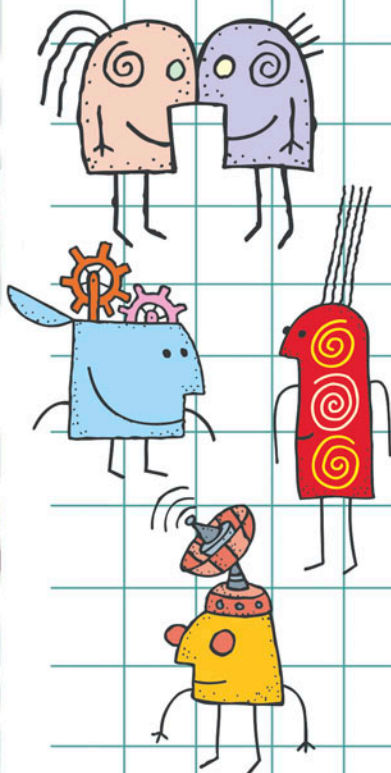


АССЕМБЛЕР



Основы программирования
на языке ассемблера

Интерфейс с языком C

Программирование
сопроцессора i80x87

Резидентные программы



Дискета содержит
архив пакета a86 v4.05,
исходные тексты программ
с рассматриваемыми примерами

Андрей Жуков
Андрей Авдюхин

Ассемблер

Санкт-Петербург
«БХВ-Петербург»

2002

УДК 681.3.068+800.92Ассемблер
ББК 32.973.26-018.1
Ж86

Жуков А. В., Авдюхин А. А.

Ж86 Ассемблер. — СПб.: БХВ-Петербург, 2002. — 448 с.: ил.
ISBN 5-94157-133-X

Книга является руководством по программированию на ассемблере для микропроцессорных систем на базе i80x86 и посвящена практическому применению этого языка на примере и с использованием ассемблера a86. Рассматриваются дополнительные возможности языков ассемблера: макрокоманды и связь с языками высокого уровня. Приводится обзор стилей языков ассемблера для разных вычислительных систем. В качестве иллюстрации применения ассемблеров рассмотрены различные вопросы, связанные с многозадачностью, — обработка прерываний и резидентные программы. Изложенный материал снабжен примерами, контрольными вопросами и заданиями к практическим работам.

Для начинающих программистов

УДК 681.3.068+800.92Ассемблер
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Анатолий Адаменко</i>
Зав. редакцией	<i>Анна Кузьмина</i>
Редактор	<i>Петр Науменко</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн обложки	<i>Игоря Цырульников</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 05.04.02.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 36,12.

Тираж 4000 экз. Заказ №

"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953 Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-133-X

© Жуков А. В., Авдюхин А. А., 2002
© Оформление, издательство "БХВ-Петербург", 2002

Содержание

Введение	13
Структура изложения	14
ЧАСТЬ I. ОСНОВЫ АССЕМБЛЕРА	17
Глава 1. Данные, имена, типы	19
1.1. Структура программы	19
1.2. Директивы определения данных	20
1.3. Обозначение чисел	22
1.4. Символические обозначения чисел, выражения	23
1.5. Переменные и метки	25
1.6. Типы имен	27
1.7. Типы и выражения	28
Глава 2. Определение имен	31
2.1. Алгоритм трансляции	31
2.2. Повторное определение имен	37
2.3. Локальные имена	37
2.4. Предопределенные имена	38
2.5. Имя <i>end</i>	38
Глава 3. Практикум по программированию данных	42
3.1. Запуск ассемблера <i>a86</i>	42
3.2. Программирование данных	44
Глава 4. Просмотр данных в отладчике	48
4.1. Запуск и завершение сеанса отладки	48
4.2. Экран отладчика	49
4.3. Окна отображения данных	50
4.4. Сохранение нажатий клавиш	51
4.5. Форматы вывода данных	51
4.5.1. Базовые форматы	52
4.5.2. Составные форматы	53
4.6. Задания на самостоятельную работу	54

Глава 5. Способы адресации	56
5.1. Данные процессора	56
5.2. Обозначения операндов машинных команд	58
5.3. Способы адресации операндов	59
5.3.1. Регистровая и непосредственная адресация	59
5.3.2. Адресация данных в памяти	60
Прямая адресация	60
Косвенная адресация	62
5.3.3. Ограничение на адресацию операндов в памяти	65
Глава 6. Система команд i80x86	66
6.1. Режим непосредственного выполнения в d86.....	66
6.2. Способы адресации операндов	67
6.2.1. Регистровая, непосредственная и прямая адресация.....	67
6.2.2. Косвенная адресация.....	68
Косвенная адресация по значению одного регистра.....	69
Косвенная адресация по сумме значений двух регистров.....	69
6.3. Обзор системы команд процессора i80x86	69
6.3.1. Команды пересылки.....	70
6.3.2. Арифметические команды	71
6.3.3. Логические команды	72
6.3.4. Команды сдвигов и вращений	72
6.3.5. Команды передачи управления.....	73
Адресация в командах передачи управления.....	74
Команды условных переходов	75
6.3.6. Воздействие команд на флаги	77
6.3.7. Строковые команды	80
Глава 7. Программирование циклов	83
7.1. Поиск в массиве байтов	83
7.2. Поиск в массиве слов	84
7.3. Поиск байта со значением больше заданного	86
7.4. Подсчет байтов в заданном диапазоне значений	87
7.4.1. Алгоритмическое решение.....	87
7.4.2. Табличное решение	88
Глава 8. Исследование программ в d86.....	90
8.1. Пример исследуемой программы	90
8.2. Названия регистров в отображении данных	92
8.3. Режимы выполнения.....	92
8.4. Постоянные точки останова	94
8.5. Редактирование команд.....	94
8.6. Принудительный останов.....	94
Глава 9. Примеры программ.....	96
9.1. Обработка данных на уровне бит	96
9.2. Вложенные циклы.....	97

