 0<br>01 110 101<br>mod BH r/m<br>00010000 | MOV BH,[10H+DI] | BH←M[10+DI] |

7. БАЗОВАЯ ИНДЕКСНАЯ АДРЕСАЦИЯ. Эффективный адрес равен сумме:

$$EA = \begin{Bmatrix} (BX) \\ (BP) \end{Bmatrix} + \begin{Bmatrix} (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8-битное смещение disp L} \\ \text{16-битное смещение dispH,L} \end{Bmatrix}$$

С помощью базовой индексной адресации возможно обращение к элементам двумерного массива. Приведем несколько примеров:

MOV AX,[BX+DI+4]; ADD DX,[BX+SI+6]; ADD BX,[BX]+[DI+б].

8. ОТНОСИТЕЛЬНАЯ АДРЕСАЦИЯ. Применяется в командах условных и безусловных переходов для вычисления адреса EA команды, к которой осуществляется переход. Формат команд коротких переходов с относительной адресацией имеет вид:

| 7 | 0 |
|---|---|
| К О П | |

| 7 | 0 |
|---|---|
| Смещение disp L | |

Эффективный адрес EA вычисляется как сумма 8-битного смещения**,** находящегося в команде, и текущего значения указателя команд EA=(IP)+<8-битное смещение в ДК>. Следовательно, этот режим обеспечивает передачу управления в диапазоне -128  +127 байт от текущей команды. Адрес текущей команды IP=IP+2, так как все команды управления с относительной адресацией двухбайтные. Отрицательное смещение означает переход назад. Смещение в команде представляют в дополнительном коде ДК.

**Пример 2.4.** Код команды  JNZ MM1

| | Адрес команды | Код команды | Мнемоника | Операция |
|---|---|---|---|---|
| EA | 0200 | - | MM1: ADD  AX,1204 | |
| | 0201 | 05 | | |
| | 0202 | 20 | | |
| | 0203 | 01 | | |
| IP | 0204 | 75 | JNZ  MM1 | ;IP←(IP + 2) + FB |
| | 0205 | FA | | ;если FZ=0 |
| IP+2 | 0206 | 90 | NOP | ;Следующая ;команда |

Чтобы определить ДК смещения dispL для команды JZN 200, необходимо вначале найти <Смещение dlsp L>=EA – (IP+2).

Для нашего примера <Смещение disp L>=200–(204+2)=–6, где EA=200 – эффективный адрес команды, к которой осуществляется переход (это команда с меткой MM1); (IP)=204 – это адрес команды перехода JNZ MM1. ДК[–6]=FA.

### 2.4. Программа сложения многобайтных BCD-чисел

Программа (табл.2.3) реализует сложение многобайтных десятичных BCD-чисел из массивов с начальными адресами ADR1=500, ADR2=510. Сумма формируется с адреса ADR3=520. Длина массива 4 байта.

Таблица 2.3. Программа сложения многобайтных BCD-чисел

| Адрес команды | Код команды | Мнемоника команды | Операция |
|---|---|---|---|
| CS:0100 | B9 | MOV CX,0004H | ;CX←4 |
| 0101 | 04 | | |
| 0102 | 00 | | |
| 0103 | BE | MOV SI,0000H | ;SI←0 |
| 0104 | 00 | | |
| 0105 | 00 | | |
| 0106 | F8 | CLC | ;CF←0 |
| 0107 | 8A | RR1: MOV AL,500H[SI] | ;Сложение |
| 0108 | 84 | | |
| 0109 | 00 | | |
| 010A | 05 | | |
| 010B | 12 | ADC AL,510H[SI] | ;BCD-чисел |
| 010C | 84 | | |
| 010D | 10 | | |
| 010E | 05 | | |
| 010F | 27 | DAA | ;Коррекция |
| 0110 | 88 | MOV 520H[SI],AL | |
| 0111 | 84 | | |
| 0112 | 20 | | |
| 0113 | 05 | | |
| 0114 | 46 | INC SI | |
| 0115 | 49 | DEC CX | |
| 0116 | 75 | JNZ RR1 | |
| 0117 | EF | | |
| 0118 | 90 | NOP | |

| Адреса данных | Данные | Комментарии |
|---|---|---|
| DS: 0500 | 37 | X=25792137 |
| 0501 | 21 | |
| 0502 | 79 | |
| 0503 | 25 | |
| 0510 | 02 | Y=56986602 |
| 0511 | 66 | |
| 0512 | 98 | |
| 0513 | 56 | |
| 0520 | 39 | X+Y=82778739 |
| 0521 | 87 | |
| 0522 | 77 | |
| 0523 | 82 | |

### 2.5. Варианты заданий

1. Вычислить Z=2X+Y, где X, Y – многобайтные BCD-числа. Число байт 3 (варианты 1-10 повышенной сложности).
2. Вычислить Z=X+Y+R, где X, Y, R – многобайтные числа в ASCII-формате. Число байт 4.
3. Найти в массиве двухбайтных знаковых чисел максимальное число. Длина массива 16 байт.
4. Вычислить Z=X-2Y, где X, Y – многобайтные числа в ASCII-формате. Число байт 4.
5. Вычислить Z=X-Y-R, где X, Y, R – многобайтные числа. Число байт 4.
6. Найти в массиве двухбайтных знаковых чисел минимальное число. Длина массива 32 байта.
7. Дана матрица однобайтных чисел размером 4x5. Составить программу подсчета количества положительных элементов матрицы.
8. Дана матрица двухбайтных чисел размером 3x4. Составить программу подсчета количества отрицательных элементов матрицы.
9. Дано N произвольных чисел. Сформировать их в массивы отрицательных и положительных чисел.
10. Дана матрица однобайтных чисел размером 4x5. Подсчитать число элементов, значения которых < 10.
11. Переслать содержимое одной области памяти в другую. Число слов в массиве 10.
12. Переслать содержимое одной области памяти в другую. Число байт в массиве 12.
13. Сложить 8 двухбайтных чисел со знаком, размещенных в памяти.
14. Сложить 10 однобайтных чисел со знаком, размещенных в памяти.
15. Сложить 6 BCD-чисел, размещенных в памяти.

16. Найти в массиве символов, представленных в ASCII-коде, цифру 5 (ее код 35). Длина массива 16

17. Найти число положительных двухбайтных чисел в массиве. Длина массива 8.

18. Найти число отрицательных чисел двойной длины**.** Длина массива 8.

19. Произвести инверсию слов, размещенных в массиве. Длина массива 10.

20. Осуществить преобразование вида X←X+1 над словами, размещенными в памяти. Число слов 8.

### 2.6. Порядок выполнения работы

1. Изучите форматы команд и режимы адресации 16-битного МП86.

2. Разработайте в машинных кодах МП86 программу, содержащую 4 различных варианта сложения двухбайтных чисел d←d+s. Операнды источника s и приемника d размещены в памяти по эффективным адресам: источник – 500+2N, приемник – 500+2(N+1), где N – номер варианта в 16-ричной системе счисления.

Варианты программы должны отличаться режимами адресации операндов приемника и источника:

– абсолютная и базовая;

– индексная и косвенная;

– косвенная и базовая;

– индексная и индексная.

Варианты отделяйте друг от друга командой NOP.

3. Составьте блок-схему и программу вычисления выражения для заданного варианта (п. 2.5).