9.3. Программирование ввода/вывода.....	98
9.4. Проблема опережающих ссылок	101
9.5. Решение проблемы опережающих ссылок	102
9.6. Упаковка четырехбитных кодов.....	103
9.7. Задания на составление программ	105
9.7.1. Задания первого уровня сложности.....	105
9.7.2. Задания второго уровня сложности.....	107

Глава 10. Сегменты и ехе-программы 112

10.1. Сегментная модель памяти.....	112
10.2. Сегменты в com-программе	116
10.3. Сегменты в ехе-программе.....	117
10.4. Особенности подготовки ехе-программы.....	121
10.5. Построение ехе-программ из нескольких модулей	123
10.6. Практикум	125
10.6.1. Компоновка.....	125
10.6.2. Организация отладки.....	126
10.6.3. Компоновка многомодульной программы.....	127
10.6.4. Задание на самостоятельную работу.....	128

ЧАСТЬ II. РАСШИРЕННЫЕ ВОЗМОЖНОСТИ АССЕМБЛЕРА 129

Глава 11. Макрокоманды и условная трансляция 131

11.1. Макрокоманды.....	132
11.1.1. Макрокоманды без параметров.....	132
11.1.2. Макрокоманды с параметрами.....	133
11.1.3. Правила подстановки параметров.....	135
11.1.4. Функции от параметров	136
11.1.5. Циклы по параметрам	138
R-циклы	139
Q-циклы	139
Задание шага в r- и q- циклах	140
11.1.6. Цикл по литерам параметра.....	142
11.1.7. Циклы, не зависящие от параметров.....	143
11.1.8. Вложенные циклы	145
Вложенные циклы с фиксированным числом повторений.....	145
Сложная обработка фактических параметров.....	146
11.2. Средства условной трансляции.....	146
11.2.1. Директива <code>#if..#endif</code>	147
11.2.2. Макроопределения и условная трансляция	147
11.2.3. Условия	149
Арифметические отношения	149
Объединение условий.....	149
Проверка существования.....	150
Определение имен в командной строке	150
Инвертирование логических значений.....	151
Пример условной трансляции в макроопределениях.....	152
11.2.4. Условная трансляция в циклах.....	154

11.3. Метки в макроопределениях	155
11.4. Средства отладки макрокоманд	158
11.5. Практикум	159
11.5.1. Макрокоманды без циклов	159
11.5.2. Макрокоманды с циклами	160
11.5.3. Функции от параметров	161
11.5.4. Вложенные циклы	161
11.5.5. Условная трансляция в макроопределениях	162
11.6. Вызов макрокоманд в i86.....	163

Глава 12. Структурный ассемблер..... 164

12.1. Логика условных переходов	164
12.2. Оператор <i>if</i> в i86.....	166
12.3. Способы реализации структурного ассемблера.....	167
12.3.1. Реализация при помощи макроассемблера.....	167
12.3.2. Реализация при помощи препроцессора.....	167
12.4. Структурный ассемблер bsp86.....	168
12.4.1. Синтаксис логических операторов.....	168
Оператор цикла с постусловием.....	169
Оператор цикла с предусловием.....	169
Оператор счетного цикла	170
Оператор бесконечного цикла.....	170
Выход из цикла	171
Оператор условного выполнения	171
Последовательность операторов	172
Оператор мультиветвления.....	173
12.5. Реализация мультиветвления в bsp86	174
12.6. Выполнение примеров.....	177
12.7. Диагностические сообщения bsp86	179
12.7.1. Сообщения первого типа	179
12.7.2. Особенности лексического анализатора.....	180
12.7.3. Сообщения второго типа	180
12.7.4. Ошибки в списках вариантов.....	181
12.8. Пример использования bsp86	182
12.9. Практикум	183

Глава 13. Интерфейс с языком С..... 185

13.1. Машинное представление данных языка С.....	185
13.2. Правила использования регистров	186
13.2.1. Сегментные регистры	186
13.2.2. Регистры общего назначения	187
13.2.3. Регистр флагов	187
13.3. Ассемблерные вставки	187
13.3.1. Ассемблерные вставки в Microsoft/Borland-С.....	187
13.3.2. Ассемблерные вставки в Watcom-С.....	188
13.4. Интерфейс С-ассемблер при раздельной трансляции.....	191
13.4.1. Соглашения о сегментах кода	191
13.4.2. Соглашения об именах С-объектов	192

13.4.3. Соглашения о результате функции	194
13.4.4. Передача параметров через регистры	195
13.4.5. Передача параметров через стек	197
Передача при ближних вызовах	197
Кадр стека при дальних вызовах	199
Инструкции <i>enter</i> и <i>leave</i>	200
Порядок передачи параметров.....	202
Соглашения об удалении параметров.....	203
Сокращенные обозначения параметров и локальных данных	204
Параметры дальние указатели	205
Обращение к глобальным данным.....	206
13.5. Выполнение примеров.....	209
Глава 14. Обработка BCD-данных.....	212
14.1. Форматы BCD.....	212
14.2. Операции над упакованными BCD	213
14.2.1. Сложение и вычитание	213
14.2.2. Умножение и деление	216
14.2.3. Дополнительные возможности <i>aad</i> и <i>aam</i>	218
14.3. Операции над упакованными BCD	219
14.3.1. Инструкции <i>daa</i> и <i>das</i>	219
14.3.2. Преобразования для деления и умножения.....	220
14.4. Операции над знаковыми BCD	221
14.5. Команды, воздействующие на флаг <i>a</i>	221
14.6. Практикум	222
Глава 15. Математический сопроцессор	224
15.1. Проверка наличия FPU	225
15.2. Загрузка и выгрузка целых чисел.....	225
15.3. Недопустимые операции и NaN	228
15.4. Организация массива данных в виде стека	228
15.5. Вычисления в стековой машине.....	230
15.5.1. Двуместные операции	230
15.5.2. Выражения в обратной польской записи.....	231
15.5.3. Расширения стековой машины в i80x87.....	232
15.6. Представление данных в FPU.....	233
15.7. Стандартный формат вещественных данных	236
15.8. Программная модель i80x87	237
15.8.1. Флаги особых ситуаций	239
15.8.2. Битовые поля управляющего слова	239
15.8.3. Битовые поля слова состояния.....	240
15.8.4. Доступ к указателям инструкции и операнда.....	242
15.9. Операции i80387	243
15.9.1. Пересылки	243
Загрузка данных	243
Команды выгрузки.....	245
Команда обмена	247