4. С помощью рассмотренных в работе № 1 редактора NC и отладчика TD введите в ЭВМ программу сложения двухбайтных чисел для различных режимов адресации и проверьте правильность ее выполнения.

Перед выполнением каждого варианта программы с помощью команд отладчика необходимо задавать начальные значения регистров МП и памяти.

5. С помощью команд редактора NC и отладчика TD введите в ЭВМ разработанную в соответствии с заданием (п. 3) программу и выполните ее в пошаговом режиме. Трассу программы приведите в отчете.

### 2.7. Содержание отчета

1. Программы сложения двухбайтных чисел для различных режимов адресации.

2. Блок-схема и программа для заданного варианта задания.

3. Исходные данные и результаты вычислений.

4. Трасса программы в пошаговом режиме.

**Лабораторная работа 3**

# ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ АССЕМБЛЕРА МП I8086. АССЕМБЛЕР TASM

## 3.1. Цель работы

Изучение элементов программирования на языке ассемблера МП 80X86. Знакомство и приобретение навыков работы с ассемблером tasm и компоновщиком tlink при создании исполняемых модулей типа .COM.

## 3.2. Элементы языка ассемблера МП 80X86

Программа на языке ассемблера представляет собой последовательность предложений (операторов), описывающих выполняемые операции. Каждый командный оператор может содержать до 4 полей (в скобках указаны необязательные поля):

[Метка:] Мнемокод [Операнд] [;Комментарий]

Например,

```
MM1:    MOV     AX,DX    ;Переслать в AX из регистра DX
```

ПОЛЕ МЕТКИ не должно начинаться с цифры. Метки могут содержать символы верхнего и нижнего регистров: буквы, цифры, символы подчеркивания _ и знаки $ @ ?. Максимальная длина метки 31 символ.

ПОЛЕ МНЕМОКОДА содержит мнемонику команды, например MOV, ADD.

ПОЛЕ ОПЕРАНДОВ. В нем указываются данные, с которыми работает команда.

ПОЛЕ КОММЕНТАРИЯ всегда начинается с символа “;”.

Для описания операндов могут быть использованы метки, константы, регистры МП, переменные памяти, текстовые строки, выражения.

В численной константе основание системы счисления определяется буквой, указанной после константы (B – 2-, O – 8-, D (или ничего) – 10-, H – 16-ричная система счисления). Константа всегда должна начинаться с цифры. Символьная константа – это символ, заключенный в кавычки. Строковые константы также заключаются в кавычки. Примеры констант: 10001111B, 23O, 45, 0A3H, “4”, “Таблица”.

ДИРЕКТИВЫ АССЕМБЛЕРА имеют формат (см. табл. 3.1):

[Метка] Директива [Операнд] [;Комментарий]

В таблице 3.1 приведены примеры записи директив ассемблера tasm.

Таблица 3.1. Примеры записи директив ассемблера tasm

| Директива | Пример записи | | | |
|---|---|---|---|---|
| ORG | | ORG | 100H | ;Программа начинается с адреса 100H |
| EQU | ALFA | EQU | 13H | ;Константе с именем ALFA<br>;присвоено значение13H |
| DB | K | DB | 60H | ;Для переменной с именем K<br>;резервируется ячейка (байт) памяти,<br>;в которую заносится 60H |
| DW | X | DW | 160H | ;Для переменной с именем X<br>;резервируются 2 ячейки (слово)<br>;памяти, в которые заносятся 60H<br>;и 01H |
| | BUF | DW | 8 DUP(?) | ;Для переменной BUF резервируется<br>;8 слов памяти |
| | TABL | DB | ‘Диск’ | ;Для переменной TABL<br>;резервируются ячейки памяти, в<br>;которые заносится последователь-<br>ность ASCII-символов «Диск» |
| PROC | TUR | PROC | | ;Начало |
| ENDP | TUR | ENDP | | ;и конец процедуры с именем TUR |
| .CODE | | | | ;Упрощенные сегментные директивы<br>;сегмента кодов, |
| .DATA | | | | ;сегмента данных, |
| .STACK | .STACK | 100H | | ;сегмента стека размером 100H байт |
| .MODEL | .MODEL | tiny | | ;Модель памяти размером 64K байт |
| END | | END | | ;Конец текста программы |

## 3.3. Примеры программ на языке ассемблера МП X86 для генерирования исполняемых модулей типа .COM

Далее приведены примеры программ на языке ассемблера микропроцессора X86 для генерирования на их основе исполняемых модулей с расширением .COM.

В приведенных программах директива .MODEL tiny задает модель памяти как один сегмент размером 64 Кбайт. Для задания сегмента кодов используется упрощенная сегментная директива .CODE. При генерации исполняемого модуля типа .COM все данные в памяти задаются в кодовом сегменте. Для указания начального адреса данных используется директива ORG. Так, директива ORG 500h размещает данные с адреса 500h. Операнды-байты или слова задаются после ORG с помощью директив DB или DW.

Директива END указывает на конец программы.

**Пример 3.1**

```
;Программа вычисления выражения  M=K+N–R +120
;для исполняемого модуля типа .COM.
;Операнды-слова k, r, m размещены в памяти с адреса 500h
;n – в регистре DX
        .MODEL    tiny          ;Программа и данные размещены
                                ;в сегменте кодов размером 64 Кбайт
        .CODE                   ;Начало кодового сегмента
;Размещение данных с адреса 500h
        org       500h
k       dw        +60h
r       dw        -10h
m       dw         0
;Команды программы с адреса 100h
        org       100h
        begin:                  ;Метка начала программы
        sub       dx,r          ;Вычисление
        add       dx,k          ;выражения
        add       dx,120h       ;m=(n-r)+k+120
        mov       m,dx          ;Загрузить результат m в память
        mov       ah,4ch        ;Функция DOS выхода
        int       21h           ;из программы
        END       begin         ;Конец прогаммы
```

**Пример 3.2**

```
;Программа вычисления выражения M=K+N–R+120 для случая
;базовой адресации данных, когда K, R, N, M размещены в памяти
;с адреса 500h
        .MODEL    tiny          ;Программа и данные размещены в
                                ;сегменте кодов размером 64 Кбайт
        .CODE                   ;Начало кодового сегмента
;Размещение данных с адреса 500h
        org       500h
k       dw        +60h
r       dw        -10h
n       dw        +30h
m       dw         0
;Команды программы с адреса 100h
        org       100h
        begin:                  ;Метка начала программы
        mov       bx,offset k   ;Загрузить в BX адрес операнда k
```

```
mov     ax,[bx]     ;Загрузить в AX операнд k
sub     ax,[bx+2]   ;Получить в AX разность k–r
add     ax,[bx+4]   ;AX←AX + n
add     ax,120h     ;AX←AX + 120h
mov     [bx+6],ax   ;Загрузить результат m в память
mov     ah,4ch      ;Функция DOS выхода
int     21h         ;из программы
END     begin       ;Конец программы
```

### 3.4 Ассемблер tasm. Трансляция исходного модуля

Ассемблер tasm – системная программа ПЭВМ, осуществляющая преобразование исходной программы на языке ассемблера МП86 в объектную программу.

Ассемблер вызывается командой

tasm [опции] <имя файла>.asm, , ↵

Если в команде будет указана одна запятая, то файл листинга не формируется.