15.9.2. Арифметические операции	248
Основные арифметические операции.....	248
Операции над знаковым битом	249
Округление до целого.....	250
Получение остатка от деления.....	251
Извлечение корня	252
Масштабирование	252
Операции сравнения и тестирования	253
15.9.3. Трансцендентные операции.....	254
Тригонометрические операции	254
Возведение в степень.....	255
15.9.4. Команды управления.....	258
15.10. Практикум	259

ЧАСТЬ III. УПРАВЛЕНИЕ ВНЕШНИМИ УСТРОЙСТВАМИ, ПРЕРЫВАНИЯ, РЕЗИДЕНТНЫЕ ПРОГРАММЫ 263

Глава 16. Управление внешними устройствами	265
16.1. Внешние устройства в программной модели вычислительной системы.....	265
16.2. Инструкции для доступа к портам	266
16.3. Исследование внешних устройств вручную	268
16.3.1. Запуск редактора портов	268
16.3.2. Определение наличия устройства	268
16.3.3. Доступ к регистрам устройства	269
Коммутация по чтению-записи.....	269
Управление коммутацией через отдельный порт	270
16.3.4. Управление устройствами	274
Принцип последовательной связи	274
Основные регистры адаптера последовательной связи.....	275
Последовательная передача в диагностическом режиме	276
Получение ошибки переполнения приемника	277
16.4. Управление устройствами по программе	277
16.4.1. Наблюдение за состоянием в режиме периодического опроса	278
16.4.2. Реакция на особые состояния в режиме прерываний	280
Задание адреса перехода при прерывании	280
Выполнение прерывания процессором i80x86.....	281
Контроллер прерываний 8259A	282
Пример организации прерываний от внешнего источника	284
Каскадное включение контроллеров прерываний.....	287
16.5. Измерение временных характеристик устройства	287
Глава 17. Прерывания и исключения	290
17.1. Прерывания и векторные вызовы подпрограмм.....	290
17.2. Типы исключений	293
17.3. Итоговая классификация прерываний.....	294
17.4. Программирование исключений.....	295
17.4.1. Trap-исключения.....	295

17.4.2. Fault-исключения.....	297
17.4.3. Различие между fault- и trap-исключениями	299
17.4.4. Практикум по trap- и fault-исключениям.....	300
17.5. Обработка прерываний при наличии системной процедуры	302
17.5.1. Сохранение и восстановление векторов.....	302
17.5.2. Дополнение к установленной процедуре обработки прерывания	306
17.6. Внешние прерывания.....	308
17.6.1. Доступ к данным из процедуры обработки прерывания.....	308
17.6.2. Ограничения на использование функций операционной системы.....	311
17.6.3. Определение причины прерывания	312
17.6.4. Практикум по внешним прерываниям	315
Прерывание от клавиатуры	315
Прерывание от последовательного канала связи.....	317
Глава 18. Резидентные программы	319
18.1. Установка резидентной процедуры	320
18.2. Взаимодействие с TSR-программой по данным	323
18.3. Уменьшение размера занимаемой памяти.....	327
18.4. Выгрузка TSR-программы	327
18.5. Вызовы DOS в TSR-процедурах	335
18.5.1. Способы определения состояния DOS	335
18.5.2. Pop-up программы	337
ЧАСТЬ IV. ПРИЛОЖЕНИЯ	343
Приложение 1. Биты, байты, слова, знаковые и беззнаковые числа.....	345
Приложение 2. Коды литер в стандарте ASCII	351
Приложение 3. Позиционные коды клавиш	354
Приложение 4. Функции BIOS-DOS	357
Функции BIOS для работы с клавиатурой.....	357
Функции прерывания 021 DOS	358
Функции низкоуровневого ввода/вывода.....	358
Функции ввода	358
Функции вывода.....	358
Функции для работы с файлами и потоками.....	359
Стандартные потоки ввода/вывода	359
Функции чтения и записи	359
Функции для создания, открытия и закрытия файлов	360
Функции для завершения программы.....	362
Функции для TSR-программ.....	362
Приложение 5. Настройки запуска a86	364
Ключи, или опции запуска a86.....	365
Переменная окружения a86.....	368

Приложение 6. Операторы и инструкции a86	369
Операторы	369
Инструкции.....	371
Условные обозначения.....	371
Общие правила установки флагов	372
Воздействие команд на флаги.....	373
AAA — Ascii Adjust after Addition	374
AAD — ASCII Adjust before Division	375
AAM — ASCII Adjust after Multiply	376
AAS — ASCII Adjust after Subtraction.....	377
ADC — Add with Carry	378
ADD — Addition.....	379
AND — Logical AND.....	379
BOUND — Check Array Index Against Bounds.....	380
CALL — Call Procedure	380
CBW — Convert Byte to Word	383
CLC — Clear Carry Flag.....	383
CLD — Clear Direction Flag	383
CLI — Clear Interrupt Flag.....	384
CMC — Complement Carry Flag	384
CMP — Compare.....	384
CMPS/CMPSB/CMPSW — Compare String Operands.....	385
CWD — Convert Word to Doubleword	387
DAA — Decimal Adjust after Addition	387
DAS — Decimal Adjust after Subtraction	388
DEC — Decrement by 1	389
DIV — Unsigned Divide	389
ENTER — Make Stack Frame for Procedure Parameters	390
IDIV — Signed Divide	391
IMUL — Signed Multiply	393
IN — Input from Port	393
INC — Increment by 1	394
INS/INSB/INSW — Input from Port to String.....	394
INT/INTO — Call to Interrupt Procedure	396
IRET — Interrupt Return.....	397
J<x> — Jump if Condition is Met.....	397
JMP — Jump	399
LAHF — Load Flags into AH Register	401
LEA — Load Effective Address	401
LEAVE — High Level Procedure Exit	402
LDS/LES — Load Far Pointer	402
LODS/LODSB/LODSW — Load String Operand.....	403
LOOP/LOOP<x> — Loop Control with CX Counter.....	404
MOV — Move Data	405
MOVS/MOVSb/MOVSsw — Move Data from String to String.....	405
MUL — Unsigned Multiplication of AL or AX.....	407
NEG — Two's Complement Negation	408

NOP — No Operation.....	408
NOT — One's Complement Negation.....	409
OR — Logical OR.....	409
OUT — Output to Port.....	410
OUTS/OUTSB/OUTSW — Output String to Port.....	410
POP — Pop a Word from the Stack.....	412
POPA — Pop all General Registers.....	412
POPF — Pop Stack into FLAGS.....	413
PUSH — Push Operand onto the Stack.....	413
PUSHA — Push all General Registers.....	414
PUSHF — Push Flags Register onto the Stack.....	414
REP/REP<x> — Repeat Following String Operation.....	415
RET — Return from Procedure.....	416
RCL/RCR/ROL/ROR — Rotate.....	417
SAHF — Store AH into Flags.....	418
SAL/SAR/SHL/SHR — Shift Instructions.....	418
SBB — Subtraction with Borrow.....	419
SCAS/SCASB/SCASW — Scan String Data.....	419
STC — Set Carry Flag.....	421
STD — Set Direction Flag.....	421
STI — Set Interrupt Flag.....	421
STOS/STOSB/STOSW — Store String Data.....	422
SUB — Subtraction.....	423
TEST — Logical Compare.....	423
WAIT — Wait until BUSY# Pin is Inactive.....	424
XCHG — Exchange Register/Memory with Register.....	424
XLAT/XLATB — Table Look-up Translation.....	425
XOR — Logical Exclusive OR.....	425

Приложение 7. Совместимость a86 с традиционными ассемблерами..... 426

Приложение 8. Прерывания от i80x87..... 431

Синхронизация процессора и сопроцессора.....	431
Обработка прерываний.....	432
Подключение сигнала прерывания.....	434
Внешнее прерывание через irq13.....	434
Внутреннее прерывание 16.....	435

Приложение 9. Ответы на контрольные вопросы из части I..... 436

К разделу 1.2.....	436
К разделу 1.4.....	436
К разделу 1.7.....	437
К разделу 2.1.....	439
К разделу 2.5.....	440
К разделу 5.1.....	441
К разделу 5.3.2. Прямая адресация.....	441
К разделу 5.3.2. Косвенная адресация.....	442

К разделу 5.3.3	442
К разделу 7.4.2	442
К разделу 10.1	443
К разделу 10.5	444
Приложение 10. Ошибки в a86 v4.05	445
Ошибка при оптимизации инструкции <i>call far</i>	445
Ошибка в операторе <i>bit</i>	445
Приложение 11. Описание дискеты	446

Введение

Ассемблер — это *язык* символического кодирования машинных инструкций, адресов и данных, а также *транслятор* с этого языка.

Главная особенность языка ассемблера — то, что в нем непосредственно отражены инструкции процессора и организация памяти. Так, например, инструкция ассемблера

```
inc ax
```

обозначает машинную команду "увеличить содержимое регистра процессора *ax* на единицу". Каждая машинная команда в программе кодируется отдельной инструкцией ассемблера.

Языки ассемблера различны, поскольку системы команд разных вычислительных систем не совпадают, а для аналогичных команд могут быть приняты разные обозначения.

Например, в процессорах семейства i80x86 имеется инструкция умножения, а в микроконтроллерах серии i8048 — нет. Соответственно, инструкции *mul*, *imul* ассемблера для i80x86 не имеют аналогов в ассемблере для i8048. Напротив, команда записи числа 10 в младший байт регистра-аккумулятора имеется в обоих семействах процессоров, но в ассемблерах обозначается по-разному:

```
mov    al, 10      mov    a, #10
```

Аналогичные команды в ассемблерах для микропроцессоров i8080 и i8085:

```
mov    #10, r0     ldi    a, 10
```

"Стандарт" ассемблера ограничен одним семейством микропроцессоров; он устанавливается авторами процессора и первого ассемблера для него.

Вместе с тем, ассемблеры — даже для разных вычислительных систем — во многом похожи, поскольку во всех процессорах имеется набор типовых инструкций — пересылка, арифметико-логических операций, переходов. Кро-

ме того, в любой современной вычислительной системе память образуется наборами 8-битных байтов.

Эта книга знакомит читателя с языком ассемблера для одного из наиболее развитых микропроцессоров — i80x86. После изучения ассемблера для i80x86 вы сможете самостоятельно осваивать ассемблеры для произвольных вычислительных систем.