Перечень опций может быть вызван командой tasm↵ Отметим наиболее используемые опции:

/zi – включает в объектный модуль информацию для отладки,

/n – подавляет вывод таблицы символов в листинге.

При выполнении работ вы изучите назначение используемых опций.

### 3.5. Порядок выполнения работы

1. Разработайте на языке ассемблера МП i8086 программу для вычисления заданного выражения (табл. 1.3) для случая прямой адресации данных (см. пример 3.1).

Разработанную на языке ассемблера программу введите в машину и выполните. Порядок ввода программы, создания исполняемого модуля и его выполнения (отладки) с помощью системных средств ПЭВМ типа IBM PC приведен ниже.

**Создание исходного файла программы**

2. Создайте в редакторе системы NORTON COMMANDER исходный файл программы на языке ассемблера (см. пп. *2-7* работы 1). Не забудьте имени файла дать расширение ASM, например PETROV. ASM.

**Ассемблирование**

3. Командой Ctrl-O уберите панели NC с экрана.

4. С помощью команды

tasm <имя файла>.asm, ,↵

запустите ассемблер tasm.

Сформированные в результате ассемблирования объектный модуль программы и ее листинг записываются на диск соответственно в виде

файлов с расширением OBJ и LST. В конце трансляции выдаются сообщения о возможных ошибках в исходной программе или об их отсутствии. Только при отсутствии ошибок сформированный объектный модуль может быть использован далее для создания редактором связи (компоновщиком) TLINK исполняемого модуля. При наличии ошибок необходимо осуществить просмотр листинга программы.

5. Командой Ctrl-O выведите панели NC на экран.

6. Выделите в панели NC файл с расширением LST и нажмите Alt-F3 (или F4). На экране изобразится листинг программы с сообщениями об ошибках. С помощью клавиш перемещения курсора просмотрите весь листинг и зафиксируйте в программе ошибки и их тип.

7. Нажав F10, выйдите из режима просмотра.

8. Для исправления ошибок в исходной программе перейдите в режим редактирования п. 2. Затем проведите ассемблирование скорректированной программы, для чего снова выполните пп. 3-5 (командную строку tasm <имя файла>.asm, , ↵ можно восстановить командой Ctrl-E).

**Создание исполняемого модуля**

9. Вызовите компоновщик TLINK, для чего введите команду

tlink /t/x <имя файла>.obj ↵

С помощью опции t задано компоновщику сформировать .COM-файл, а x – не генерировать .MAP-файл исполняемой программы. После выполнения tlink будет создан исполняемый модуль типа .COM, который может быть использован далее для его выполнения (отладки).

**Выполнение и отладка программы**

10. С помощью команды

td <имя файла>.com↵

запустите отладчик TD для работы с исполняемым модулем исходной программы. Порядок отладки программы с помощью окна CPU приведен в работе 1.

11. Удалите все ненужные файлы программы, кроме файла с расширением .asm. Удаление выделенного файла производится в NC нажатием клавиши F8, а затем ↵.

12. Разработайте программу вычисления заданного выражения для случая индексной адресации данных (четные варианты) или косвенной адресации (нечетные варианты заданий). Пример программы для режима базовой адресации приведен в п. 3.3.

13. По аналогии с пп. 2-11 введите в ЭВМ разработанную программу для заданного режима адресации и проверьте правильность ее выполнения.

Чтобы ускорить процесс разработки исходного файла программы, целесообразно за его основу взять файл предыдущей программы, который хранится на диске с расширением asm. Для этого выделите в панели имя этого файла и нажмите F4 – вы войдете в редактор с этим файлом, напри-

мер PROGR. Затем с помощью команд редактора удалите в сегменте команд ненужные строки и введите команды программы.

### 3.6. Содержание отчета

1. Программы вычисления заданного выражения для различных режимов адресации.
2. Исходные данные и результаты вычислений.
3. Трассы программ.

## Лабораторная работа 4

## СОСТАВ СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПЭВМ РС XT/AT ДЛЯ РАЗРАБОТКИ И ОТЛАДКИ ПРИКЛАДНЫХ ПРОГРАММ МИКРОСИСТЕМ НА БАЗЕ МП X86

### 4.1. Цель работы

Целью работы являются изучение состава системного программного обеспечения ПЭВМ для разработки и отладки 16-битных программ на языке ассемблера МП X86 и приобретение навыков работы по созданию прикладных программ.

### 4.2. Состав системного программного обеспечения ПЭВМ для разработки программ на языке ассемблера МП X86

Персональные компьютеры с системой команд МП X86 (К1810ВМ86, INTEL 8086, 80286, 80386, 80486, Pentium) являются эффективным средством автоматизации процесса разработки и отладки прикладных программ микросистем, построенных на основе МП86. Реализация этапов разработки и отладки прикладных программ требует наличия на магнитном диске ПЭВМ специальных программ, образующих системное программное обеспечение (СПО). В данной работе рассматривается порядок работы с этими программами при создании прикладных программ (рис. 4.1).

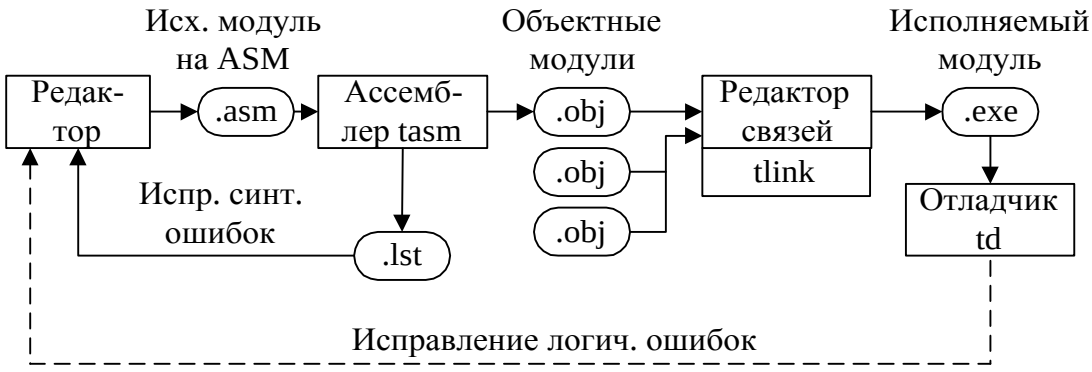


Рис. 4.1. Этапы разработки и отладки программ на языке ассемблера

В состав программного обеспечения для разработки программ на языке ассемблера МП X86 входят следующие системные программы.

1. Турбоассемблер TASM

Ассемблер преобразует исходную программу, еще ее называют исходным модулем типа .ASM, в переместимый объектный модуль типа OBJ. Объектный модуль наряду с кодами команд содержит служебную информацию, необходимую для формирования исполняемого модуля.

2. Редактор связей (компоновщик) TLINK

Объединяет созданные ассемблером объектные модули в единый исполняемый модуль типа .EXE для размещения его в основной памяти ПЭВМ для исполнения. Редактор связей запускается командой

tlink [опции] <объектные файлы> ↵

Отметим наиболее употребляемые опции:

/x – не создавать .MAP-файл с картой исполняемой программы;

/v – включить символьную информацию для отладчика;

/t – создать исполняемый .COM-файл.

3. Отладчик TD

Управляет процессом отладки программ и является инструментом для поиска ошибок в выполняемой программе.