Ассемблер a86 разработан автором asm86 — первого "официального" ассемблера для i8086. Ассемблер a86 следует "стандарту" tasm, но гораздо проще в обращении и содержит меньше ошибок. Особенности ассемблера a86 и отладчика d86 позволили нам представить в этой книге уникальные практики по программированию данных, освоению системы команд процессора i80x86, изучению форматов данных и системы команд арифметического сопроцессора i80x87.

После освоения a86 переход на более распространенные ассемблеры tasm или tasm не представляет сложности.

Структура изложения

Книга состоит из четырех частей.

Часть I, с изложением базовых средств и принципов ассемблера, включает в себя 10 глав. В них рассматриваются: определение данных, имен и типов, методы адресации, система команд i80x86, программирование циклов, сегментная модель памяти и eхе-программы.

Часть II, с изложением дополнительных возможностей языка ассемблера, состоит из пяти глав (11—15). В них рассмотрены: макрокоманды и средства условной трансляции, структурное программирование на ассемблере, интерфейс с языком C, обработка данных BCD-формата, форматы данных и система команд арифметического сопроцессора i80x87.

В части III (главы 16—18) изучаются: основы управления внешними устройствами, обработка прерываний и исключений, резидентные программы.

Часть IV содержит дополнительный материал и состоит из 11 приложений.

В приложении 1 приведены отдельные сведения из базового курса информатики, помогающие в изучении материала первой главы.

Приложения 2 и 3 содержат сведения, имеющие отношение к вводу/выводу символьной информации.

В приложении 4 дан краткий перечень функций операционной системы, используемых в примерах.

В приложении 5 рассмотрены настройки запуска ассемблера a86.

Приложение 6 представляет собой краткий справочник по операторам ассемблера и системе команд i80x86.

В приложение 7 рассмотрены вопросы совместимости с традиционными ассемблерами.

Приложение 8 посвящено организации аппаратных прерываний от арифметического сопроцессора i80x87.

В приложении 9 содержатся ответы на вопросы из первой части, выборочно.

В приложении 10 приведен перечень ошибок a86 v4.05, обнаруженных до апреля 2002 г.

К книге прилагается дискета с пакетом a86 v4.05 (2000 г.) и исходными текстами примеров. Описание дискеты дано в приложении 11.

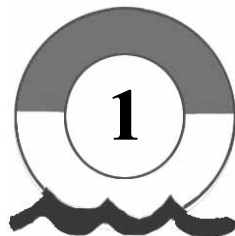


Часть I

Основы ассемблера

- Глава 1.** Данные, имена, типы
- Глава 2.** Определение имен
- Глава 3.** Практикум по программированию данных
- Глава 4.** Просмотр данных в отладчике
- Глава 5.** Способы адресации
- Глава 6.** Система команд i80x86
- Глава 7.** Программирование циклов
- Глава 8.** Исследование программ в d86
- Глава 9.** Примеры программ
- Глава 10.** Сегменты и ехе-программы

ГЛАВА 1



Данные, имена, типы

Я вас научу, с какого конца редьку есть!

Из к/ф "Сказ про то, как царь Петр арапа женил"

Основные вопросы первых двух глав — определение *данных*, определение и использование *имен*, понятие *типа* имени. Мы начинаем знакомство с ассемблером с данных, имен и типов потому, что эти объекты и понятия, с одной стороны, составляют существенную часть языка и, с другой стороны, в наименьшей степени зависят от системы команд микропроцессора.

1.1. Структура программы

Предварительно укажем местоположение данных в исходном тексте программы. Структура исходного текста программы на языке ассемблера а86, для получения исполняемой программы com-формата, следующая:

```
jmp start      ; первая исполняемая инструкция -
                ; команда обхода области данных
...
...           ; директивы определения данных
...
start:         ; начало программного кода
...
...           ; инструкции
...
ret           ; завершающая инструкция "возврат из
                ; подпрограммы", для выхода в DOS
```

Программа представляет собой последовательность операторов, по одному на строке.

В качестве оператора может быть задана:

- ☐ инструкция ассемблера, т. е. машинная инструкция в символическом виде;
- ☐ директива определения данных;
- ☐ директива ассемблера, т. е. команда для управления трансляцией.

Перед инструкцией — в той же строке — может быть поставлена метка (например, `start:`); аналогично, перед директивой определения данных можно поставить уникальное имя, которое в дальнейшем обозначает адрес данных в символическом виде. Литера ";" открывает комментарий — до конца строки.

В редких случаях — обычно, в учебниках — программа состоит только из инструкций или, еще реже (как в начале этой книги), из одних данных. (В таких случаях команда обхода данных не нужна.) В начале знакомства с ассемблером мы ограничиваемся данными и именами для их определения, составляя неисполняемые программы, без инструкций.

1.2. Директивы определения данных

Директивы `db`, `dw` и `dd` резервируют соответственно байты, 2-байтные слова и 4-байтные двойные слова. После оператора `db` (или `dw`, или `dd`) записывается значение элемента данных; знаком вопроса обозначается произвольная величина (неопределенное исходное значение элемента данных).

Примеры:

```
dw    10           ; 2-байтное слово со значением 10
dd    ?            ; двойное слово с произвольным значением
db    '?'          ; байт со значением ASCII-кода вопроса
db    '8'          ; байт со значением ASCII-кода '8'
db    1            ; байт со значением 1
db    1            ; еще один байт со значением 1
db    1            ; и еще один байт со значением 1
```

Следующие друг за другом операторы `db` (или `dw`, или `dd`) можно записать в одну строку, разделив значения запятыми:

```
db    '?', '8', 1, 1, 1
```

Обозначения ASCII-символов в директивах `db` можно сгруппировать, задав их одной строкой в кавычках:

```
db    '?8', 1, 1, 1
```

Повторяющиеся элементы внутри оператора `db` (или `dw`, или `dd`) группируются при помощи конструкции `dup`:

```
db    '?', '8', 3 dup 1
```

Конструкции `dup` допускают вложенность. Например, директива

```
db 2 dup (1, 3 dup 0)
```

задает двукратное повторение последовательности `(1, 3 dup 0)`:

```
db 1, 3 dup 0, 1, 3 dup 0
```

В итоге, эта директива означает:

```
db 1, 0, 0, 0, 1, 0, 0, 0
```

В табл. 1.1 приведены диапазоны чисел в машинном представлении в зависимости от размерности данных. Диапазоны *знаковых* и *беззнаковых* чисел в табл. 1.1 объединены, поэтому результирующие диапазоны несимметричны. Например, один байт представляет знаковое число в пределах от -128 до $+127$ и/или беззнаковое число от 0 до 255 ; результирующий диапазон — от -128 до 255 .

Таблица 1.1. Диапазоны данных

Размерность	Диапазон
Байт	$-128 - +255$
Слово	$-32\,768 - +65\,535$
Двойное слово	$-2\,147\,483\,648 - +4\,294\,967\,295$

Примечание

Один и тот же набор бит могут представлять разные значения в зависимости от того, считается ли оно знаковым или беззнаковым. Так, например, последовательность бит `11111111` задает беззнаковое число 255 и, в то же время, знаковое число -1 .

Слово можно задать по байтам: сначала младший байт, затем старший. Аналогично, двойное слово может быть определено парой смежных слов, или четверкой байт по последовательным адресам.

Примечание

В архитектуре Intel принято, что наиболее значимая часть числа из двух байтов/слов находится в байте/слове с *наибольшим* адресом.

В качестве примера приведем варианты определения слова со значением 256 и двойного слова со значением $65\,539$:

```
dw 256
dw 0100 ; 256
db 0, 1
```

```
dd    010003    ; 65539
dw    3, 1
db    3, 0, 1, 0
```

Контрольные вопросы

1. Запишите без использования `dup` директиву определения данных:

```
dw    3 dup (2 dup 5, 7), 9
```
2. Задайте одной директивой (без использования `dup`, сгруппировав ASCII-коды) следующие данные:

```
db    'AB'
db    '1', 2 dup (3 dup '?', 'x'), '0'
```
3. Задайте с использованием вложенных конструкций `dup` следующие данные:

```
dd    6, 1, 0, 1, 0, 5, 1, 0, 1, 0, 5
```
4. Определите десять байт и десять слов с произвольными (неопределенными) значениями.
5. Запишите директиву определения последовательности байт со значениями 'A', 'B', 'C', 0, 'D', 'E' сгруппировав обозначения ASCII-кодов.
6. Задайте последовательность данных: 30 байт со значениями 100, два слова — со значениями −5 и 19, двойное слово со значением 1000000.
7. Запишите директиву определения 12 байт со значениями 256.
8. Задайте одной директивой `db` следующую последовательность данных: пять слов со значением 1, четыре байта со значением 1, пять слов со значением 1, четыре байта со значением 1, и т. д. всего 100 раз.

1.3. Обозначение чисел

Числа могут быть заданы в двоичной, восьмеричной, десятичной и шестнадцатеричной системе счисления, а также кодами ASCII (литера в кавычках).

По умолчанию действует десятичная система счисления. Если первая цифра числа ноль, число считается шестнадцатеричным. Суффиксы `b`, `q`, `d`, `h` позволяет изменить систему счисления для одного (текущего) числа — на двоичную, восьмеричную, десятичную, шестнадцатеричную соответственно.

Внимание!

Если первая цифра 0, суффикс `d` или `b` воспринимается как шестнадцатеричная цифра.

Примеры (в скобках справа показаны десятичные значения):

```
14      — десятичное число           (14)
012     — шестнадцатеричное число    (18)
```

12h	— шестнадцатеричное число	(18)
11b	— двоичное число	(3)
011q	— восьмеричное число	(3)
011b	— шестнадцатеричное число	(283)
11d	— десятичное число	(11)
011d	— шестнадцатеричное число	(285)
ah	— имя	
0ah, 0a	— шестнадцатеричные числа	(10)

Рекомендуется взамен суффиксов `b` и `d` применять аналогичные суффиксы `xB` и `xD` — ассемблер не перепутает их с шестнадцатеричной цифрой.

Примеры:

1011, 01011xD	— десятичное
01011, 1011h	— шестнадцатеричное
1011q, 1011Q	— восьмеричное
01011xB	— двоичное
041	— ASCII-код буквы A
'A'	— ASCII-код буквы A
13	— ASCII-код служебного символа 'возврат каретки'
10	— ASCII-код служебного символа 'перевод строки'

Числа, кратные 1024, можно обозначать с использованием суффикса `k`. Суффикс `k` — это коэффициент 1024; число перед `k` — десятичное. В качестве примера приведем варианты определения 4 Кбайт данных:

```
db 4096 dup ?
db 4k dup ?
dw 2k dup ?
dd 1k dup ?
```

1.4. Символические обозначения чисел, выражения

Смысл введения символов для обозначения чисел — такой же, как и в языках высокого уровня: упростить корректировку параметров программы, сделать исходный текст понятней.

Символ для обозначения числа вводится директивой:

```
<name> equ <const>
```

где `<name>` — имя, составленное по общим правилам (начинается с буквы или литеры подчеркивания, за которой может следовать любое количество букв, литер подчеркивания и цифр), `<const>` — это число или *выражение*, составленное из чисел и/или символов, обозначающих числа.

Примеры:

```
_dozen    equ 12
size2     equ 200
sign_bit  equ 1000000xb
max_byte  equ 255
```

Операции, допустимые в выражениях, приведены в табл. 1.2.