### 4.3. Пример программы на языке ассемблера МП x86 для генерирования исполняемого модуля типа .EXE

В примере 4.1 приведен текст программы выдачи сообщения на экран. Исходный модуль представляет собой последовательность операторов (или предложений) языка ассемблера. Программа оформлена в виде процедуры PROC с именем OUR_PROG.

Директива .MODEL small задает модель памяти, при которой программа и данные размещаются в отдельных сегментах. Псевдооператоры .STACK, .DATA и .CODE делят программу на сегменты стека, данных и команд.

В сегменте данных оператор TEXT1 определяет текст сообщения, выводимого на экран, а SIMB – последовательность символов *, выделяющую этот текст.

Сегмент команд содержит 2 группы команд:

– первая группа устанавливает DS на сегмент данных,

– вторая группа – это команды программы вывода сообщений на экран.

В подпрограмме VDISP используется команда прерывания DOS INT 21H, с помощью которой производится обращение к функции для работы с дисплеем. Адрес типа функции задается значением регистра AH=9. Знак $ в тексте определяет конец изображаемой строки, начальный адрес которой задается в регистре DX командой MOV DX,OFFSET TEXT1. Коды

0Dh и 0Ah обеспечивают перевод маркера в начало следующей строки (0D – возврат каретки, 0A – переход на следующую строку).

**;Пример 4.1.** Программа вывода текста на экран

```
        .MODEL  small       ;Ближние модели программы и данных
        .STACK  64          ;Размер стека 64 байта
        .DATA               ;Начало сегмента данных
text1   db      'Иванова И.А.',0dh,0ah          ;Тексты
        db      'Петрова Н.В.',0dh,0ah,'$'      ;сооб-
simb    db      '******************', 0dh,0ah,'$'  ;щений
        .CODE               ;Начало кодового сегмента
our_prog    proc            ;Процедура с именем OUR_PROG
        mov     ax,@data    ;Указывает DS на
        mov     ds,ax       ;сегмент данных
;Команды программы
        mov     dx,offset text1     ;Адрес строки TEXT1 в DX
        call    vdisp               ;Вызов ПП вывода строки
        mov     dx,offset simb      ;Адрес строки SIMB в DX
        call    vdisp               ;Вызов ПП вывода строки
        mov     ah,4ch              ;Функция DOS
        int     21h                 ;выхода из программы
our_prog    endp                    ;Конец процедуры
;Подпрограмма вывода строки на экран
vdisp:  push    ax
        mov     ah,9                ;Функция DOS
        int     21h                 ;Вывода строки на экран
        pop     ax
        ret
        END             our_prog            ;Конец программы
```

### 4.4. Порядок выполнения работы

1. Изучите состав базового программного обеспечения.

2. Разработайте на языке ассемблера МП X86 программу вычисления заданного в работе 2 варианта выражения.

**Создание исходного файла программы**

3. Для создания нового файла, находясь в оболочке Norton Commander, с помощью клавиш Shift-F4 вызовите режим создания нового текстового файла, введите имя создаваемого файла: <имя файла>.asm и нажмите ↵. Если исходный файл уже существует, то вход в режим его редактирования осуществляется клавишей F4.

4. Редактирование. С помощью команд текстового редактора введите программу на языке ассемблера, приведенную в примере 4.1. Специально допустите 2-3 ошибки. Далее при трансляции программы ассемблер обнаружит ошибки, и вы их исправите.

**Создание объектного и исполняемого модулей**

5. Для создания исполняемого модуля типа .EXE следует вызвать и исполнить 2 команды:

5.1. Наберите в командной строке команду

tasm /zi <имя файла>.asm,, ↵

В случае отсутствия ошибок ассемблером tasm будет создан объектный модуль .OBJ. Если при трансляции ассемблер обнаружит ошибку, то на экран будет выведено сообщение об ошибке и номер строки, в которой она допущена.

5.2. При наличии .OBJ-модуля, наберите команду

tlink /v/x <имя файла>.obj ↵

При этом программой tlink будет сгенерирован исполняемый .EXE-модуль с символьной информацией для отладчика.

6. Исправьте с помощью текстового редактора ошибки и повторите пп. 3-5, пока система не выдаст сообщение, что в программе нет ошибок.

**Выполнение и отладка программы**

7. Для выполнения программы в отладчике вызовите команду:

td <имя файла>.exe ↵

После недолгой загрузки система перейдет в отладчик TD. На экране изобразятся 2 окна отладки: окно модуля Module с текстом программы и ниже – окно просмотра Watches. Далее приводится порядок отладки программы с помощью этих окон.

8. В окне модуля выделите с помощью маркера регистр AX, а затем нажмите Ctrl-W. При этом в окне просмотра изобразится значение переменной AX. Аналогичным образом просмотрите переменную DX.

9. Значение переменной можно проверить, затем модифицировать с помощью окна инспекции Inspecting. Сделайте это для переменной AX. Для этого маркером выделите в окне модуля регистр AX, а затем нажмите Ctrl-I. При этом в окне инспекции изобразится значение AX. Введите с клавиатуры новое значение AX, например 22. Нажав клавишу Esc, уберите окно проверки. Аналогичным образом можно просматривать и изменять переменные в памяти.

10. Порядок выполнения и перезагрузки программы, установки и отмены точек останова в окне модуля такой же, как и в окне CPU (см. работу 1). Последовательно нажимая F7, выполните программу по шагам. Следите при этом в окне просмотра за изменением переменных. После выполнения команд INT 21H и программы в целом с помощью клавиши Alt-F5 откройте (а затем закройте) экран пользователя, на котором изобра-

жаются видимые строки сообщений. Если при отладке выявлены логические ошибки, требующие изменения в программе, то с помощью команды Alt-X необходимо выйти из отладчика в редактор, внести изменения в исходную программу, затем создать исполняемый модуль и снова выполнить программу с помощью отладчика.

11. При необходимости с помощью команд F10; View; Dump↵ и F10; View; Registers ↵ можно вывести окна памяти и регистров. Это особенно важно для отладки программ с большим числом переменных, хранящихся в памяти и регистрах.

Клавишей Ctrl-F2 перезагрузите программу, а затем вызовите окна памяти и регистров. Не забудьте сместить эти окна на край экрана. Необходимо отметить, что начальный адрес в сегменте данных окна Dump устанавливают только после выполнения команд mov ax,@data и mov ds,ax.

Выполните программу по шагам, нажимая последовательно F8.

12. С помощь команды F10; View; CPU ↵ можно вызвать окно CPU, содержащее также и символьную информацию для отладки, и произвести выполнение программы в этом окне (см. работу 1). Сделайте это. Чтобы при отладке вам не мешали другие окна, увеличьте размер окна CPU или удалите ненужные окна.

13. Нажимая Alt-X, выйдите из отладчика.

14. Отлаженную программу можно запустить с помощью команды DOS. Для этого необходимо ввести команду

<имя файла>.exe ↵ ,

в которой указываются имя выполняемой команды и ее тип.

15. Изучив этапы создания и отладки прикладных программ, по аналогии с пп. 3-13 введите разработанную вами программу и выполните ее.

### 4.5. Содержание отчета

1. Исходная программа для заданного варианта.
2. Листинг программы.
3. Результаты выполнения программы.

**Лабораторная работа 5**

# РАЗРАБОТКА И ОТЛАДКА 32-РАЗРЯДНЫХ ПРИКЛАДНЫХ ПРОГРАММ НА БАЗЕ МП 80Х86

## 5.1. Цель работы

Целью работы является изучение этапов разработки и отладки программ на языке ассемблера 32-разрядных МП 80x86 (или X86) для реального режима их работы.

## 5.2. Особенности разработки 32-разрядных программ на языке ассемблера МП i80x86 для реального режима

Микропроцессоры i80386, i80486, Pentium имеют режимы: реального адреса (R-режима) и защищенного (P-режима).

Только в P-режиме МП i386, i486, Pentium выполняют 32-битные программы с использованием всех новых возможностей 32-разрядных МП (МП-32):

1) адресное пространство памяти — 4 Гбайт;

2) использование 32-битной адресации команд;

3) новые привилегированные команды для защищенного режима;

4) применение 32-битных РОНов МП (EAX, EBX, ECX и т.д.);

5) использование масштабированной индексной адресации, например: MOV EAX, [EBX+ESI*2+200h] — базовая индексная с масштабированием и смещением (может содержать до 15 байт);

6) использование режимов 32-битной адресации данных, включая и новые для МП-32 режимы, например, варианты индексной адресации с масштабированием.

Однако есть возможность использовать преимущества, указанные в пп. 4-6, если МП-32 работает в реальном режиме.

Единственный минус – все эти программы для R-режима работают только в сегментах емкостью 64 Кбайт и в памяти 1 Мбайт. Это объясняется тем, что старшие биты A31-A16 эффективного адреса, например [EBX] или [ESI], не изменяются.

Если используются 32-битные адреса и операнды, то команда (рис. 5.1) содержит дополнительные байты – атрибуты: 66h — размер операнда 32 бита; 67h — размер адреса 32 бита.

По умолчанию ассемблер генерирует коды для 16-разрядного МП i8086. Если он встретит команду из другого МП, то транслятор сгенерирует ошибку, что указывает на другой тип используемого МП. Поэтому необходимо указывать тип МП: **. 386 . 486**.

Для задания сегментов могут быть использованы директивы **SEGMENT –** начало сегмента и **ENDS –** конец сегмента.

| Префикс команды REP 0-1 байт | Префикс размера адреса 0-1 байт | Префикс размера операнда 0-1 байт | Префикс замены сегмента 0-1 байт |
|---|---|---|---|
| | 67h | 66h | |

| КОП 1 или 2 байта | mod reg r/m 0 или 1 байт | Масштаб 0 или 1 байт | Смещение в команде 0-2, 4 байта | Непосредств. операнд 0-2, 4 байта |
|---|---|---|---|---|
| | | Коэф. 1, 2, 4, 8 | | |

Рис. 5.1. Формат команды МП i80x86

На рис. 5.2 приведена схема формирования эффективного адреса при 32-битной адресации для команды mov EAX, [EBX+ESI*2+200h].

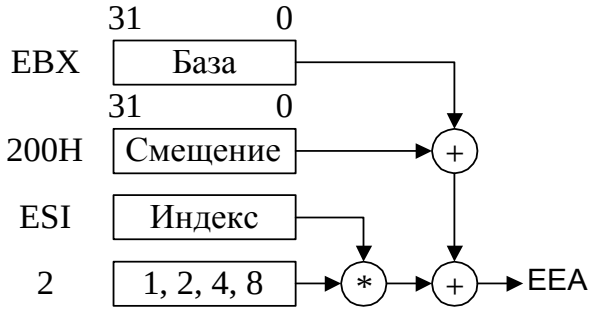


Рис. 5.2. Схема формирования адреса EEA=EBX+ESI*2+200h

Далее приведен пример 5.1 программы для R-режима с использованием 32-битных регистров и 32-битных режимов адресации данных.

**Пример 5.1.** Программа на языке ассемблера МП i486 с использованием 32-битных регистров и режимов адресации данных для R-режима

```
.486
stack1      segment     stack       use16       ‘stack’
            db          64          dup(0)      ;Стек 64 байта
stack1      ends
dseg        segment     para        use16       ‘data’
            k           dd          +60h        ;Переменные k
            r           dd          -10h        ; r
            n           dd          +30h        ; n
            m           dd          0           ;Результат m
dseg        ends
cseg        segment     para        use16       ‘code’
            assume      cs:cseg,ds:dseg,ss:stack1
begin proc  far                                 ;Дальняя процедура
            push        ds
```

```
                sub         eax,eax
                push        ax
                mov         ax,dseg          ;Указание регистру ds
                mov         ds,ax            ; на сегмент данных
    ;Команды программы
                mov         ebx,offset k
                mov         esi,2
                mov         eax,[ebx]        ;Вычисление
                sub         eax,[ebx+esi*2]  ; выражения
                add         eax,[ebx+esi*4]  ; m=
                add         eax,120h         ; = n-r+k+120
                mov         [ebx+8+4],eax
                mov         ah,4ch
                int         21h
    begin       endp
    cseg        ends
                end         begin
```

В программе применяются индексная с масштабированием адресация данных (см. рис. 5.2) и стандартные директивы ассемблера. В частности, директива USE16 используется для R-режима, при котором МП работает в сегментах емкостью 64 Кбайт, но с использованием 32-битных регистров и режимов адресации данных, хотя адреса команд 16-битные.

### 5.3. Порядок выполнения работы

**Порядок формирования EXE-файла разработанной 32-битной программы с отладочной информацией**

1. Для созданного исходного файла с расширением .asm выполните последовательность программ:

tasm /zi <имя файла>.asm,,

tlink /3/v <имя файла>.obj

td <имя файла>.exe

**Порядок работы в отладчике td.exe:**

2. Войдите в окно регистров

F10, VIEW, REGISTER [ENTER];

3. Находясь в окне регистров, вызовите локальное меню (Alt-F10) и установите длину регистров 32 бита

REGISTER 32 bit | YES |

4. Установите длину данных 4 байта:

F10, VIEW, DUMP, Ctrl-D, LONG

5. Последовательно, нажимая F8, выполните все команды до mov ds, ax (включительно).

6. В окне данных установите начальный адрес данных

Alt-F10, GOTO, DS:0 [ENTER]

7. Выполните далее программу по шагам (F8), наблюдая за результатами.

8. Просмотрите кодировку программы: F10, CPU.

9. Разработайте на языке ассемблера МП X86 32-разрядную программу вычисления заданного в работе 1 варианта выражения. Принять длину операндов 32 бита. Пусть все операнды размещены в памяти. По аналогии с пп. 1-8 произведите отладку программы.

### 5.4. Содержание отчета

1. Исходная программа для заданного варианта.
2. Листинг программы.
3. Результаты выполнения программы.

## Лабораторная работа № 6

## КОМАНДЫ ОБРАБОТКИ ЦЕПОЧЕК МП X86

### 6.1. Цель работы

Изучение цепочечных команд МП X86 и их использования в программах обработки цепочек (строк) данных.

### 6.2. Цепочечные команды

Под цепочкой понимается последовательность байт или слов, находящихся в смежных ячейках памяти. Такую связанную структуру данных образует, например, последовательность вводимых с терминала символов.

В МП X86 имеется пять однобайтных цепочечных команд-примитивов (см. Приложение), предназначенных для обработки одного элемента цепочки. Если элементом является байт, то в мнемонике команд указывается буква B, если слово, то – W. Например: MOVSB и MOVSW. В общем случае при обработке строк данных:

- цепочка-приемник находится в дополнительном сегменте данных и ее элемент адресуется регистром DI,
- цепочка-источник находится в сегменте данных и ее элемент адресуется регистром SI.

Задание начальных адресов цепочек приемника STRING1 и источника STRING2 производится командами LEA DI,ES:STRING1 и LEA SI,STRING2. Однако обе цепочки можно разместить в сегменте данных, если перед их обработкой сделать равными значения сегментных регистров ES и DS.

В зависимости от состояния признака направления DF регистра F после каждой операции над элементом цепочки выполняется инкремент

(DF=0) или декремент (DF=1) указателей DI и SI (при обработке байт – на единицу, при обработке слов – на два).

Цепочечной команде может предшествовать специальный префикс повторения REP, который заставит МП повторить команду и модифицировать CX←CX+1. Операции повторяются, пока CX≠0. Перед командами сравнения CMPS и SCAS мнемоника префикса имеет вид REPZ/REPE или REPNZ/REPNE. Для REPZ примитив повторяется, покаCX≠0 или признак ZF=1, а для REPNZ – пока CX≠0 или ZF=0.

Программы, приведенные в примерах 6.1 и 6.2 выполняют одно и то же действие – сравнивают строки, пока не будут просмотрены 100 элементов или пока не встретятся совпадающие элементы.

**Пример 6.1.** Найти совпадения

```
mm1: cmpsb          ;Сравнить [SI]–[DI], SI←SI+1, DI←DI+1
     dec     cx     ; CX←CX-1
     jnz     mm1    ;Повторять, покаCX≠0 или FZ=0
```

**Пример 6.2.** Найти совпадения.

```
repnz     cmpsb     ;Те же операции, что и в примере 6.1.
```

Используя в примере 6.1 вместо JNZ другие команды условного перехода, можно сравнивать элементы и по другим признакам, например по признаку “больше” или “меньше”.

В примере 6.3 показана процедура копирования 10 элементов из строки STRING2 в строку STRING1.

**Пример 6.3.** Копирование элементов строки

```
;Пусть цепочки находятся в сегменте данных
string2  db  ‘Петрова Н., Иванов А.’ ;Цепочка-источник
string1  db 10 dup(?)                ;Цепочка-приемник
;Процедура копирования
copir  proc
       push  ds          ;Заставить  указывать
       pop   es          ;на сегмент данных
       cld               ;Сброс  для обработки слева направо
       lea   si,string2  ;Адрес цепочки STRING2 в SI
       lea   di,string1  ;Адрес цепочки STRING1 в DI
       mov   cx,10       ;Число элементов в CX
rep    movsb             ;Скопировать байты
       ret
copir  endp
```

### 6.3. Ввод строковых данных в ПЭВМ

Значения элементов в строке можно задать с помощью директивы DB или DW.

При вводе строки с клавиатуры целесообразно пользоваться функцией DOS 0AH, вызываемой командой INT 21H. Чтобы воспользоваться ей, необходимо зарезервировать в сегменте данных место для строки. Например, оператор

```
string      db      64,65 dup(?)
```

резервирует место для строки STRING, содержащей 64 элемента.

Чтение строки с клавиатуры в память без изображения на экране производится командами, приведенными в примере 6.4.

**Пример 6.4.** Чтение строки с клавиатуры

```
lea     dx, string    ;Сделать  DX указателем буфера
mov     ah,0ah        ;Прочитать
int     21h           ;строку. Ввод завершить нажатием <ВК>
```

После их выполнения в 1-м байте буфера STRING находится длина буфера 64, во 2-м – число фактически введенных символов. Элементы строки STRING+2 размещаются с 3-го байта.

### 6.4. Варианты заданий

1. Скопировать строку в обратном порядке.
2. Найти строку (список) с ключевым словом.
3. Найти в строке символ и заменить его.
4. Найти в строке символ «точка» и после нее заменить все элементы пробелами.
5. Найти в строке справа налево первый символ «;». Элементы, размещенные справа от ; , заменить пробелами.
6. Найти две одинаковые строки и одну из них заменить новой.
7. Составить из 3-х строк (слов) предложение.
8. Подсчитать число элементов в строке до символа ВК.
9. Определить в строке число полей, разделенных пробелами. Найти в строке комбинацию двух символов и заменить их.

### 6.5. Порядок выполнения работы

1. Изучите форматы цепочечных команд и примеры программ обработки строк (пп. 6.2 - 6.3).
2. Разработайте для заданного варианта (п. 6.4.) программу обработки строк. Для более наглядного выполнения программы используйте процедуры ввода и изображения строк, приведенные в примере 6.4.
3. С помощью системных программ (см. работу 4) сформируйте исходный объектный и исполняемый модули.
4. Выполните программы в отладчике TD.
5. Выполните программу в DOS.

### 6.5. Содержание отчета

1. Программа для заданного варианта.
2. Результаты выполнения прграммы.

## Библиографический список

1. Юров, В.И. Assembler [Текст]/ В.И. Юров.- Учебник для вузов.- 2-е издание.- СПб.: Питер, 2006.- 637 с.: ил.- ISBN: 5-94723-581-1

2. Юров, В.И. Assembler. Практика [Текст]/ В.И. Юров.- Учебник для вузов.- 2-е издание.- СПб.- Питер, 2006.- 399 с.: ил.- ISBN: 5-94723-671-0

3. Абель, П. Язык ассемблера для IBM PC и программирования [Текст]/П. Абель/ Пер. с англ. Ю.В. Сальникова.- М.: Высшая школа, 1992.-447 с., ил.

4. Пирогов, П.Ю. ASSEMBLER. Учебный курс [Текст]/ П.Ю. Пирогов.- М.: Издатель Молгачева С.В.- Нолидж, 2001.- 848 с.- ил.- ISBN: 5-89251-101-4

5. Микропроцессоры Intel 80x86. Архитектура и программирование: Методические указания к лабораторным работам / Рязан. гос. радиотехн. акад.; Сост. В.Н. Локтюхин. Рязань, 2007. 68 с.

## СОДЕРЖАНИЕ

**Приложение**

# СИСТЕМА КОМАНД МП Intel 8086

При описании команд используются следующие обозначения:

reg - один из РОН в соответствии со следующей таблицей:

| reg | W=1 | W=0 | reg | W=1 | W=0 | sreg | Сегм. рег. |
|---|---|---|---|---|---|---|---|
| 000 | AX | AL | 001 | CX | CL | 00 | ES |
| 010 | DX | DL | 011 | BX | BL | 01 | CS |
| 100 | SP | AH | 101 | BP | CH | 10 | SS |
| 110 | SI | DH | 111 | DI | BH | 11 | DS |

D - если D=l, то в reg, если D=0, то из reg;

W - если W=l, то команда оперирует словом, - W=0, то - байтом;

X0- один байт данных.

S:W = 01 - два байта данных,

11 - один байт данных, расширенный со знаком до 16 бит;

V - если V=0, то счетчик равен 1, V=l - счетчик в CL;

Z - применяется в цепочечных командах для сравнения с ZF;

data L, data H - младшая и старшая часть (если W=l) непосредственного операнда;

mod - поле режима адресации;

r/m - при mod=11 интерпретируется как reg,

при mod≠11 определяет эффективный адрес (EA) операнда в памяти в соответствии со следующей таблицей, в которой disp L, disp H,L - 8- и 16-битные смещения в 3-м или в (3-4)-м байтах команды:

| mod / r/m | 00 | 01 | 10 |
|---|---|---|---|
| 000 | (BX)+(SI) | (BX)+(SI)+disp L | (BX)+(SI)+disp H,L |
| 001 | (BX)+(DI) | (BX)+(DI)+disp L | (BX)+(DI)+disp H,L |
| 010 | (BP)+(SI) | (BP)+(SI)+disp L | (BP)+(SI)+disp H,L |
| 011 | (BP)+(DI) | (BP)+(DI)+disp L | (BP)+(DI)+disp H,L |
| 100 | (SI) | (SI)+disp L | (SI)+disp H,L |
| 101 | (DI) | (DI)+disp L | (DI)+disp H,L |
| 110 | disp H,L | (BP)+disp L | (BP)+disp H,L |
| 111 | (BX) | (BX)+disp L | (BX)+disp H,L |

Команды передачи данных

| Команды | 1-й байт | 2-й байт | 3-й байт | 4-й байт |
|---|---|---|---|---|
| MOV – передать | | | | |
| Регистр или память в/из регистра | 100010DW | mod reg r/m | | |
| Непосредственный операнд в регистр или память | 1100011W | mod 000 r/m | data L | data H (W=1) |

| | | | | |
|---|---|---|---|---|
| Непосредственный операнд в регистр (спецформат) | 1011Wreg | data L | data H, (W=1) | |
| Память в аккумулятор (спецформат) | 1010000W | Мл. адрес EA | ст. адрес EA | |
| Аккумулятор в память (спецформат) | 1010001W | Мл. адрес EA | ст. адрес EA | |
| Регистр или память в регистр сегмента | 10001110 | mod 0 sreg r/m | | |
| Регистр сегмента в регистр или память | 10001100 | mod 0 sreg r/m | | |
| PUSH – включить в стек | | | | |
| Регистр/память | 11111111 | mod 110 r/m | | |
| Регистр (спецформат) | 01010reg | | | |
| Регистр сегмента | 000sreg110 | | | |
| POP – исключить | | | | |
| Регистр/память из стека | 10001111 | mod 000 r/m | | |
| Регистр (спецформат) | 01011reg | | | |
| Регистр сегмента | 000sreg111 | | | |
| XCHG – обменять | | | | |
| Регистр/память с регистром | 1000011W | mod reg r/m | | |
| Регистр с аккумулятором | 10010reg | | | |
| IN – ввести из | | | | |
| Фиксированного порта | 1110010W | Порт | | |
| Переменного порта | 1110110W | | | |
| OUT – вывести в | | | | |
| Переменный порт | 1110111W | | | |
| Фиксированный порт | 1110110W | Порт | | |
| XLAT – передать байт в AL | 11010111 | | | |
| LEA – загрузить EA в регистр | 10001101 | mod reg r/m | | |
| LDS–загрузить указатель в DS | 11000101 | mod reg r/m | | |
| LES–загрузить указатель в ES | 11000100 | mod reg r/m | | |
| LAHF – загрузить в AH признаки | 10011111 | | | |
| SAHF – запомнить AH в регистре признаков | 10011110 | | | |
| PUSHF – включить признаки | 10011100 | | | |
| POPF – исключить признаки | 10011101 | | | |

Команды арифметических операций

| Команды | 1-й байт | 2-й байт | 3-й байт | 4-й байт |
|---|---|---|---|---|
| ADD – сложить | | | | |
| Регистр/память с регистром | 000000DW | mod reg r/m | | |

| | | | | |
|---|---|---|---|---|
| Непосредственный операнд с регистром/памятью | 100000SW | mod 000 r/m | data L | data H (SW=01) |
| Непосредственный операнд с аккумулятором | 0000010W | data L | data H, (w=1) | |
| ADC – сложить с переносом | | | | |
| Регистр/память с регистром | 000100DW | mod reg r/m | | |
| Непосредственный операнд с регистром/памятью | 100000SW | mod 010 r/m | data L | data H (SW=01) |
| Непосредственный операнд с аккумулятором | 0001010W | data L | data H, (w=1) | |
| INC – инкремент | | | | |
| Регистра/памяти | 1111111W | mod 000 r/m | | |
| Регистра | 01000reg | | | |
| AAA – ASCII – коррекция сложения | 00110111 | | | |
| DAA – десятичная коррекция сложения | 00100111 | | | |
| SUB – вычесть | | | | |
| Регистр/память с регистром | 001010DW | mod reg r/m | | |
| Непосредственный операнд из регистра/памяти | 100000SW | mod 101 r/m | data L | data H (SW=01) |
| Непосредственный операнд из аккумулятора | 0010110W | data L | data H, (w=1) | |
| SBB – вычесть с заемом | | | | |
| Регистр/память с регистром | 000110DW | mod reg r/m | | |
| Непосредственный операнд из регистра/памяти | 100000SW | mod 011 r/m | data L | data H (SW=01) |
| Непосредственный операнд из аккумулятора | 0010110W | data L | data H, (w=1) | |
| DEC – декремент | | | | |
| Регистр/память | 1111111W | mod 001 r/m | | |
| Регистр | 01001reg | | | |
| NEG – изменить знак | 1111011W | mod 011 r/m | | |
| CMP - сравнить | | | | |
| Регистр/память и регистр | 001110DW | mod reg r/m | | |
| Непосредственный операнд с регистром/памятью | 100000SW | mod 111 r/m | data L | data H (SW=01) |
| Непосредственный операнд с аккумулятором | 0011110W | data L | data H, (w=1) | |
| AAS – ASCII коррекция вычитания | 00111111 | | | |
| DAS – десятичная коррекция вычитания | 00101111 | | | |
| IMUL – умножение целых со знаком | 1111011W | mod 101 r/m | | |

| | | | | |
|---|---|---|---|---|
| MUL – умножение без знака | 1111011W | mod 100 r/m | | |
| AAM – ASCII – коррекция умножения | 11010100 | 00 001 010 | | |
| DIV – деление без знака | 1111011W | mod 110 r/m | | |
| IDIV – деление со знаком | 1111011W | mod 111 r/m | | |
| AAD – ASCII коррекция деления | 11010101 | 00 001 010 | | |
| CBW – преобразование байта в слово | 10011000 | | | |
| CWD – преобразование слова в двойное слово | 10011001 | | | |

Команды логических операций и сдвигов

| Команды | 1-й байт | 2-й байт | 3-й байт | 4-й байт |
|---|---|---|---|---|
| NOT – инвертирование | 1111011W | mod 010 r/m | | |
| SHL/SAL – сдвиг логический/арифм. влево | 110100VW | mod 100 r/m | | |
| SHR – сдвиг логич. вправо | 110100VW | mod 101 r/m | | |
| SAR – сдвиг арифм. вправо | 110100VW | mod 111 r/m | | |
| ROL – сдвиг цикл. влево | 110100VW | mod 000 r/m | | |
| ROR – сдвиг цикл. вправо | 110100VW | mod 001 r/m | | |
| RCL – сдвиг цикл. влево | 110100VW | mod 010 r/m | | |
| RCR – сдвиг цикл. вправо через перенос | 110100VW | mod 011 r/m | | |
| AND – коньюнкция И | | | | |
| Регистр/память с регистром | 001000DW | mod reg r/m | | |
| Непосредственный операнд с регистром/памятью | 1000000W | mod 100 r/m | data L | data H (W=1) |
| Непосредственный операнд с аккумулятором | 0010010W | data L | data H (W=1) | |
| TEST – И без записи результ. | | | | |
| Непосред. операнд с аккум. | 1010100DW | data L | data H | |
| Регистр/память с регистром | 1000010W | mod reg r/m | | |
| Непосредственный операнд с регистром/памятью | 1111011W | mod 000 r/m | data L | data H (W=1) |
| OR – дизьюнкция ИЛИ | | | | |
| Регистр/память с регистром | 000010DW | mod reg r/m | | |
| Непосредственный операнд с регистром/памятью | 1000000W | mod 001 r/m | | |
| Непосредственный операнд с аккумулятором | 0000110W | data L | data H (W=1) | |
| XOR – ИЛИ исключающее | | | | |
| Регистр/память с регистром | 001100DW | mod reg r/m | | |
| Непосредственный операнд с регистром/памятью | 1000000W | mod 001 r/m | data L | data H (W=1) |
| Непосредственный операнд с | 0011010W | data L | data H | |

| | | | | |
|---|---|---|---|---|
| аккумулятором | | | (W=1) | |

Команды операций с цепочками данных

| Команды | 1-й байт |
|---|---|
| REP - повторить | 1111001Z |
| MOVS – передать байт/слово: [DI] ← [SI] | 1010010W |
| CMPS – сравнить байт/слово: [DI] ← [SI] | 1010011W |
| SCAS – сканировать байт/слово: acc ←[DI] | 1010111W |
| LODS – загрузить байт/слово из [SI] в AL/AX | 1010110W |
| STOS – запомнить байт/слово из AL/AX в[DI] | 1010101W |

Команды передачи управления (безусловные)

| Команды | 1-й байт | 2-й байт | 3-й байт |
|---|---|---|---|
| CALL – ВЫЗОВ | | | |
| Прямой в сегменте | 11101000 | Мл. смещ. | Ст. смещ. |
| Косвенный в сегменте | 11111111 | mod 010 r/m | |
| Прямой межсегментный | 10011010 | Мл. смещ.<br>Мл. сег. (4-й байт) | Ст. смещ.<br>Мл.сег.(5-й байт) |
| Косвенный межсегментный (mod ≠ 11) | 11111111 | mod 011 r/m | |
| JMP – БЕЗУСЛ. ПЕРЕХОД | | | |
| Прямой в сегменте | 11101001 | Мл. смещ. | Ст. смещ. |
| Прямой в сегменте короткий | 11101011 | Смещение | |
| Косвенный в сегменте | 11111111 | mod 100 r/m | |
| Прямой межсегментный | 11101010 | Мл. смещ.<br>Мл. сег.(4-й байт) | Ст. смещ.<br>Мл.сег.(5-й байт) |
| Косвенный межсегментный (mod ≠ 11) | 11111111 | mod 101 r/m | |
| RET – ВОЗВРАТ | | | |
| В сегменте со слож. непосредст. операнда с SP | 11000010 | data L | data H |
| В сегменте | 11000010 | | |
| Межсегментный | 11001011 | | |
| Межсегментный со слож. непосредст. операнда с SP | 11001010 | data L | data H |

Команды условных переходов и циклов

| Команды | Условие | | | | |
|---|---|---|---|---|---|
| JA/JNBE | CF ∪ ZF=0 | > | 77 | Смещ. | Перейти, если выше* /не ниже или равно |
| JNC/JAE/JNB | CF=0 | >= | 73 | Смещ. | Выше или равно/ не ниже (нет переноса) |
| JC/JB/JNAE | CF=1 | < | 72 | Смещ. | Ниже/не выше или равно (есть перенос) |
| JBE/JNA | CF ∪ ZF=1 | <= | 76 | Смещ. | Ниже или равно/не выше |

| | | | | | |
|---|---|---|---|---|---|
| JE/JZ | ZF=1 | = | 74 | Смещ. | Равно/нуль |
| JG/JNLE | (SF+OF)ZF=0 | > | 7F | Смещ. | Больше* /не меньше и = |
| JGE/JNL | SF+OF=0 | >= | 7D | Смещ. | Больше или равно/не меньше |
| JL/JNGE | SF+OF=1 | < | 7C | Смещ. | Меньше/не больше или равно |
| JLE/JNG | (SF+OF)ZF=1 | <= | 7E | Смещ. | Меньше или равно/не равно |
| JNZ/JNE | ZF=0 | | 75 | Смещ. | Не равно/не нуль |
| JNO | OF=0 | | 71 | Смещ. | Нет переполнения |
| JNP/JPO | PF=0 | | 7B | Смещ. | Нет паритета/паритет нечетный |
| JNS | SF=0 | | 79 | Смещ. | Нет знака (отрицание) |
| JO | OF=1 | | 70 | Смещ. | Есть переполнение |
| JP/JPE | PF=1 | | 7A | Смещ. | Есть паритет/паритет четный |
| JS | SF=1 | | 78 | Смещ. | Есть знак (отрицательный) |
| JCNX | (CX)=0 | | E3 | Смещ. | Содержение регистра CX=0 |
| LOOP | (CX) ≠ 0 | | E2 | Смещ. | Зациклить CX раз |
| LOOPZ/LOOPE | (CX)≠0, ZF=1 | | E1 | Смещ. | Зациклить до нуля/равно |
| LOOPNZ/LOOPNE | (CX)≠0, ZF=0 | | E0 | Смещ. | Зацикл. до нуля/не равно |

* Термины “больше”, “меньше” относятся к знаковым числам, представленным в ДК, а “выше”, “ниже” – к беззнаковым.

Команды прерывания

| Команды | 1-й байт | 2-й байт |
|---|---|---|
| INT - прерывание | 11001101 | Тип |
| INT 3 – прерывание типа 3 | 11001100 | |
| INTO – прерывание по переп. | 11001110 | |
| IRET – возврат из прерывания | 11001111 | |

Команды управления микропроцессором

| Команды | 1-й байт | 2-й байт |
|---|---|---|
| CLC – сброс переноса | 11111000 | |
| CMC – дополнение переноса | 11110101 | |
| STC – установка переноса | 11111001 | |
| STD – установка направления | 11111101 | |
| CLD – сброс направления | 11111100 | |
| CLI – брос прерываний | 11111010 | |
| STI – установка прерываний | 11111011 | |
| HLT – останов | 11110100 | |
| WAIT – ожидание | 10011011 | |
| ESC – обращение к сопроцес. | 11011xxx | mod yyy r/m |
| LOCK – префикс блокировки | 11110000 | |

Префикс замены сегмента: 001sreg110

*Учебное издание*

**Чернов** Андрей Владимирович
**Тишина** Анджела Викторовна

# АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ

Учебно-методическое пособие
для выполнения лабораторных и практических работ

Часть 1

Микропроцессоры INTEL 80X86

«Электронный университет» ФГБОУ ВО РГУПС

---

Адрес университета:
344038, Ростов н/Д, пл. Ростовского Стрелкового Полка Народного Ополчения, 2.