Таблица 1.2. Операции, допустимые в выражениях

Операция	Действие или результат	Очередность
()	Изменение очередности операций	1
bit n	Число, в котором n-й бит установлен в 1, а остальные сброшены (биты нумеруются от нуля)	
high n	Значение старшего байта числа n	2
low n	Значение младшего байта числа n	
k / n	Целая часть результата деления k на n	
k mod n	Остаток от деления k на n	
k * n	Произведение чисел k и n	3
k shl n	Сдвиг двоичного кода числа k на n бит влево	
k shr n	Сдвиг значений разрядов числа k вправо на n бит	
k + n	Сумма чисел k и n	4
k - n	Разность чисел k и n	
not k	Поразрядная инверсия числа k	5
k and n	Поразрядное логическое умножение чисел k и n	6
k or n	Поразрядное логическое сложение чисел k и n	7
k xor n	Поразрядное исключающее "или" чисел k и n	
*k by n	Число, старший байт которого равен k, а младший n	8

Примечание

После знака операции +, -, / или * число с минусом следует задавать в скобках, например: -4*(-2). К сожалению, в этой ситуации транслятор a86 не считает отсутствие скобок ошибкой. В результате, выражение -4*-2, оказывается, равно -2.

Примеры:

```

alfa      equ 10 / 4                ; 2
beta      equ low (alfa * 083)     ; 6
gamma     equ low alfa * 083       ; 0106
          db 1 shl 5                ; 100000xb
          db bit 5                  ; 100000xb
          db not bit 0              ; 11111110xb
          db bit 1 + bit 3          ; 1010xb
          dw beta + 1 dup 'A' by 10

```

Последняя директива задает семь ($\text{beta}+1$) слов со значением ASCII-кода литеры 'A' в старшем байте и числом 10 в младшем байте.

Контрольные вопросы

1. Запишите директиву определения десяти кило-слов данных, каждый *байт* которых содержит ASCII-код литеры '0'.
2. Запишите число, в котором установлен пятый разряд, двумя способами — при помощи суффикса `xb` и при помощи конструкции `bit n`.
3. Запишите десятичные числа, соответствующие обозначениям:
`015`, `15`, `11b`, `17q`, `0a`, `bit 3`, `'A'`
4. Определите константу `hours`, равную произведению числа дней в году на число часов в сутках. Используя имя `hours`, определите слово со значением числа часов в году; затем, по-прежнему используя `hours`, задайте двойное слово со значением количества минут в году, и двойное слово со значением числа секунд в году.
5. Определите константу `len` со значением 119, затем определите слово со значением площади квадрата со стороной `len`.
6. Определите константу `hip` со значением 314, затем определите слово со значением площади прямоугольного равнобедренного треугольника со сторонами `hip`.
7. При помощи операций `bit` и операции `+` или `or` определите константу, содержащую единицы в третьем и седьмом разрядах; определите константу с другим именем, но с тем же значением, используя вместо `bit` операцию `shl`.
8. При помощи операций `bit`, операции `+` или `or`, а также операции `not` определите байт, в котором сброшены нулевой и третий разряды.

1.5. Переменные и метки

Пользовательские имена — это имена, определенные в тексте программы. Пользовательские имена применяются для обозначения адресов и констант взамен чисел. Рассмотренные в *разд. 1.4* символические обозначения чисел — это один из вариантов определения и применения пользовательских имен.

Адреса в символьной форме обозначаются именами *переменных* и *меток*.

Переменная — это символическое обозначение адреса *первого байта* (слова/двойного слова), определенного директивой `db` (`dw/dd`). Имя переменной задается перед директивой `db` (`dw/dd`); в дальнейшем это имя может быть использовано для обозначения адреса данных в *операндах инструкций*.

Метка — это символическое обозначение адреса *инструкции*. Метка определяется в начале строки как имя с двоеточием. Имя метки может использоваться для задания адреса перехода в командах *передачи управления*.

Примеры:

```

        jmp  start           ; использование метки start
                               ; в инструкции перехода

area1   db    2, -8, 7       ; определение переменной area1
        dw    -1
area2   dw    2, -8, 7       ; определение переменной area2

start:                               ; определение метки start
        inc   area1
        inc   area2+2
        inc   area2-2
        ...

```

Имя `area1` обозначает адрес *первого байта* из трех байт, определенных директивой `db`. Эти байты можно адресовать как `area1`, `area1+1`, `area1+2` (слагаемое задает смещение в байтах). Аналогично, слова, определенные директивой `dw`, можно обозначать как `area2`, `area2+2`, `area2+4` (смещение — вдвое, т. к. каждое слово занимает два байта).

Мнемоника `inc` обозначает инструкцию инкремента (увеличение на 1). Первая из таких инструкций задает инкремент байта по адресу `area1`; после ее выполнения значение байта равно трем. Вторая инструкция задает инкремент слова, исходное значение которого равно `-8`. Выполнение третьей инструкции увеличивает слово с начальным значением `-1`.

Внимание!

Обозначения `area1+3` и `area2-2` неэквивалентны (хотя это один и тот же адрес), поскольку `area1+/-<n>` адресует *байт*, а `area2+/-<n>` — *слово*.

Выводы из рассмотренного примера:

- ❑ с именем переменной связан *тип*: переменная обозначает либо адрес байта, либо адрес слова, либо адрес двойного слова — в зависимости от директивы — `db`, `dw` или `dd`, перед которой определено имя;
- ❑ одна и та же *мнемоника инструкции* может задавать операцию над байтом, словом или двойным словом — в зависимости от типа переменной.

1.6. Типы имен

Тип символьных констант и меток — `abs`. Типы переменных — `byte ptr` (сокращенно `byte`), `word ptr` (сокращенно `word`), `dword ptr` (сокращенно `dword`); `ptr` значит `pointer` — указатель.

Тип позволяет одинаково обозначать операцию над байтом, словом или двойным словом. Так, например, обозначение `inc` задает операцию увеличения на 1. Чего именно — байта, слова или двойного слова? Это следует из типа операнда.

Тип назначается автоматически при определении имени. Явное указание типа при помощи обозначений `abs`, `byte`, `word`, `dword` используется в `extrn`-описаниях (см. *разд. 9.5 и 10.5*); обозначения `byte`, `word`, `dword` можно применять для временного изменения типа переменной. Временное преобразование к типу `abs` выполняется оператором `offset`.

Пример с именами, определенными в *разд. 1.5*:

```
inc word ptr areal      ; (1)
inc word areal          ; (2)
inc byte area2+2        ; (3)
inc start               ; (4)   ?!
inc dword start         ; (5)
mov ax, area2           ; (6)
mov ax, offset area2    ; (7)
```

Инструкции (1) и (2) задают инкремент *слова* по адресу `areal`, несмотря на то, что `areal` обозначает адрес *байта*. Аналогично, инструкции (3) и (5) задают инкремент байта и двойного слова, независимо от типа имени; обратите внимание, что выполнение (5) приведет к искажению кода инструкции по адресу `start`. Инструкция (6) задает копирование *содержимого* слова по адресу `area2` в регистр `ax` процессора (значение `ax`, в результате, станет равно двум). Напротив, инструкция (7) задает запись в `ax` значения *адреса* `area2`.

Инструкция (4) воспринимается как ошибка, поскольку тип операнда — `abs`. Этот тип может обозначать либо константу, либо адрес инструкции. Константу нельзя изменить по определению, а модификация машинной команды — это либо ошибка, либо сомнительный программистский трюк. В (5) последнее ограничение обошли за счет временного преобразования типа.

Преобразования типов действуют временно — в пределах текущего обозначения имени. Так, например, имя `areal` сохраняет тип `byte`, несмотря на преобразование типа в инструкциях (1) и (2).

1.7. Типы и выражения

Рассмотрим правила *совместимости типов* в выражениях и правило определения типа *результата выражения*.

Правила совместимости:

1. Над именами типа `abs` выполнимы все операции, предусмотренные в выражениях (напоминаем, что типом `abs` характеризуются числа и метки).
2. Имя переменной (тип `byte`, `word`, `dword`) в выражении допускается только справа и/или слева от знаков `+` и `-`. (Имя переменной обозначает начальный адрес данных, определенных директивой `db/dw/dd`.)

Правило определения типа результата: результат операции `-` или `+` принимает тип *левого* операнда.

Исключение: результат операции `+` с одной переменной всегда получает тип этой *переменной*.

В следующем примере, где иллюстрируется правило определения результата, использован оператор `type <name>`. Этот оператор возвращает количество байт, которое занимает переменная `<name>`:

- ❑ 1, если переменная типа `byte`;
- ❑ 2, если `word`;
- ❑ 4, если `dword`;
- ❑ 0, если `<name>` — константа или метка (тип `abs`).

Пример:

```

byt1 db 4 dup ?
word1 dw ?

obj1 equ type (byt1 - 1)           ; 1 - "byte"
obj2 equ type (word1 - 1)          ; 2 - "word"
obj3 equ type (1 - byt1)           ; 0 - "abs"
obj4 equ type (1 - word1)          ; 0 - "abs"
obj5 equ type (word1 - byt1)        ; 2 - "word"
obj6 equ type (byt1 - word1)        ; 1 - "byte"
obj7 equ 2 - word1                  ; "abs"

_obj1 equ type (byt1 + 1)           ; 1 - "byte"
_obj2 equ type (word1 + 1)          ; 2 - "word"
_obj3 equ type (1 + byt1)           ; 1 - "byte"
_obj4 equ type (1 + word1)          ; 2 - "word"
_obj5 equ type (word1 + byt1)        ; 2 - "word"
_obj6 equ type (byt1 + word1)        ; 1 - "byte"
_obj7 equ 2 + word1                  ; "word"
```

```
var1    dw    byte1, 10 dup word1
var2    dw    offset byte1, 10 dup offset word1
var3    db    offset (word1 - byte1) dup ?
var4    db    offset word1 - offset byte1 dup ?
```

Обратите внимание: в списке значений, следующем за директивой `dw` (в частности, после `dup`), допускаются имена любого типа. Если указана переменная, то элемент данных принимает значение адреса; таким образом, `<var>` и `offset <var>` в качестве инициализирующих значений — это одно и то же.

Внимание!

Директива `equ` предназначена не только для задания символических констант. В общем случае, директива `<name> equ <exp>` вводит новое имя `<name>`, со значением и *типом* выражения `<exp>`. Так, имя `_obj7` в примере определено — при помощи `equ` — как переменная типа `word` с адресом на два больше, чем адрес `word1`.

Примеры ошибок и неточностей:

```
var5    db    (word1 - byte1) dup ?
var6    db    word1
var7    db    offset byte1
```

Ошибка — в первом операторе: счетчик повторов в конструкции `dup` должен быть типа `abs`. В следующих двух операторах допущена неточность: старший байт 16-разрядного адреса переменной будет отброшен (вероятно, с потерей значащих разрядов).

Контрольные вопросы

1. Перечислите имена и их типы, определенные в следующем фрагменте:

```
noodle   equ   '0' by 0
needle   db    40k / 7 dup 0
al       dw    offset needle - 0100
```

2. Найдите ошибки в нижеприведенном фрагменте:

```
src       db    "You're genius
dest      dw    bit 16 + offset src
size      equ   (dest - src) / 2
```

3. Данные определены таким образом:

```
byte1     db    1, 2
word1     dw    03040, 05060
```

Какие значения получают данные после выполнения последовательности инструкций: