ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Белгородский государственный технологический университет
им. В. Г. Шухова

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ
Основы программирования на языке Ассемблер

Методические указания к выполнению лабораторных работ
для студентов специальности 230105 — Программное обеспечение
вычислительной техники и автоматизированных систем

Белгород
2009

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Белгородский государственный технологический университет
им. В. Г. Шухова
Кафедра программного обеспечения вычислительной техники
и автоматизированных систем

Утверждено
научно-методическим советом
университета

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ
Основы программирования на языке Ассемблер

Методические указания к выполнению лабораторных работ
для студентов специальности 230105 — Программное обеспечение
вычислительной техники и автоматизированных систем

Белгород
2009

УДК 004(075.8)
ББК 32.973.26я73
О-64

Составители: ст. преп. А. И. Гарибов
ст. преп. Д. А. Куценко

Рецензент: канд. техн. наук, начальник отдела информационного и программного обеспечения ОАО «Гидроузел» М. С. Розанов

О-64 **Организация** ЭВМ и систем. Основы программирования на языке Ассемблер: методические указания к выполнению лабораторных работ для студентов специальности 230105 — Программное обеспечение вычислительной техники и автоматизированных систем / сост.: А. И. Гарибов, Д. А. Куценко. — Белгород: Изд-во БГТУ, 2009. — 92 с.

В методических указаниях содержатся краткие теоретические сведения и задания к выполнению лабораторных работ, посвящённых практическому изучению архитектурных особенностей и системы команд ЭВМ, построенной на базе процессора Intel, работающего в реальном режиме функционирования.

Методические указания предназначены для студентов специальности 230105 — Программное обеспечение вычислительной техники и автоматизированных систем.

Издание публикуется в авторской редакции.

УДК 004(075.8)
ББК 32.973.26я73

Содержание

Введение

Первая часть методических указаний к выполнению лабораторных работ по дисциплине «Организация ЭВМ и систем» содержат описания семи двухчасовых лабораторных работ, посвящённых практическому изучению архитектурных особенностей и системы команд ЭВМ, построенной на базе процессора Intel. Выполнение этих работ рассчитано на один учебный семестр продолжительностью 17 или 34 аудиторных часа.

Цель лабораторных работ — закрепление теоретических знаний и практическое изучение архитектуры современных ЭВМ, а также освоение инструментальных средств, необходимых при разработке программ на языке Ассемблер.

Каждая работа содержит краткие теоретические сведения, в которых изложен материал, необходимый для выполнения работы. В тексте содержатся ссылки на главы Учебника. Словом «Учебник» обозначается книга [1], которая является необходимым дополнением для выполнения работ. В этой книге содержится более полный теоретический материал по темам, которые рассматриваются в лабораторных работах.

В работах даются описания команд ассемблера, которые включают мнемонику команды и необходимые директивы, которые должны быть указаны в тексте программы перед использованием этих команд, их машинные коды и краткое описание. В машинных кодах команд используются следующие обозначения, после которых может указано возможное количество разрядов:

- data — непосредственный операнд;
- disp — смещение адреса операнда;
- oper — операнд;
- dest — операнд-приёмник;
- src — операнд-источник.

Также содержание работы включают варианты заданий к работам и контрольные вопросы, ответы на которые студент должен дать после выполнения работы.

Организация и проведение лабораторных работ

Студенты группы выполняют работы на закреплённых за ними компьютерах. Каждый студент получает индивидуальное задание в соответствии с порядковым номером в журнале.

Выполнение лабораторной работы предполагает предварительное изучение соответствующего раздела курса, методических указаний и

глав Учебника, ссылки на которые имеются в тексте работы. Далее студент выполняет своё индивидуальное задание и оформляет отчёт о лабораторной работе.

По результатам выполнения лабораторной работы проводится её защита, которая предполагает проверку теоретических знаний и практических умений, полученных в ходе выполнения лабораторной работы. Основанием для проведения защиты является наличие отчёта о лабораторной работе.

Структура и оформление отчёта о лабораторной работе

Для оформления отчётов о лабораторных работах следует завести отдельную тетрадь. Отчёты оформляются в рукописном виде, однако, допускается распечатка листингов программ и их трассировки, полученных в процессе компиляции программ.

Отчёт к лабораторной работе должен иметь следующую структуру:

1. номер лабораторной работы и её тема;
2. цель работы;
3. краткие теоретические сведения (по усмотрению студента);
4. индивидуальное задание
5. результаты выполнения индивидуального задания;
6. выводы по работе.

Лабораторная работа № 1
Структура программы на языке Assembler. Работа с программой DEBUG и пакетом TASM

Цель работы: изучить организацию ЭВМ и процессора, структуру программы на ассемблере, приобрести навыки работы с программой-отладчиком DEBUG и программами пакета TASM.

Основные понятия

Assembler — это язык программирования низкого уровня. Язык ассемблера является символическим представлением машинного языка, он неразрывно связан с архитектурой самого процессора. По мере внесения изменений в архитектуру процессора совершенствуется и сам язык ассемблера.

Для того чтобы научиться писать программы на ассемблере, нужно сначала изучить организацию компьютера и программно-аппаратную архитектуру процессоров Intel. Эта архитектура используется и в процессорах других производителей (AMD, SIS). Её описание представлено в главах 1 и 2 Учебника («Организация современного компь-

ютера» и «Программно-аппаратная архитектура IA-32 процессоров Intel»).

Для разработки программ на ассемблере используются пакеты программ TASM фирмы Borland и MASM фирмы Microsoft. Для разработки и отладки небольших простых программ также можно воспользоваться программой DEBUG.

Работа с программой DEBUG

DEBUG — это системная программа, позволяющая выполнять просмотр и изменение состояний процессора и памяти компьютера, побайтное тестирование и побайтную обработку дисковых файлов, что обеспечивает возможность выполнения отлаживаемых программ небольшими порциями. При этом программа выполняется под «наблюдением» отладчика.

Таким образом, основное назначение этой программы — отладка программ на уровне машинных кодов и языка ассемблера. Однако возможности, предоставляемые этой программой, делают её удобным инструментом для изучения организации персональных компьютеров.

Программа DEBUG включена во все версии операционной системы Windows. Для её запуска нужно в командной строке ввести команду debug. В качестве параметра команда запуска отладчика может включать имя обрабатываемого файла (или полный путь к файлу, если он находится не в текущем каталоге), например:

```
debug lab1.com
```

DEBUG — это программа, работающая по принципу «команда — действие». Для того чтобы произвести некоторую операцию отладчик должен получить соответствующую команду. В качестве сигнала о готовности принять команду, отладчик посылает на экран стандартный запрос — дефис («-»).

Для управления процессом отладки в DEBUG применяется набор команд, список которых можно получить, введя команду помощи (символ «?»).

При работе с программой DEBUG вводимые команды должны удовлетворять следующим правилам:

- все команды начинаются с буквы, заглавной или строчной;
- бо́льшая часть команд требует введения дополнительных параметров, часть из которых является необязательными;
- если два подряд расположенных параметра являются числами, то они разделяются пробелом или запятой (в противном случае параметры можно не отделять один от другого);

- все числа должны вводиться в шестнадцатеричном представлении;
- некоторые команды принимают в качестве параметра адрес, который может вводиться в двух формах: полный логический адрес — два шестнадцатеричных числа, записанные через двоеточие (первое число — сегментная компонента логического адреса, второе число — смещение) и короткий адрес — одно шестнадцатеричное число (смещение);
- при указании полного логического адреса допускается в качестве сегментной компоненты приводить имя сегментного регистра, из которого данная компонента будет выбираться.

Команда ассемблирования A(SSEMBLE) (перевод мнемокода ассемблера в машинный код)

Отладчик DEBUG можно использовать для введения операторов ассемблера непосредственно в оперативную память компьютера. Команду *ASSEMBLE* можно использовать при составлении коротких программ на ассемблере, а также при внесении изменений в существующие программы. Эта команда позволяет вводить мнемокод ассемблера непосредственно в память, избавляя от необходимости транслировать (ассемблировать) программу. Вводимый текст не может включать метки перехода в чистом виде.

При введении команды необходимо набрать *a* или *A* и, через пробел, необязательный параметр — адрес первой команды загружаемой программы. Если указан короткий адрес, то адрес сегмента выбирается из регистра CS. Если адрес не задан вообще, то машинный код будет помещаться в память, начиная с того места, где закончилась обработка предыдущей командой *ASSEMBLE.* Если после старта отладчика команда вводится в первый раз и в командной строке отсутствует начальный адрес, то размещение машинного кода производится с адреса CS:0100.

После введения команды ассемблирования на экране появляется начальный адрес. Это сигнал на введение первой команды программы. Если команда введена без ошибок, на экран выдается адрес следующей команды и отладчик опять переходит в режим ожидания. В случае ошибки отладчик обозначает её месторасположение. Если введены все команды программы, то нажимается *Enter* — команда *ASSEMBLE* заканчивает работу и возвращает управление отладчику.

Пример ассемблирования небольшой программы:

```
-a 0976:0100
0976:0100 MOV AL, 2A
```

```
0976:0102 MOV DI, 0200
0976:0105 MOV CX, 001D
0976:0108 CLD
0976:0109 REPNZ STOSB
0976:010B MOV AL, 24
0976:010D STOSB
0976:010E PUSH ES
0976:01OF POP DS
0976:0110 MOV DX, 0200
0976:0113 MOV AH, 09
0976:0115 INT 21
0976:0117 INT 20
0976:0119           <— Нажимается Enter
-
```

Команда дизассемблирования U(NASSEMBLE) (перевод машинного кода в мнемокод ассемблера)

Команда *UNASSEMBLE* служит для перевода машинного кода на язык ассемблера. При введении команды необходимо набрать *u* или *U* и, через пробел, необязательные параметры — начальный адрес обрабатываемого кода, конечный адрес обрабатываемого кода или его размер.

В командной строке *UNASSEMBLE* можно не указывать начальный адрес обрабатываемого кода. Если указан короткий адрес, то адрес сегмента выбирается из регистра CS. Если адрес не задан вообще, то машинный код обрабатывается с того места, где закончилась обработка предыдущей командой *UNASSEMBLE*. Если после старта отладчика команда вводится в первый раз и в командной строке отсутствует начальный адрес, то обработка машинного кода производится с адреса CS:0100.

Обрабатываемый участок памяти можно определить начальным и конечным адресами. При этом независимо от формы начального адреса конечный адрес должен быть коротким. Другой вариант задания обрабатываемого участка памяти — задание его начального адреса и размера. Чтобы отличить размер от короткого конечного адреса перед ним вводится символ *L*.

Если размер участка памяти, обрабатываемой командой *UNASSEMBLE*, не определён, то по умолчанию длина обрабатываемого участка равна 32 байтам.

Результатом выполнения команды дизассемблирования является листинг программы, сгруппированный в три колонки. В листинге слева (первая колонка) указывается полный логический адрес команды. Затем (вторая колонка) — значение составляющих команду

байтов в машинном коде. В третьей колонке находится соответствующая этому коду инструкция ассемблера.

Пример дизассемблирования программы, введённой в предыдущем примере:

```
-u CS:0100 L19
0976:0100      B02A    MOV AL, 2A
0976:0102      BF0002  MOV DI, 0200
0976:0105      B91D00  MOV CX, 001D
0976:0108      FC      CLD
0976:0109      F2      REPNZ
0976:010A      AA      STOSB
0976:010B      B024    MOV AL, 24
0976:010D      AA      STOSB
0976:010E      06      PUSH ES
0976:01OF      1F      POP DS
0976:0110      BA0002  MOV DX, 0200
0976:0113      B409    MOV AH, 09
0976:0115      CD21    INT 21
0976:0117      CD20    INT 20
-
```

Команда ввода данных в память E(NTER)

Ввод данных осуществляется с помощью *команды ENTER*. Эта команда позволяет побайтно корректировать содержимое памяти. Команда состоит из буквы *e* или *E* и адреса первого байта корректируемого блока. Если указан короткий адрес, то адрес сегмента выбирается в регистре DS.

Вводимые данные также включаются в командную строку. Они представляют собой последовательность чисел в шестнадцатеричном представлении и/или символьных значений, разделённых пробелом или запятой. Символьные значения заключаются в апострофы или кавычки; если требуется ввести символ апострофа или кавычки соответственно, то они удваиваются.

Проиллюстрируем работу *ENTER* на следующем примере:

```
-e DS:0000 20 2A 44 41 54 41 20 'IS' 20 48 45 52 45 2A 20
```

Команда вводит 16 значений. Данные последовательно заполняют память (побайтно), начиная с адреса *DS*:0000. Четырнадцать байтов занимают числа в шестнадцатеричном формате, два байта отводятся под символьную константу 'IS'.

Команда *ENTER* может использоваться для отображения и, в случае необходимости, корректировки значения конкретного байта. В этом случае команда состоит из буквы *e* или *E* и следующего за ней адреса.

При введении команды на экране появляется адрес байта и его значение:

```
-e DS:0000 0958:0000 20.
```

При нажатии на клавишу пробела на экране появляется значение следующего байта:

```
-e DS:0000 0958:0000 20. 2A.
```

Значение байта можно изменить. Для этого вводится новое шестнадцатеричное число. Однако символьные переменные в этом случае вводить нельзя.

Чтобы завершить выполнение команды, нажимается *Enter*. Появление дефиса — стандартного запроса отладчика — свидетельствует о его готовности принять следующую команду.

Команда вывода содержимого участка памяти на экран D(UMP)

Команда *DUMP* (*d* или *D*) служит для отображения на экране содержимого участка памяти. Полученный кусочек памяти — дамп, представляет собой последовательность значений байтов в шестнадцатеричном представлении, а также — в коде *ASCII*:

```
-d
0958:0100  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0958:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0958:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0958:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0958:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0958:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0958:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0958:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Числа в первом столбце, разделённые двоеточием — это полный логический адрес памяти, следующие 16 столбцов — значения, последовательно содержащиеся в памяти, начиная с этого адреса (в шестнадцатеричном виде), и, наконец, строка из 16 символов — символьное представление этих значений.

Значения, не имеющие символьного представления в коде *ASCII*, обозначаются символом «.». В рассмотренном примере изображены только точки, поскольку в коде *ASCII* не существует печатных символов со значением 00. Дамп отображает содержимое 128 последовательно расположенных байтов. В приведённом выше примере начальный адрес дампа — 0958:0100, конечный — 0958:017F.

Если вводить команду *d* не указывая параметров, *DEBUG* будет по-

следовательно выводить по 128 байтов памяти, т.е. начальный адрес дампа будет на единицу превышать конечный адрес дампа, полученного при введении предыдущей команды *d.* Если команда *d* вводится первоначально, то дамп выводится, начиная с адреса, по которому был загружен обрабатываемый файл.

Команда *DUMP* может использоваться с параметрами, которые задают начальный адрес дампа и его размер. Использование параметров аналогично использованию параметров в команде *UNASSEMBLE,* за тем исключением, что, если начальный адрес дампа задан в коротком формате, то сегментная компонента адреса выбирается из регистра DS.

Команда просмотра и изменения содержимого регистров R(EGISTER)

Команда *REGISTER* (*r* или *R*) выводит на экран и корректирует значения регистров и флагов состояния процессора. Эта команда также выдаёт информацию о следующей выполняемой команде:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100 NV UP Dl PL NZ NA PO NC
0958:0100 0000        ADD    [BX+SI],AL             DS:0000=CD
-
```

С помощью команды *r* можно изменить значение регистра. В этом случае в командной строке указывается его имя. Значение регистра выводится на экран. Теперь можно вводить новое число. Чтобы сохранить старое значение регистра, нажмите *Enter.*

```
-r CX
CX 0000
:245D
-r
AX=0000 BX=0000 CX=245D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100 NV UP Dl PL NZ NA PO NC
0958:0100 0000          ADD     [BX+SI],AL           DS:0000=CD
-
```

Команда *rf* выводит на экран флаги состояния процессора. Получив значения флагов, их можно изменить. Для этого вводится одно или несколько новых значений. Мнемонические обозначения состояний флагов приведены в табл 1.1.

Символьные значения вводятся в любом порядке через пробел или вообще без разделителя. Установим, например, значения флагов переполнения, знака и переноса:

Таблица 1.1

Мнемонические обозначения состояний флагов

Флаг	Установлен	Сброшен
Переполнения (есть/нет)	OV	NV
Направления (увеличение/уменьшение)	DN	UP
Прерывания (разрешение/запрещение)	EI	DI
Знака (минус/плюс)	NG	PL
Нуля (да/нет)	ZR	NZ
Дополнительного переноса (да/нет)	AC	NA
Чётности (чёт/нечет)	PE	PO
Переноса (да/нет)	CY	NC

```
-rf
NV UP Dl PL NZ NA PO NC -OV NG CY <-Подчёркнутое вводит
пользователь
-r
AX=0000 BX=0000 CX=245D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100 OV UP DI NG NZ NA PO CY
0958:0100 CD20          INT     20
-
```

Команда пошагового выполнения программы T(RACE)

Команда *TRACE* (*t* или *T*) — трассировка осуществляет пошаговое выполнение программы в машинном коде. При трассировке после выполнения каждой команды производится останов работы программы и на экран выводятся регистры и флаги состояния процессора. Полученная картинка аналогична картинке, получаемой с помощью команды *REGISTER*. Разница заключается только в том, что при введении *TRACE* перед появлением картинки, выполняется одна команда отлаживаемой программы.

Проиллюстрируем работу *TRACE* на примере нашей программы. Если она не загружена в память, то запустим *DEBUG* и введём:

```
-e CS:0100 BO 2A BF 00 02 B9 1D 00 FC F2 AA BO 24
-e CS:010D AA 06 1F BA 00 02 B4 09 CD 21 CD 20
```

Чтобы узнать адрес программы, введем команду *REGISTER*:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0976 ES=0976 SS=0976 CS=0976 IP=0100 NV UP Dl PL NZ NA PO NC
0976:0100 B001          MOV     AL,2A
```

При введении *t* выполняется команда по адресу CS:IP. После этого на экран выводятся регистры и флаги состояния:

```
-t
```

```
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0976 ES=0976 SS=0976 CS=0976 IP=0102 NV UP Dl PL NZ NA PO NC
0976:0102 BF0002        MOV     DI,0200
-t

AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0976 IP=0105 NV UP Dl PL NZ NA PO NC
0976:0105 B91D00        MOV     CX,001D
-
```

В командной строке *TRACE* можно указать адрес выполняемой команды. В этом случае после *t* набирается знак равенства (=) и нужный адрес. Если указан короткий адрес, то адрес сегмента выбирается из регистра *CS*:

```
-t=0100

AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0976 IP=0102 NV UP Dl PL NZ NA PO NC
0976:0102 BF0002        MOV     DI,0200
-
```

В этом случае выполнена команда по адресу CS:0100. Адрес следующей команды находится в регистрах CS:IP. Он равен 0976:0102.

Одной командой *TRACE* можно одновременно трассировать несколько команд отлаживаемой программы. Для этого при введении *t* просто указывается их количество. После выполнения каждой команды на экране появляется картинка с содержимым регистров и флагов состояния. При заполнении экрана новые данные выводятся в нижней его части, сдвигая данные в верхней части за пределы экрана.

```
-t6

AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0958 IP=0105 NV UP Dl PL NZ NA PO NC
0976:0105 B91D00        MOV     CX,001D

AX=002A BX=0000 CX=001D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0958 IP=0108 NV UP Dl PL NZ NA PO NC
0976:0108 FC            CLD

AX=002A BX=0000 CX=001D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109 NV UP Dl PL NZ NA PO NC
0976:0109 F2            REPNZ
0976:010A AA            STOSB

AX=002A BX=0000 CX=001C DX=0000 SP=FFEE BP=0000 SI=0000 DI=0201
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109 NV UP Dl PL NZ NA PO NC
0976:0109 F2            REPNZ
```

```
0976:010A AA            STOSB

AX=002A BX=0000 CX=001B DX=0000 SP=FFEE BP=0000 SI=0000 DI=0202
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109 NV UP Dl PL NZ NA PO NC
0976:0109 F2            REPNZ
0976:010A AA            STOSB

AX=002A BX=0000 CX=001A DX=0000 SP=FFEE BP=0000 SI=0000 DI=0203
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109 NV UP Dl PL NZ NA PO NC
0976:0109 F2            REPNZ
0976:010A AA            STOSB
-
```

Команда задания имени файла программы N(AME)

Команда *NAME* (*n* или *N*) присваивает имя обрабатываемому файлу. Затем этот файл загружается в память командой *LOAD* или записывается на диск командой *WRITE*.

Чтобы идентифицировать файл, наберите *n* или *N* и через пробел — имя файла. Воспользуемся *NAME*, чтобы присвоить нашей программе имя “mytest.pro”:

```
-n mytest.pro
-
```

Команда загрузки файла в память L(OAD)

Загрузка файла в память осуществляется, если в командной строке *DEBUG* указать имя файла. Другой способ — использование *команды LOAD* (*l* или *L*).

При использовании команды *LOAD* необходимо специфицировать файл с помощью команды *NAME*.

В командной строке *LOAD* можно указать начальный адрес, по которому загружается файл. Если указан короткий адрес, то адрес сегмента выбирается из регистра CS. При отсутствии начального адреса, загрузка производится по адресу CS:0100.

После загрузки отладчик запоминает количество занятой файлом памяти (в байтах) в регистрах BX (старшее слово) и CX (младшее слово).

Рассмотрим пример, в котором загружается в память файл «mytest.pro» по адресу CS:0100:

```
-n mytest.pro
-l
-r
```

```
AX=0000 BX=00CF CX=00CF DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100 NV UP Dl PL NZ NA PO NC
0958:0100 2A2A          SUB     CH,[BP+SI]           SS:0000=CD
-
```

В регистрах BX и CX находится значение 207 (000000CF). Это значит, что файл занял 207 байт. Тот же результат можно получить при введении спецификации файла в командной строке команды старта отладчика (“debug mytest.pro”).

Команда записи области памяти в файл W(RITE)

Команда *WRITE* (*w* или *W*) переписывает на диск данные, выбирая их из памяти. При этом спецификация создаваемого файла должна задаваться с помощью команды *NAME*.

Перед введением команды *WRITE* в регистры BX и CX записывается размер занимаемой файлом памяти в байтах (шестнадцатеричное число, занимающее 4 байта). Поэтому перед записью необходимо проверить содержимое этих регистров (с помощью *REGISTER*).

В командной строке *WRITE* можно указать начальный адрес памяти, по которому производится чтение данных с последующей записью их на диск. Если указан короткий адрес, то адрес сегмента выбирается из регистра CS.

Если начальный адрес не указан, то запись производится, начиная с адреса CS:0100.

Команда выхода из отладчика Q(UIT)

Чтобы выйти из отладчика и передать управление операционной системе, на его стандартный запрос вводится команда *q*:

```
-q
```

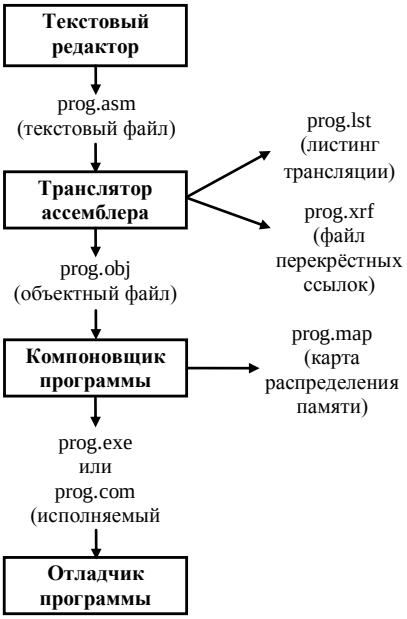
Разработка программ на языке ассемблера средствами пакета TASM

На рисунке (см. след. страницу) приведена общая схема процесса разработки программы на ассемблере. На схеме выделено четыре этапа этого процесса.

На первом этапе, когда вводится код программы, можно использовать любой текстовый редактор. В операционной системе Windows таким редактором может быть Блокнот (Notepad). При выборе редактора нужно учитывать, что он не должен вставлять «посторонних» символов (специальных символов форматирования). С этой точки зрения Microsoft Word в качестве основного редактора ассемблерных про-

грамм не годится.

Очень интересный редактор — Notepad++. Это бесплатная программа очень удобна для написания программ на различных языках программирования благодаря наличию подсветки ключевых слов синтаксиса соответствующего языка. Созданный с помощью текстового редактора файл должен иметь расширение *asm*.



Процесс разработки программы на ассемблере

Для выполнения остальных этапов разработки требуются специализированные программные средства из пакета MASM или TASM. Мы рассмотрим Turbo Assembler, поскольку процесс разработки ассемблерных программ с использованием этого пакета более нагляден. Из всего пакета для выполнения лабораторных работ нам понадобятся четыре файла: tasm.exe, tlink.exe, rtm.exe и td.exe.

Программа на ассемблере может записываться в двух формах: в обычной и упрощённой. В первых работах мы будем использовать упрощённую форму записи, так как она не требует детального описания всех сегментов программы и позволяет сконцентрировать внимание на особенностях выполнения команд ассемблера.

В упрощённом виде программа имеет следующую структуру:

```
директивы_TASM
режим_работы_TASM
MODEL модификатор модель памяти
.STACK размер стека
.DATA
        описание_переменных
.CODE
описание_процедур
имя_главной_процедуры PROC
        операторы ассемблера
имя_главной_процедуры ENDP
END имя_главной_процедуры
```

Здесь:

- директивы_TASM — директивы, разрешающие использование инструкций ассемблера, которые были введены в процессоры моделей старших i8086 (могут отсутствовать или встречаться перед командами ассемблера): .x86 — разрешает использование непривилегированных инструкций процессора i80x86; .x86p — разрешает использование всех инструкций процессора i80x86;
- режим_работы_TASM — задаёт режим ассемблирования MASM или IDEAL. В большинстве случаев должен использоваться режим MASM;
- модификатор — используется при наличии директив .386 и выше: use16 — сегменты выбранной модели памяти используются как 16-битные; use32 — сегменты выбранной модели памяти используются как 32-битные; dos — программа будет работать в MS-DOS;
- модель_памяти — определяет набор сегментов программы, размеры сегментов данных и кода, способ связывания сегментов и сегментных регистров: TINY — код и данные объединены в одну группу с именем DGROUP. Используется для создания программ формата *com*; SMALL — код занимает один сегмент, данные объединены в одну группу с именем DGROUP. Эту модель обычно используют для большинства программ на ассемблере; MEDIUM — код занимает несколько сегментов, по одному на каждый объединяемый программный модуль. Все ссылки на передачу управления имеют тип far. Данные объединены в одной группе; все ссылки на них имеют тип near; COMPACT — код в одном сегменте; ссылки на данные имеют тип far; LARGE — код в нескольких сегментах, по одному на каждый объединяемый программный модуль; FLAT — код и данные в одном 32-битном сегменте (плоская модель памяти);
- размер_стека — определяет размер сегмента стека. Стек задаётся директивой .STACK. Обычно размер стека выбирается равным 100h (или 256);
- описание_переменных — резервирование именованных областей памяти заданного размера;
- описание_процедур — подпрограммы, которые могут встречаться до или после главной процедуры;
- имя_главной_процедуры — определяет точку входа в программу. Рекомендуется (но не обязательно) выбрать имя

главной процедуры main или start, которое должно встречаться в открывающей и закрывающей директивах главной процедуры PROC и ENDP, а также в последней строке программы END.

В программах на ассемблере для TASM все числовые значения, в отличие от DEBUG, по умолчанию записываются в десятичной системе счисления (28, 173). Чтобы записать число в двоичной системе, нужно добавить в конец букву *b* (11100b, 10101101b), а в шестнадцатеричной — букву *h*. В последнем случае если число начинается с буквы *A, B, C, D, E* или *F*, то перед числом нужно поставить ноль (1Ch, 0ADh).

Директива .DATA определяет сегмент данных, в котором можно объявлять и инициализировать начальными значениями переменные и константы. Выделение памяти для переменных будет рассмотрено в следующей лабораторной работе.

Операторы ассемблера представляют собой инструкции процессора. В каждой команде может быть один, два или ни одного операнда. Каждая инструкция записывается в отдельной строке. Если в строке встречается символ «;», то за ним и до конца строки размещается комментарий.

Если в сегменте данных есть переменные, которые будут использоваться в программе, то до обращения к переменной должны выполняться инструкции, выполняющие настройку регистра сегмента данных на адрес сегмента данных в программе:

```
MOV AX, @data
MOV DS, AX
```

Если в сегменте данных переменных нет, то директиву .DATA можно опустить и настройку регистра сегмента данных не производить.

В главной процедуре последними должны идти инструкции завершения работы программы:

```
MOV AX, 4C00h
INT 21h
```

После выполнения этих команд программа передаёт управление вызвавшей её операционной системе.

Созданный в текстовом редакторе файл с текстом программы передаётся транслятору, затем, компонуется в запускаемый файл, который запускается в отладчике для проверки правильности работы программы. Подробно этот процесс изложен в главе 6 Учебника («Первая про-

грамма»). Для создания запускаемого файла средствами TASM нужно выполнить следующие команды:

```
tasm /zi prog , , ,
tlink /v prog.obj
```

Здесь prog соответствует файлу с текстом программы prog.asm, prog.obj — объектный файл, который создаётся транслятором и передаётся компоновщику. Эти файлы могут иметь и другие имена.

Если в программе используются 32-разрядные регистры или 32-разрядная адресация, то набирать команду TLINK нужно так:

```
tlink /v /3 prog.obj
```

В результате получим исполняемый модуль с расширением *exe* — prog.exe. Этот файл открывается в отладчике следующей командой:

```
td имя_исполняемого_модуля
```

Содержание работы

1. Ознакомиться с теоретическим материалом.
2. В соответствии со своим вариантом описать действия команд ассемблера и выполнить их в программе DEBUG (в этой работе все численные значения представлены в шестнадцатеричной системе счисления, во всех остальных — в десятеричной). В отчёт включить результаты ассемблирования команд, содержимое дампа памяти и трассировку программы.
3. Эти же команды представить в виде программы на ассемблере для TASM, произвести трансляцию и компоновку программы. В отчёт включить текст программы, листинг и карту распределения памяти.
4. Запустить скомпилированную программу в отладчике TD в пошаговом режиме. Сравнить содержимое регистров и флагов процессора после выполнения каждой инструкции с трассировкой этой программы в DEBUG. Сделать выводы.

Пример выполнения задания

```
MOV DX, AF
MOV AX, 9C60
MOV BX, DX
DIV BL
ADD AL, DL
```

Первая команда заносит в регистр DX значение AFh.

Вторая команда заносит в регистр AX значение 9C60h.

Третья команда заносит в регистр BX значение регистра DX.

Четвёртая команда выполняет деление слова, расположенного в регистре AX, на содержимое регистра BL. Частное заносится в регистр AL, остаток — в регистр AH.

Пятая команда производит сложение содержимого регистров AL и DL и заносит результат в регистр AL.

С помощью команды ассемблирования введём заданные команды. Добавим в конец команду NOP, которая не выполняет никаких действий и обозначает конец выполнения заданных команд.

```
-a
17B6:0100 MOV DX, AF
17B6:0103 MOV AX, 9C60
17B6:0106 MOV BX, DX
17B6:0108 DIV BL
17B6:010A ADD AL, DL
17B6:010C NOP
17B6:010D
```

В памяти ЭВМ начиная с адреса CS:100h находятся машинные коды, соответствующие введённым ассемблерным командам. Просмотрим эти коды с помощью команды отображения данных из памяти:

```
-d cs:100
17B6:0100  BA AF 00 B8 60 9C 89 D3-F6 F3 00 D0 90  ....`........
```

С помощью команды просмотра регистров просмотрим содержимое регистров и флагов до выполнения заданных команд, а также машинный код и адрес первой команды:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B6 ES=17B6 SS=17B6 CS=17B6 IP=0100 NV UP EI PL NZ NA PO NC
17B6:0100 BAAF00        MOV     DX,00AF
```

С помощью команды трассировки выполним заданные команды до команды NOP:

```
-t

AX=0000 BX=0000 CX=0000 DX=00AF SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B6 ES=17B6 SS=17B6 CS=17B6 IP=0103 NV UP EI PL NZ NA PO NC
17B6:0103 B8609C        MOV     AX,9C60
-t

AX=9C60 BX=0000 CX=0000 DX=00AF SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B6 ES=17B6 SS=17B6 CS=17B6 IP=0106 NV UP EI PL NZ NA PO NC
17B6:0106 89D3          MOV     BX,DX
```

```
-t

AX=9C60 BX=00AF CX=0000 DX=00AF SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B6 ES=17B6 SS=17B6 CS=17B6 IP=0108 NV UP EI PL NZ NA PO NC
17B6:0108 F6F3          DIV     BL
-t

AX=84E4 BX=00AF CX=0000 DX=00AF SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B6 ES=17B6 SS=17B6 CS=17B6 IP=010A NV UP EI PL NZ NA PO NC
17B6:010A 00D0          ADD     AL,DL
-t

AX=8493 BX=00AF CX=0000 DX=00AF SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B6 ES=17B6 SS=17B6 CS=17B6 IP=010C NV UP EI NG NZ AC PE CY
17B6:010C 90            NOP
```

Соответствующая программа на ассемблере для TASM будет выглядеть следующим образом:

```
MASM
MODEL SMALL
.STACK 256
.CODE
main PROC
       MOV DX, 0AFh
       MOV AX, 9C60h
       MOV BX, DX
       DIV BL
       ADD AL, DL

       MOV AX, 4C00h
       INT 21h
main ENDP
END main
```

После компилирования заданной программы получили листинг:

```
Turbo Assembler Version 4.1       27/09/09 22:55:59     Page 1
prog.asm

      1                            MASM
      2 0000                       MODEL SMALL
      3 0000                       .STACK 256
      4 0000                       .CODE
      5 0000                       main PROC
      6 0000  BA 00AF                      MOV DX, 0AFh
      7 0003  B8 9C60                      MOV AX, 9C60h
      8 0006  8B DA                        MOV BX, DX
      9 0008  F6 F3                        DIV BL
     10 000A  02 C2                        ADD AL, DL
     11
     12 000C  B8 4C00                      MOV AX, 4C00h
```

```
     13 000F  CD 21                        INT 21h
     14 0011                          main ENDP
     15                               END main

Turbo Assembler Version 4.1       27/09/09 22:55:59     Page 2

Symbol Table
Symbol Name                       Type   Value

??DATE                            Text   "27/09/09"
??FILENAME                        Text   "prog    "
??TIME                            Text   "22:55:59"
??VERSION                         Number 040A
@32BIT                            Text   0
@CODE                             Text    TEXT
@CODESIZE                         Text   0
@CPU                              Text   0101H
@CURSEG                           Text   _TEXT
@DATA                             Text   DGROUP
@DATASIZE                         Text   0
@FILENAME                         Text   PROG
@INTERFACE                        Text   000H
@MODEL                            Text   2
@STACK                            Text   DGROUP
@WORDSIZE                         Text   2
MAIN                              Near    TEXT:0000

Groups & Segments                 Bit Size Align  Combine Class

DGROUP                            Group
  STACK                      16  0100 Para         Stack   STACK
   DATA                      16  0000 Word         Public  DATA
_TEXT                        16  0011 Word         Public  CODE
```

После компоновки программы будет создана карта распределения памяти:

```
 Start  Stop   Length Name               Class
 00000H 00010H 00011H  TEXT             CODE
 00020H 00020H 00000H  DATA             DATA
 00020H 0011FH 00100H STACK             STACK

Program entry point at 0000:0000
```

Варианты заданий

1.
```
MOV  AL, 20
MOV  BL, 10
ADD  AL, BL
SUB  BL, 3
AND  AX, BX
```

2.
```
MOV  DX, 4DF3
INC  DH
MOV  DL, 0
OR   AX, DX
NOT  AX
```

3.
```
MOV  AX, 1111
MOV  BX, 2
MUL  BX
INC  AX
XOR  AX, FFFF
```

4.
```
MOV  AL, 3
DEC  AL
ADD  AL, 24
MOV  BX, 2F
MUL  BL
```

5.
```
MOV  BL, FF
PUSH BX
DEC  BL
POP  AX
SUB  AX, BX
```

6.
```
MOV  BX, 100
MOV  AX, 347
ADD  AX, BX
NEG  AX
SUB  BX, AX
```

7.
```
MOV  AX, 7
SUB  AX, 4
MOV  DX, AX
MOV  BL, 7
DIV  BL
```

8.
```
MOV  BX, 13
DEC  BX
MOV  AX, BX
IDIV BL
MOV  BL, AL
```

9.
```
MOV  BX, FF00
ROR  BX, 1
NOT  BX
SUB  BX, F5
MOV  AX, BX
```

10.
```
MOV  DX, FFFF
MOV  AX, 1111
ADD  AX, 1234
XOR  AX, DX
NOT  AX
```

11.
```
MOV  AX, 3003
DEC  AH
INC  AL
AND  AX, 4
XOR  AX, FFFF
```

12.
```
INC  AX
XOR  AX, FFFF
OR   AX, 4D
NEG  AX
MOV  CX, AX
```

13.
```
OR   AX, 40
XOR  BX, 40A0
MOV  BX, FFEE
ADD  AX, BX
SUB  BX, 96
```

14.
```
MOV  AX, 3940
ADD  AH, AL
MOV  AL, F1
MOV  BL, AH
MUL  BL
```

15.
```
MOV  AX, F
ROR  AX, 1
MOV  BX, 330
OR   AX, BX
XOR  AX, 13
```

16.
```
MOV  AX, 492
PUSH AX
MOV  CL, 19
DIV  CL
POP  CX
```

17.
```
DEC  AX
NOT  AX
MOV  CL, 5
MUL  CL
NEG  AX
```

18.
```
NOT  AX
AND  AX, 908
ROR  AX, 1
ADD  AX, 2EA
SUB  AX, A
```

19.
```
MOV  AX, 4DC1
PUSH AX
MOV  BX, 98
DIV  BL
POP  BX
```

20.
```
MOV  CL, 20
MOV  CH, 20
AND  CH, CL
SUB  CX, 20
IMUL CX
```

21.
```
XOR  BH, 3F
MOV  AX, 30
MUL  BH
ADD  AL, AH
SUB  BH, AL
```

22.
```
MOV  BX, 9347
NEG  BX
MOV  DX, 300
ADD  BX, DX
PUSH BX
```

23.
```
MOV  CL, 15
MOV  CH, 15
PUSH CX
POP  AX
XOR  AX, CX
```

24.
```
INC  AX
DEC  BX
SUB  AX, BX
AND  AX, 43
NOT  AX
```

25.
```
MOV  AH, 20
MOV  AL, 13
ADD  AH, AL
MUL  AH
SUB  AL, AH
```

Контрольные вопросы

1. Перечислите основные элементы персонального компьютера.

2. Что такое «ассемблер» и для чего он предназначен?
3. Что такое «архитектура ЭВМ»?
4. Основные принципы фон-неймановской архитектуры.
5. Основные компоненты процессора Intel Pentium и их назначение.
6. Какие ресурсы включает в себя программная модель архитектуры IA-32?
7. Что такое «регистр»? Классификация регистров.
8. Флаги регистра EFLAGS и их назначение.
9. Что такое «сегмент памяти»?
10. Виды сегментов, используемых при написании программ на ассемблере.
11. Назначение и основные задачи программы DEBUG.
12. Какие действия нужно выполнить в DEBUG, чтобы определить машинный код какой-либо команды ассемблера?
13. Какие действия нужно выполнить в DEBUG, чтобы выполнить программу на ассемблере при некоторых начальных значениях регистров и флагов?
14. Какие действия нужно выполнить в DEBUG, чтобы проверить правильность работы программы на ассемблере?
15. Основные этапы разработки программы на ассемблере средствами пакета TASM.
16. Структура программы на ассемблере в упрощённом виде.
17. Назначение и возможности транслятора, компоновщика и программного отладчика, входящих в пакет TASM.
18. Для чего нужны листинг и карта памяти, получаемые при создании исполняемого файла программы на ассемблере?
19. Как проверить правильность работы программы на ассемблере, используя пакет TASM?

Лабораторная работа № 2
Система машинных команд ассемблера

Цель работы: изучить способы задания операндов в командах ассемблера, представление команд в виде машинных кодов.

Основные понятия

Программирование на уровне машинных команд — это минимальный уровень, на котором возможно программирование компьютера. Система машинных команд должна быть достаточной для того, чтобы

реализовать требуемые действия, выдавая указания аппаратуре машины.

Каждая машинная команда состоит из двух частей: *операционной части,* определяющей, «что делать», и *операндной части,* определяющей объекты обработки, то есть то, «над чем делать». Вся эта информация должна быть определённым образом закодирована и формализована. Синтаксис языка Ассемблер подробно описан в главе 5 Учебника («Синтаксис ассемблера»).

Способы задания операндов

Операнды — это объекты, над которыми или при помощи которых выполняются действия, задаваемые инструкциями или директивами. Машинные команды могут либо совсем не иметь операндов, либо иметь один или два операнда. Большинство команд требует двух операндов, один из которых является источником (src), а другой — приёмником (dest — операндом назначения). Один операнд может располагаться в регистре или памяти, а второй операнд обязательно должен находиться в регистре или непосредственно в команде. Непосредственный операнд может быть только источником.

Операндами могут быть числа, регистры, ячейки памяти, символьные идентификаторы. При необходимости для расчёта некоторого значения или определения ячейки памяти, на которую будет воздействовать данная команда или директива, используются выражения, то есть комбинации чисел, регистров, ячеек памяти, идентификаторов с арифметическими, логическими, побитовыми и атрибутивными операторами.

Возможны следующие способы адресации операндов:

- неявная — операнд задаётся неявно на микропрограммном уровне;
- непосредственная — операнд задаётся в машинном коде команды;
- регистровая — операнд задаётся в одном из регистров общего назначения процессора;
- неявное задание операнда — операнд задаётся неявно на микропрограммном уровне;
- прямая — операнд находится в памяти, эффективный адрес операнда — в машинном коде команды;
- косвенная базовая — операнд находится в памяти, эффективный адрес операнда — в одном из 16- или 32-разрядных регистров общего назначения;

- косвенная базовая со смещением — операнд находится в памяти, эффективный адрес операнда определяется суммой значений указанного регистра общего назначения и смещения в машинном коде команды;
- косвенная индексная со смещением — операнд находится в памяти, эффективный адрес операнда определяется суммой значений указанного индексного регистра и смещения в машинном коде команды;
- косвенная базовая индексная — операнд находится в памяти, эффективный адрес операнда определяется суммой значений указанного регистра общего назначения и индексного регистра;
- косвенная базовая индексная со смещением — операнд находится в памяти, эффективный адрес операнда определяется суммой значений указанного регистра общего назначения, индексного регистра и смещения в машинном коде команды.

При задании операнда в памяти сегментная составляющая адреса по умолчанию берётся из сегментного регистра SS, если при формировании значения эффективного адреса операнда используется регистр BP/EBP, а в остальных случаях — из сегментного регистра DS.

Подробно способы задания операндов описываются в главе 5 Учебника («Синтаксис ассемблера»).

При использовании программы-отладчика DEBUG существуют некоторые ограничения, связанные с тем, что DEBUG эмулирует работу процессора Intel 8086, а многие функции появились в старших моделях процессоров:

- отсутствует доступ к регистрам FS, GS, а также ко всем 32-разрядным регистрам — все эти регистры появились в процессоре Intel 80386;
- недоступны команды, появившиеся в процессорах старше Intel 8086;
- можно использовать только 16-разрядную адресацию команд и данных;
- нельзя именовать переменные и метки — вместо имён нужно подставлять численные значения смещений адресов;
- при косвенной базовой адресации можно использовать только базовые регистры BX и BP;
- при косвенной индексной адресации можно использовать только индексные регистры SI и DI.

Типы данных в языке Ассемблер

При обработке данных командами ассемблера важную роль играет понятие типа данных. В отличие от других языков программирования, в ассемблере типы данных рассматриваются на двух уровнях — физическом и логическом.

На физическом уровне тип данных определяет объём памяти, занимаемой операндом. Большинство команд ассемблера могут обрабатывать данные размером 1, 2, 4 и 8 байт.

На логическом уровне тип данных определяет множество значений, которые может принимать операнд. При этом физическому типу данных может соответствовать один или несколько логических. Например, участок памяти размером 1 байт, в котором хранится последовательность из восьми нулей и единиц, соответствует физическому типу «байт», на логическом уровне может представлять:

- целое беззнаковое число в диапазоне 0 — 255;
- целое число со знаком в диапазоне –128 — +127;
- код символа в кодировке ASCII (или какой-либо другой);
- упакованное двузначное BCD-число.

Основные типы данных, поддерживаемых командами ассемблера, указаны в главе 5 Учебника («Синтаксис ассемблера»).

Формат машинных команд процессора IA-32

Машинная команда представляет собой закодированное по определенным правилам указание процессору на выполнение некоторой операции. Правила кодирования команд называются форматом команд. Длина машинной команды IA-32 может составлять от 1 до 15 байт.

Каждой команде процессора с конкретным набором операндов соответствует машинный код. И наоборот — любому машинному коду соответствует одна команда процессора, т.е. машинная команда всегда однозначна по отношению к производимым ею действиям на уровне аппаратуры.

Структура машинных команд ассемблера и правила получения их машинного кода описаны в главе 3 Учебника («Система команд процессора IA-32»). Коды операций основных команд процессора и значения некоторых полей представлены в приложении.

Содержание работы

1. Ознакомиться с теоретическим материалом.
2. В соответствии со своим вариантом описать действия команд ассемблера и указать способы адресации операндов.

3. Осуществить ассемблирование и дизассемблирование заданных команд.

Пример выполнения задания

```
MOV [BX+SI+2067], DX
```

Команда выполняет пересылку слова из регистра DX в память по адресу DS:[BX+SI+813h]. Первый операнд имеет базово-индексную адресацию со смещением, второй — регистровую.

Машинный код заданной команды: 89901308.

Первый байт: 10001001. Код операции — 100010 — соответствует команде MOV. Поле d = 0 — пересылка данных из поля reg в поле r/m. Поле w = 1 — пересылка слова (16 бит, или 2 байта).

Второй байт: 10010000. Поле mod = 10 — поле смещения в команде занимает два байта. Поле reg = 010 — регистр DX. Поле mod = 10 и r/m = 000 — эффективный адрес определяется следующим образом: BX+SI+disp16.

Следующие два байта представляют собой значение смещения disp16: 0813h = 2067. Проанализировав все поля машинной команды можно сделать вывод, что машинный код 89901308 соответствует команде MOV [BX+SI+0813h], DX.

Варианты заданий

1.
```
MOV  BX, 100
MOV  BP, [BX]
MOV  [BP+2], DL
MOV  AX, DX
ADD  AX, [BX+DI+4]
```

2.
```
MOV  AX, 200
ADD  AX, [BX]
MOV  DI, AX
SUB  AX, [BX+DI+2]
MOV  [DI], AL
```

3.
```
MOV  DI, 4
MOV  BL, 100
MOV  AX, [BX+DI+4]
MOV  CL, AL
MOV  [BX+SI], CX
```

4.
```
MOV  BP, 70
ADC  BP, [BP]
MOV  AL, BL
ADD  AX, [BP+7]
MOV  [BP+4], DX
```

5.
```
MOV  AX, 200
MOV  BX, AX
MOV  [BX+4], BX
MOV  CX, [SI+12]
MOV  DL, [BP+DI+14]
```

6.
```
ADD  BX, [BX+2]
MOV  DI, 8
MOV  DX, [BX+DI+8]
SUB  BX, DX
MOV  [SI], AL
```

7.
```
MOV  BX, 18
MOV  [BX+200], AL
ADD  BX, 1
MOV  DI, BX
MOV  DX, [BX+DI+761]
```

8.
```
MOV  AX, DX
MOV  SI, AX
ADD  AL, [SI]
MOV  [BP+4], AX
MOV  AX, [BX+DI+17]
```

9.
```
MOV  BP, 40
MOV  BX, [BP+2]
MOV  [BX+DI+2], AX
MOV  CX, AX
MOV  DL, [DI]
```

10.
```
MOV  AX, [BX+DI+14]
ADD  BL, AL
MOV  DX, [BX]
MOV  [DI], DX
SUB  AX, [BX+8]
```

11.
```
MOV  SI, 720
MOV  BP, 200
MOV  AL, [BP+SI+3]
MOV  [BP], DX
MOV  CX, [SI]
```

12.
```
MOV  BP, 30
MOV  SI, AX
MOV  [BP+SI], DL
MOV  DX, [BP]
ADD  AX, DX
```

13.
```
MOV  AX, 120
MOV  DI, AX
MOV  [BP], AX
MOV  CX, [SI+2]
MOV  DL, [BP+DI+10]
```

14.
```
MOV  AX, 8080
MOV  BL, AL
MOV  [BX+20], DX
MOV  DX, [BP]
ADD  AX, DX
```

15.
```
MOV  DI, 18
MOV  SI, AX
MOV  [BX], BP
MOV  AL, [BP+SI+2]
XOR  CX, [BX+DI]
```

16.
```
MOV  DX, [BP+SI+3]
MOV  CX, [BP]
ADD  DX, CX
MOV  [BP+2], DL
MOV  AX, [SI]
```

17.
```
MOV  AX, [BX+4]
MOV  BX, AX
MOV  [BX+DI+4], CX
ADD  BL, CL
MOV  DX, [SI+7]
```

18.
```
MOV  AL, 90
MOV  BL, [BP+SI+12]
MOV  [BX], AX
SUB  AX, 34
MOV  [SI+13], AX
```

19.
```
MOV  AX, 7070
MOV  BX, AX
MOV  [BX], DL
MOV  CX, [BX+4]
MOV  BX, [SI+1]
```

20.
```
MOV  BP, 100
MOV  BYTE PTR[BP],30
ADD  AX, [SI]
MOV  [BX+2], DX
SUB  CX, 6
```

21.
```
MOV  SI, 48
MOV  AX, [SI]
MOV  BP, AX
MOV  DL, [BP+DI+200]
MOV  [BP], CX
```

22.
```
MOV  AX, 2002
MOV  BX, AX
MOV  AL, [BX+9]
MOV  [SI], DX
ADD  BX, 794
```

23.
```
MOV  DI, 20
MOV  [SI], DI
MOV  CX, [BX+DI+7]
MOV  DX, [BP+31]
ADD  CL, DL
```

24.
```
MOV  BP, 324
ADD  BP, 15
SUB  AL, [BP]
MOV  [BX+56], AX
MOV  [BP+SI], BP
```

25.
```
MOV  BX, 700
MOV  CX, [BX]
ADD  BX, CX
MOV  [BP+DI], BL
MOV  [SI+1], BH
```

Контрольные вопросы

1. Алфавит и основные синтаксические конструкции языка Ассемблер.
2. Что такое «операнд»?
3. Виды операндов в командах ассемблера.
4. Перечислить режимы адресации операндов в памяти.
5. В каких случаях используются операнды-выражения?
6. Для чего применяются операторы переопределения типа и сегмента?
7. В чём отличие понятия «тип данных» на физическом и логическом уровнях?
8. Какие типы данных поддерживаются командами ассемблера?
9. Как формируется машинный код команды ассемблера?
10. Привести примеры команд, в машинных кодах которых присутствуют поля префиксов?
11. Как влияет на выполнение команды изменение значений отдельных полей её машинного кода?
12. Как определить местоположение операндов в команде по её машинному коду?
13. На какие классы делятся команды процессора?

Лабораторная работа № 3
Команды пересылки данных

Цель работы: изучить команды пересылки данных и особенности их применения.

Основные понятия

Команды пересылки данных можно подразделить на следующие группы:

- команды пересылки общего назначения;
- команды ввода/вывода через порт;
- команды пересылки адреса;
- команды для работы с регистром флагов.

Команды пересылки общего назначения

MOV dest, src

```
100010dw    mod reg r/m                     oper: 8/16/32
100011d0    mod sreg r/m                    oper: 16/32
1010000w    disp 8/16/32                    dest: AL/AX/EAX
1010001w    disp 8/16/32                    src : AL/AX/EAX
1011wreg    data 8/16/32                    oper: 8/16/32
1100011w    mod 000 r/m     data 8/16/32    oper: 8/16/32
```

Копирование содержимого операнда src в операнд dest.

Команда MOV используется для различного рода пересылок данных и имеет ряд особенностей (они касаются большинства команд ассемблера):

- направление пересылки в команде MOV всегда справа налево, т.е. из второго операнда в первый;
- значение второго операнда не изменяется;
- нельзя осуществить пересылку из одной области памяти в другую. Если такая необходимость возникает, то нужно использовать в качестве промежуточного буфера любой доступный в данный момент регистр общего назначения;
- нельзя загрузить в сегментный регистр значение непосредственно из памяти. Для такой загрузки требуется промежуточный объект. Это может быть регистр общего назначения или стек;
- нельзя переслать содержимое одного сегментного регистра в другой сегментный регистр. Выполнить такую пересылку можно, используя в качестве промежуточных регистры общего назначения;
- нельзя использовать сегментный регистр CS в качестве операнда назначения;
- если использовать в качестве одного из операндов регистр AL/AX/EAX, то ассемблер сгенерирует более быструю форму команды MOV.

```
MOV AL, 5             ;пересылка непосредственных данных
                      ;в регистр
MOV BL, AL            ;пересылка данных из одного регистра
                      ;в другой
MOV BX, DS            ;пересылка данных из сегментного регистра
                      ;в регистр общего назначения
MOV DX, [BX+SI+2]     ;пересылка данных из памяти в регистр
MOV WORD PTR [BX], 10 ;пересылка непосредственных данных
                      ;в память
```

MOVSX dest, src (386)

```
00001111    1011111w     mod reg r/m          dest: 16/32
```

Пересылка со знаковым расширением.

Команда преобразует операнд со знаком в эквивалентный ему операнд большей размерности со знаком. Для этого содержимое операнда src, начиная с младших разрядов, записывается в операнд dest. Старшие биты операнда dest заполняются значением знакового разряда операнда src. Первый операнд должен располагаться в регистре, второй — в регистре или памяти.

MOVZX dest, src (386)

```
00001111    1011011w     mod reg r/m          dest: 16/32
```

Пересылка с нулевым расширением.

Команда преобразует операнд без знака в эквивалентный ему операнд большей размерности без знака. Для этого содержимое операнда src, начиная с младших разрядов, записывается в операнд dest. Старшие биты операнда dest заполняются нулями. Первый операнд должен располагаться в регистре, второй — в регистре или памяти.

PUSH src

```
11111111    mod 110 r/m                    oper: 16/32
01010reg                                   oper: 16/32
011010s0    data 8/16/32                   oper: 8/16/32
00001110                                   src : CS
00010110                                   src : SS
00011110                                   src : DS
00000110                                   src : ES
00001111    10100000                       src : FS
00001111    10101000                       src : GS
```

Размещение операнда в стеке.

Команда уменьшает значение регистра-указателя стека SP/ESP на 2/4 и записывает значение источника в вершину стека по адресу SS:SP.

POP dest

```
10001111    mod 000 r/m                    oper: 16/32
01011reg                                   oper: 16/32
00011111                                   dest: DS
00000111                                   dest: ES
00010111                                   dest: SS
00001111    10100001                       dest: FS
00001111    10101001                       dest: GS
```

Извлечение значений из стека.

Команда копирует значение из вершины стека в регистр, ячейку памяти или сегментный регистр, после чего содержимое регистра SP/ESP увеличивается на 2/4. При этом нельзя вытолкнуть значение в регистр CS.

Команда PUSH используется совместно с командой POP для записи значений размером в слово или двойное слово в стек и извлечения их из стека. Также в стек можно помещать непосредственные значения и содержимое регистра CS. Операнды этих команд могут находиться не только в регистрах, но и ячейках памяти. При этом разрешены все виды адресации.

```
PUSH AX      ;помещение в стек содержимого регистра
PUSH CS      ;помещение в стек содержимого сегментного регистра
PUSH [BX+1] ;помещение в стек содержимого ячейки памяти
POP  BX      ;извлечение из стека содержимого регистра
POP  DS      ;извлечение из стека содержимого сегментного
             ;регистра
POP  [SI]    ;извлечение из стека содержимого ячейки памяти
```

PUSHA/PUSHAD (386)

```
01100000
```

Запись всех регистров общего назначения в стек.

Команда помещает в стек содержимое регистров в следующей последовательности: AX/EAX, CX/ECX, DX/EDX, BX/EBX, SP/ESP, BP/EBP, SI/ESI, DI/EDI.

POPA/POPAD (386)

```
01100001
```

Восстановление содержимого регистров общего назначения из стека.

Команда восстанавливает содержимое регистров общего назначения в порядке, обратном занесению значений в стек командой PUSHA/PUSHD.

XCHG dest, src

```
10010reg                                    src : AX/EAX
1000011w    mod reg r/m                     oper: 8/16/32
```

Обмен значений между операндами.

Команда предназначена для двунаправленной пересылки данных. Операнды должны иметь один тип. Не допускается напрямую обменивать между собой содержимое двух ячеек памяти.

```
XCHG AX, BX     ;обмен содержимого регистров
XCHG DX, [BX+2] ;обмен содержимого регистра и ячейки памяти
```

CMPXCHG dest, src (486)

```
00001111    1011000w     mod reg r/m          oper: 8/16/32
```

Сравнение с аккумулятором и обмен.

Если аккумулятор и dest не равны, то команда сбрасывает флаг ZF и пересылает содержимое dest в аккумулятор, иначе команда устанав-

ливает флаг ZF и пересылает src в dest. Первый операнд должен располагаться в регистре или памяти, второй — в регистре.

CMPXCHG8B dest (586p)

```
00001111    11000111     mod 001 r/m          oper: 64
```

Сравнение и обмен восьми байтов.

Команда сравнивает два операнда: один операнд находится в паре регистров EDX:EAX, другой — в 64-разрядной ячейке памяти. Если значения равны, то содержимое операнда dest заменяется содержимым 32-разрядных регистров ECX:EBX и устанавливается флаг ZF. Если значения не равны, то в пару регистров EDX:EAX заносится содержимое операнда dest и флаг ZF сбрасывается.

BSWAP src (386)

```
00001111    11001reg                          oper: 32
```

Изменение порядка следования байтов в операнде src.

Команда предназначена для перестановки байтов операнда в обратном порядке. В качестве операнда может быть только 32-разрядный регистр.

XLAT

```
11010111
```

Табличное преобразование байта.

Команда извлекает байт из памяти по адресу DS:[BX/EBX+AL] и помещает его в регистр AL. При этом допускается замена сегмента.

Команды ввода/вывода через порт

IN dest, src

```
1110010w    data 8                          dest: AL/AX/EAX
1110110w                                    dest: AL/AX/EAX
                                            src : DX
```

Ввод операнда размером байт, слово, двойное слово из порта.

Команда помещает данные из порта ввода-вывода в аккумулятор. Номер порта задаётся вторым операндом в виде непосредственной величины значением 0 — 255 или значения в регистре DX. Размер данных определяется размерностью первого операнда.

OUT dest, src

```
1110011w    data 8                          src : AL/AX/EAX
1110111w                                    dest: DX
                                            src : AL/AX/EAX
```

Вывод значения в порт ввода-вывода.

Команда выводит значения из аккумулятора в порт ввода-вывода, номер которого определяется операндом dest аналогично команде IN.

Команды пересылки адреса

LEA dest, src

```
10001101    mod reg r/m                       oper: 16/32
```

Загрузка эффективного адреса (смещения) операнда src в dest.

Команда заносит значение смещения адреса второго операнда, расположенного в памяти, в первый операнд, находящийся в регистре. Также команду можно использовать для вычисления сумм, разностей и произведений для трёх операндов, используя при этом особенности режимов адресации операндов.

```
LEA BX, [BX+DI+1]    ;в BX помещается значение BX+DI+1
LEA EAX, [EAX+EBX]   ;сложение значений регистров
LEA EAX, [EAX+EAX*2] ;вычисление утроенного значения
                     ;регистра EAX
LEA EAX, [EAX-EBX+2] ;вычисление суммы трёх значений
```

LDS dest, src

```
11000101    mod reg r/m                       oper: 16/32
```

LES dest, src

```
11000100    mod reg r/m                       oper: 16/32
```

LSS dest, src (386)

```
00001111    10110010     mod reg r/m          oper: 16/32
```

LFS dest, src (386)

```
00001111    10110100     mod reg r/m          oper: 16/32
```

LGS dest, src (386)

```
00001111    10110101     mod reg r/m          oper: 16/32
```

Загрузка из памяти полного указателя.

Команда заносит в пару регистров DS/ES/SS/FS/GS (в зависимости от используемой команды):dest полный адрес некоторой ячейки памяти, который записан в другой ячейке памяти.

```
LDS BX, [DI]    ;занесение в DS:BX значения из памяти
                ;по адресу DS:[DI]
LES SI, [BX+DI] ;занесение в ES:SI значения из памяти
                ;по адресу DS:[BX+DI]
```

Команды для работы с регистром флагов

LAHF

```
10011111
```

Загрузка регистра AH содержимым младшего байта регистра флагов EFLAGS.

SAHF

```
10011110
```

Загрузка регистра флагов EFLAGS из регистра AH.

Биты регистра AH для этих команд представляют значения флагов в следующей последовательности: SF:ZF:0:AF:0:PF:1:CF.

PUSHF
10011100
PUSHFD (386)
10011100
Размещение регистра флагов в стеке.

POPF
10011101
POPFD (386)
10011101
Извлечение регистра флагов из стека.

Эти команды аналогичны командам PUSH и POP, но их операндом является регистр флагов.

CLC
11111000
Сброс флага переноса CF.

STC
11111001
Установка флага переноса CF.

CMC
11110101
Инвертирование флага переноса CF.

CLD
11111100
Сброс флага направления DF.

STD
11111101
Установка флага направления DF.

CLI
11111010
Сброс флага прерывания IF.

STI
11111011
Установка флага прерывания IF.

Эти команды позволяют работать напрямую с флагами CF, DF и IF.

Более подробно о командах пересылки данных изложено в главе 7 Учебника («Команды обмена данными»).

Содержание работы

1. Ознакомиться с теоретическим материалом.
2. Исследовать выполнение всех команд пересылки данных с любыми возможными типами операндов.
3. В соответствии со своим вариантом решить поставленные задачи с помощью DEBUG и TASM.
4. В отчёт включить все необходимые листинги.

Задачи к выполнению лабораторной работы

1. Используя любые известные команды промоделировать выполнение команды PUSH.
2. Используя любые известные команды промоделировать выполнение команды POP.
3. Используя любые известные команды промоделировать выполнение команды XCHG.
4. Используя любые известные команды промоделировать выполнение команды XLAT.
5. Используя любые известные команды промоделировать выполнение команды LDS.
6. Обменять содержимое сегментных регистров DS и ES.
7. Сохранить в стеке содержимое всех регистров и регистра флагов, а затем восстановить их содержимое.
8. Установить флаги OF, DF, ZF и CF. Остальные флаги сбросить.
9. Сбросить значения флагов SF, ZF, AF, PF и CF. Остальные значения флагов установить.
10. Загрузить содержимое регистра флагов в регистр CX.

Пример решения задачи

Используя любые известные команды промоделировать выполнение команды MOVZX.

Команда MOVZX имеет два операнда. Промоделируем следующую команду:

```
MOVZX AX, BL
```

Команда пересылает содержимое регистра BL в младшие биты регистра AX, остальные биты регистра AX заполняются нулями.

Выполним эти действия в DEBUG, не используя команду MOVZX (при этом начальные значения всех регистров могут быть любыми):

```
-a
17C1:0100 MOV AL, BL
17C1:0102 MOV AH, 0
17C1:0104 NOP
```

Выполним эти же действия в TASM:

```
MASM
MODEL SMALL
.STACK 100h
.CODE
main PROC
       MOV AL, BL
       MOV AH, 0

       MOV AX, 4C00h
       INT 21h
main ENDP
END main
```

Варианты заданий

Номер варианта	Номера задач	Номер варианта	Номера задач
1	1, 6	14	3, 9
2	1 ,7	15	3, 10
3	1 ,8	16	4, 6
4	1, 9	17	4 ,7
5	1, 10	18	4 ,8
6	2, 6	19	4, 9
7	2 ,7	20	4, 10
8	2 ,8	21	5, 6
9	2, 9	22	5 ,7
10	2, 10	23	5 ,8
11	3, 6	24	5, 9
12	3, 7	25	5, 10
13	3, 8		

Контрольные вопросы

1. Классификация команд пересылки данных.
2. Какие способы адресации возможны для операндов в командах пересылки данных?
3. Какие команды позволяют пересылать данные в сегмент стека? Из сегмента стека?

4. Как можно изменить значения флагов, отличных от CF, DF и IF?

Лабораторная работа № 4
Команды выполнения арифметических операций

Цель работы: изучить арифметические команды и особенности их применения.

Основные понятия

Арифметические команды можно подразделить на следующие группы:

- команды сложения;
- команды вычитания;
- команды умножения;
- команды деления;
- команды изменения знака и размера операнда.

Команды сложения

INC oper

```
1111111w    mod 000 r/m                     oper: 8/16/32
01000reg                                    oper: 16/32
```

Увеличение операнда размером байт, слово, двойное слово на единицу.

ADD dest, src

```
0000010w    data 8/16/32                    dest: AL/AX/EAX
100000sw    mod 000 r/m     data 8/16/32    oper: 8/16/32
000000dw    mod reg r/m                     oper: 8/16/32
```

Сложение двух целочисленных двоичных операндов.

Команда складывает значения, находящиеся в операндах, и помещает результат в dest.

ADC dest, src

```
0001010w    data 8/16/32                    dest: AL/AX/EAX
100000sw    mod 010 r/m     data 8/16/32    oper: 8/16/32
000100dw    mod reg r/m                     oper: 8/16/32
```

Сложение с учётом флага переноса CF.

Команда складывает значения, находящиеся в операндах, прибавляет к результату значение флага CF и помещает результат в dest.

Если при выполнении команд ADD и ADC происходит перенос из старшего разряда результата, это означает выход результата за преде-

лы разрядности операнда. При этом устанавливается флаг CF, в противном случае флаг сбрасывается.

Если при выполнении сложения происходит перенос в старший разряд или из старшего разряда результата (но не оба переноса одновременно), это означает переполнение результата. При этом устанавливается флаг OF, в противном случае флаг сбрасывается. Анализ этого флага имеет значение только в случае сложения чисел с учётом знака.

XADD dest, src

```
00001111    1100000w     mod reg r/m          oper: 8/16/32
```

Обмен и сложение операндов.

Команда аналогична команде ADD, но перед сложением операндов обменивает их местами. Результат сложения заносится в операнд dest.

AAA

```
00110111
```

Корректировка результата сложения неупакованных BCD-чисел.

Если младшая тетрада регистра AL > 9 или AF = 1, то команда прибавляет к регистру AL значение 6, увеличивает значение регистра AH на 1 и устанавливает флаги AF и CF, в противном случае сбрасывает флаги AF и CF. При этом в обоих случаях команда обнуляет старшую тетраду регистра AL.

Команда применяется после сложения двух неупакованных BCD-чисел командой ADD. Каждая цифра неупакованного BCD-числа занимает младшую тетраду байта. Если результат сложения двух однозначных BCD-чисел больше 9, то число в младшем полубайте не является BCD-числом. Команда позволяет сформировать правильное BCD-число в младшем полубайте регистра AL и учесть перенос в старший разряд путём увеличения содержимого регистра AH на 1.

```
MOV AL, 5
MOV BL, 8  ;5 + 8
ADD AL, BL ;AL=5h+8h=0dh
AAA        ;AX=0103h (BCD-число 13)
```

DAA

```
00100111
```

Десятичная коррекция результата сложения двух упакованных BCD-чисел с целью получения правильного двузначного десятичного числа.

Если AF = 1 или значение младшей тетрады регистра AL > 9, то команда увеличивает содержимое регистра AL на 6, устанавливает флаг AF, при возникновении переноса при сложении устанавливает флаг CF (если флаг был установлен до выполнения команды, то он

сбрасывается), иначе сбрасывает флаг AF. Если CF = 1 или значение старшей тетрады регистра AL > 90h, то команда увеличивает значение регистра AL на 60h и устанавливает флаг CF, иначе сбрасывает флаг CF.

Команда используется после сложения двух упакованных BCD-чисел с целью корректировки двоичного результата сложения в правильное двузначное BCD-число. Если AF = CF = 1, то сумма больше десятичного значения 99. Если AF = 1, то сумма меньше 99 и перенос был учтён в старшей цифре результата. Если AF = CF = 0, то никакого переноса не было и корректировка результата не производится.

```
MOV AL, 69h
MOV BL, 74h ;69 + 74
ADD AL, BL  ;AL=69h+74h=0DDh
DAA         ;CF=1, AL=43h (BCD-число 143)
```

Команды вычитания

DEC oper

```
1111111w    mod 001 r/m                     oper: 8/16/32
01001reg                                    oper: 16/32
```

Уменьшение значения операнда на единицу.

SUB dest, src

```
0010110w    data 8/16/32                    dest: AL/AX/EAX
100000sw    mod 101 r/m     data 8/16/32    oper: 8/16/32
001010dw    mod reg r/m                     oper: 8/16/32
```

Вычитание двух целочисленных двоичных операндов.

Команда вычитает из значения операнда dest значение операнда src и заносит результат в операнд dest.

SBB dest, src

```
0001110w    data 8/16/32                    dest: AL/AX/EAX
100000sw    mod 011 r/m     data 8/16/32    oper: 8/16/32
000110dw    mod reg r/m                     oper: 8/16/32
```

Вычитание с заёмом.

Команда вычитает из значения операнда dest значение операнда src и значение флага CF и заносит результат в операнд dest.

Команды вычитания SUB и SBB не учитывают знаки операндов. Если после вычитания CF = 1, то это означает, что произошёл заём из старшего разряда и результат получился в дополнительном коде. Если OF = 1, то это означает, что результат вышел за диапазон представления знаковых чисел (то есть изменился старший бит) для операнда данного размера, и требуется корректировка результата.

При сложении и вычитании чисел также изменяются флаги ZF и SF.

CMP dest, src

```
0011110w    data 8/16/32                        dest: AL/AX/EAX
100000sw    mod 111 r/m      data 8/16/32       oper: 8/16/32
001110dw    mod reg r/m                         oper: 8/16/32
```

Сравнение двух операндов.

Команда выполняет те же действия, что и команда SUB, но при этом не меняются значения операндов, а изменяются только значения флагов.

AAS

```
00111111
```

Коррекция результата вычитания двух неупакованных одноразрядных BCD-чисел.

Если младшая тетрада регистра AL > 9 или AF = 1, то команда вычитает из регистра AL значение 6, уменьшает значение регистра AH на 1 и устанавливает флаги AF и CF, в противном случае сбрасывает флаги AF и CF. При этом в обоих случаях команда обнуляет старшую тетраду регистра AL.

Команда используется для коррекции результата вычитания двух неупакованных одноразрядных BCD-чисел после команды SUB. Операндами в команде SUB должны быть правильные одноразрядные BCD-числа.

```
MOV AX, 0105h ;BCD-число 15
MOV BL, 8     ;15 - 8
SUB AL, BL    ;AL=0FDh
AAS           ;AX=7
```

DAS

```
00101111
```

Десятичная коррекция после вычитания двух BCD-чисел в упакованном формате.

Если AF = 1 или значение младшей тетрады регистра AL > 9, то команда уменьшает содержимое регистра AL на 6, устанавливает флаг AF, в случае заёма при вычитании устанавливает флаг CF (если флаг был установлен до выполнения команды, то он сбрасывается), иначе сбрасывает флаг AF. Если CF = 1 или значение старшей тетрады регистра AL > 90h, то команда уменьшает значение регистра AL на 60h и устанавливает флаг CF, иначе сбрасывает флаг CF.

Команда используется после вычитания двух упакованных BCD-чисел с целью корректировки двоичного результата вычитания в пра-

вильное двузначное десятичное число. Если AF = CF = 0, то для вычитания требовался заём и результат скорректирован для получения двух правильных BCD-цифр. Если AF = 1, а CF = 0, то для вычитания младших цифр требовался заём, который был учтён в старшей цифре результата, и корректировка результата не требуется. Если после команды CF = 1, то имел место заём единицы из старшего разряда. Если у вычитаемого нет больше двоичных разрядов, то результат следует трактовать как отрицательное двоичное дополнение. Для определения его абсолютного значения нужно вычесть 100 из результата в регистре AL.

```
MOV AL, 44h
MOV BL, 27h ;44 - 27
SUB AL, BL  ;AL=1Dh
DAS         ;AL=17h (BCD-число 17)
```

Команды умножения

MUL oper

```
1111011w    mod 101 r/m                    oper: 8/16/32
```

Целочисленное умножение без учёта знака.

Команда выполняет умножение без учёта знаков. Явно указывается один из множителей, который может находиться в регистре или в памяти. Второй множитель задаётся неявно в аккумуляторе. Местонахождение результата определяется размером множителей (табл. 4.1).

Таблица 4.1

Местоположение множителей и результата при выполнении команд MUL и IMUL с одним операндом

Размер операндов	Второй множитель	Результат
Байт	AL	AX
Слово	AX	DX:AX
Двойное слово	EAX	EDX:EAX

Если старшая часть результата нулевая, то после операции произведения CF = 0 и OF = 0. Если эти флаги ненулевые, то результат вышел за пределы младшей части произведения и состоит из двух частей.

IMUL oper

```
1111011w    mod 100 r/m                      oper: 8/16/32
```

IMUL dest, oper (386)

```
00001111    10101111     mod reg r/m         oper: 16/32
011010s1    mod reg r/m       data 8/16/32   oper: 16/32
```

IMUL dest, oper2, oper1 (386)

```
011010s1    mod reg r/m      data 8/16/32      oper: 16/32
```

Целочисленное умножение со знаком.

Команда производит целочисленное умножение операндов с учётом их знаковых разрядов. Команда имеет три формы, отличающиеся количеством операндов:

- с одним операндом — требует явного указания местоположения только одного сомножителя, который может быть расположен в ячейке памяти или регистре, местонахождение произведения зависит от размерности множителей (см. табл. 4.1);
- с двумя операндами — оба операнда являются множителями, результат заносится на место первого операнда;
- с тремя операндами — первый операнд определяет местоположение результата, второй операнд — положение одного из множителей (регистр или память), третий операнд может быть непосредственным значением размером в байт, слово или двойное слово.

Для однооперандной команды флаги устанавливаются аналогично команде MUL. Для двух- и трёхоперандной команды если результат слишком большой и усекается, то устанавливаются флаги OF и CF, в противном случае флаги сбрасываются.

AAM

```
11010100    00001010
```

Коррекция результата умножения двух неупакованных BCD-чисел.

Команда делит содержимое регистра AL на 10, частное записывает в регистр AH, остаток — в регистр AL.

Команда используется для коррекции результата умножения двух неупакованных BCD-чисел, которые умножаются поразрядно как обычные двоичные числа командой MUL. Она не обязательно должна предшествовать операции умножения, с её помощью можно преобразовывать двоичное число из регистра AX от 0h до 63h в его десятичный эквивалент от 0 до 99.

```
MOV AL, 8
MOV BL, 9  ;8 × 9
MUL BL     ;AX=48h
AAM        ;AX=0702h (BCD-число 72)

MOV AX, 96 ;AX=60h
AAM        ;AX=0906h (BCD-число 96)
```

Команды деления

DIV oper

```
1111011w    mod 110 r/m                    oper: 8/16/32
```

Беззнаковое деление.

Команда выполняет целочисленное деление операндов с выдачей результата в виде частного и остатка от деления. Делимое задаётся неявно, его размер зависит от размера делителя, который явно указывается в команде и находится в регистре или памяти. В табл. 4.2 показано местоположение делимого, частного и остатка в зависимости от их размерности. При выполнении операции деления возможно возникновение исключительной ситуации 0 — ошибка деления. Эта ситуация имеет место в одном из двух случаев:

1. делитель равен 0;
2. частное слишком велико для его размещения в регистре AL/AX/EAX.

Таблица 4.2

Местоположение делимого, частного и остатка для команд DIV и IDIV

Размер операнда	Делимое	Частное	Остаток
Байт	AX	AL	AH
Слово	DX:AX	AX	DX
Двойное слово	EDX:EAX	EAX	EDX

IDIV oper

```
1111011w    mod 111 r/m                    oper: 8/16/32
```

Целочисленное деление со знаком.

Команда выполняет целочисленное деление операндов с учётом их знаковых разрядов. Расположение операндов и принцип работы аналогично команде DIV. При этом остаток всегда имеет знак делимого. Знак частного зависит от состояния знаковых битов делимого и делителя.

AAD

```
11010101    00001010
```

Подготовка двузначного неупакованного BCD-числа в регистре AX перед делением.

Команда умножает значение регистра AH на 10 и прибавляет результат к регистру AL, после чего обнуляет регистр AH.

Команда используется для преобразования двузначного неупакованного BCD-числа в регистре AX в двоичный эквивалент. Эти дей-

ствия необходимо выполнить перед операцией деления BCD-чисел с помощью команды DIV.

```
MOV AX, 0108h ;BCD-число 18
MOV BL, 9     ;18 / 9
AAD           ;AX=12h
DIV BL        ;AL=2, AH=0
```

Команды изменения знака и размера операнда

NEG dest

```
1111011w    mod 011 r/m                    oper: 8/16/32
```

Изменение знака (формирование двоичного дополнения) значения.

Команда вычисляет двоичное дополнение операнда путём вычитания его исходного значения из 0. Операнд располагается в регистре или в памяти. Команда инвертирует все разряды операнда и складывает его с единицей. Если операнд отрицательный, то операция NEG над ним означает получение его модуля.

CBW

```
10011000
```

Преобразование байта в слово.

Команда копирует значение старшего бита регистра AL во все биты регистра AH.

CWD

```
10011001
```

Преобразование слова в двойное слово.

Команда копирует значение старшего бита регистра AX во все биты регистра DX.

CWDE (386)

```
10011000
```

Преобразование слова в двойное слово.

Команда копирует значение старшего бита регистра AX во все биты старшей половины регистра EAX.

CDQ (386)

```
10011001
```

Преобразование двойного слова в учетверённое слово.

Команда копирует значение старшего бита регистра EAX во все биты регистра EDX.

Эти команды предназначены для расширения разрядности числа со знаком для подготовки делимого к операции деления или для приведе-

ния операндов к одной размерности в командах умножения, сложения и вычитания.

Более подробно об арифметических командах изложено в главе 8 Учебника («Арифметические команды»).

Содержание работы

1. Ознакомиться с теоретическим материалом.
2. Исследовать выполнение всех арифметических команд с любыми возможными типами операндов.
3. В соответствии со своим вариантом вычислить значение выражения с помощью DEBUG. Все операнды считать 8-разрядными знаковыми числами.
4. Решить поставленную задачу с помощью TASM. Все операнды считать 16-разрядными знаковыми числами.
5. В отчёт включить все необходимые листинги.

Пример решения задачи

Вычислить значение выражения: $\frac{b-a+1}{3c}$.

Напишем программу, решающую поставленную задачу, на DEBUG. Пусть значение переменной *a* заносится в регистр AL, значение *b* — в BL, значение *c* — в CL. Для тестирования возьмём значения a = 17, b = 34, c = 51, но программа должна работать для всех возможных значений переменных без отбрасывания значащих разрядов. Значение результата после выполнения программы будет записано: целая часть результата — в регистр AX, остаток — в DX.

```
-a
17C0:0100 MOV  AL, 11 ;AL=a
17C0:0102 MOV  BL, 22 ;BL=b
17C0:0104 MOV  CL, 33 ;CL=c
17C0:0106 CBW         ;AX=a
17C0:0107 XCHG AX, BX ;AL=b BX=a
17C0:0109 CBW         ;AX=b
17C0:010A SUB  AX, BX ;AX=b-a
17C0:010C INC  AX     ;AX=b-a+1
17C0:010D XCHG AX, CX ;AL=c CX=b-a+1
17C0:010F MOV  BL, 3  ;BL=3
17C0:0111 IMUL BL     ;AX=3c
17C0:0113 XCHG AX, CX ;AX=b-a+1 CX=3c
17C0:0115 CWD         ;DX:AX=b-a+1
17C0:0116 IDIV CX     ;(b-a+1)/(3c): целая часть - в AX,
                      ;остаток - в DX
17C0:0118 NOP
```

При трассировке получаем результат: целая часть = 0h = 0, остаток = 12h = 18.

Напишем программу для TASM. Для размещения значений переменных *a*, *b*, *c* отведём необходимое место в памяти в сегменте данных. Поскольку размер операндов в два раза больше, чем в программе для DEBUG, текст программы будет несколько отличаться.

```
MASM
MODEL SMALL
.STACK 100h
.DATA
        a  DW 17
        b  DW 34
        c  DW 51
        r1 DD ?  ;целая часть результата
        r2 DD ?  ;остаток результата
.CODE
main PROC
        MOV AX, @DATA
        MOV DS, AX

        .386
        MOV  AX, b    ;AX=b
        CWDE          ;EAX=b
        MOV  EBX, EAX ;EBX=b
        MOV  AX, a    ;AX=a
        CWDE          ;EAX=a
        SUB  EBX, EAX ;EBX=b-a
        INC  EBX      ;EBX=b-a+1
        MOV  AX, c    ;AX=c
        CWDE          ;EAX=c
        MOV  ECX, 3   ;ECX=3
        IMUL ECX      ;EDX:EAX=3c, но здесь весь результат
                      ;вмещается в EAX
        XCHG EAX, EBX ;EAX=b-a+1 EBX=3c
        CDQ           ;EDX:EAX=b-a+1
        IDIV EBX      ;(b-a+1)/(3c): целая часть – в EAX,
                      ;остаток – в EDX
        MOV  r1, EAX
        MOV  r2, EDX

        MOV AX, 4C00h
        INT 21h
main ENDP
END main
```

Варианты заданий

1. $\dfrac{2c-d+23}{\dfrac{a}{4}-1}$
2. $\dfrac{-2c+82d}{\dfrac{a}{4}-1}$
3. $\dfrac{\dfrac{c}{4}-62d}{a^2+1}$
4. $\dfrac{2c-\dfrac{d}{4}}{a^2+1}$
5. $\dfrac{2c-\dfrac{d}{3}}{b-4a}$
6. $\dfrac{2c-\dfrac{42}{d}}{c+a-1}$
7. $\dfrac{c-\dfrac{d}{2}+23}{2a^2-1}$
8. $\dfrac{cd+23}{\dfrac{a}{2}-4d-1}$
9. $\dfrac{2c+51d}{d-a-1}$
10. $\dfrac{2c-\dfrac{d}{2}+1}{a^2+7}$
11. $\dfrac{\dfrac{12}{c}-4d+73}{a^2+1}$
12. $\dfrac{-\dfrac{53}{a}+d-4a}{1+ab}$
13. $\dfrac{-\dfrac{25}{a}+c-ba}{1+\dfrac{cb}{2}}$
14. $\dfrac{8b+1-c}{\dfrac{a}{2}+bc}$
15. $\dfrac{\dfrac{4b}{c}-1}{12c+a-b}$
16. $\dfrac{a+\dfrac{c}{b}-28}{ba+1}$
17. $\dfrac{2b-a+bc}{\dfrac{c}{4}-1}$
18. $\dfrac{a-4b-1}{\dfrac{c}{31}+ab}$
19. $\dfrac{21-\dfrac{ac}{4}}{1+\dfrac{c}{a}+b}$
20. $\dfrac{2b-38c}{b+\dfrac{a}{c}+1}$
21. $\dfrac{\dfrac{ab}{4}-1}{41-ba+c}$
22. $\dfrac{ab+2c}{41-\dfrac{b}{c}+1}$
23. $\dfrac{2c+a-21}{\dfrac{cb}{a}+1}$
24. $\dfrac{8b-1-c}{2a+\dfrac{b}{c}}$
25. $\dfrac{\dfrac{4b}{c}+1}{2c+ac-b}$

Контрольные вопросы

1. Классификация команд выполнения арифметических операций.

2. Каким образом возможно задание операндов в командах ADD, SUB, MUL, DIV, IMUL и IDIV?
3. В каких случаях используют команды ADC и SBB?
4. Как организовать выполнение арифметических операций над BCD-числами в упакованном и неупакованном формате?
5. Для каких целей можно использовать команды CBW, CWD, CWDE и CDQ?
6. Что произойдёт, если при делении двух чисел командой DIV или IDIV результат не помещается в отведённый ему регистр?
7. Как можно установить факт возникновения переполнения или заёма при сложении или вычитании?
8. Какие действия следует предпринять, чтобы при выполнении арифметических операций не происходило переполнение?

Лабораторная работа № 5
Команды сопроцессора

Цель работы: изучить организацию сопроцессора, команды для выполнения операций над числами с плавающей точкой и особенности их применения.

Основные понятия

Арифметический сопроцессор предназначен для выполнения операций с вещественными числами в форме с плавающей точкой. В его состав входят дополнительные регистры и компоненты, расширяющие логику команд процессора.

Структура и программная модель сопроцессора описывается в главе 17 Учебника («Архитектура и программирование сопроцессора»).

Система команд сопроцессора включает в себя около 80 машинных команд. Эти команды можно разделить на несколько классов:

- команды передачи данных;
- команды сравнения данных;
- арифметические команды;
- команды трансцендентных функций;
- команды управления сопроцессором.

Мнемоническое обозначение команд сопроцессора характеризует особенности их работы. Все мнемонические обозначения начинаются с символа F (Float). Вторая буква мнемонического обозначения определяет тип операнда в памяти, с которым работает команда:

- I — целое двоичное число;

- B — целое десятичное число;
- отсутствие буквы — вещественное число.

Последняя буква P в мнемоническом обозначении команды означает, что последним действием команды обязательно является извлечение операнда из стека. Последняя или предпоследняя буква R (reversed) в мнемоническом обозначении команды означает реверсивное следование операндов при выполнении команд вычитания и деления, так как для них важен порядок следования операндов.

Минимальная длина команды сопроцессора — 2 байта. Рассмотрим основные команды сопроцессора.

Команды передачи данных

Группа команд передачи данных предназначена для организации обмена между регистрами стека, вершиной стека сопроцессора и ячейками оперативной памяти. Команды этой группы имеют такое же значение для программирования сопроцессора, как команда MOV — для программирования основного процессора. С помощью команд передачи данных осуществляются все перемещения значений операндов в сопроцессор и из него. По этой причине для каждого из трёх типов данных, с которыми может работать сопроцессор, существует своя подгруппа команд передачи данных. Собственно, на этом уровне все его умения по работе с различными форматами данных и заканчиваются. Главной функцией всех команд загрузки данных в сопроцессор является преобразование данных к единому представлению в виде вещественного числа расширенного формата. Это же касается и обратной операции — сохранения в памяти данных из сопроцессора.

Команды передачи данных в вещественном формате

`FLD src`

Загрузка вещественного значения из ячейки памяти в вершину стека.

Операнд src — значение в вещественном формате в памяти (m32/m64/m80) или в регистре ST(i). Команда уменьшает значение поля SWR.TOP на единицу; проверяет тег для регистра стека, физический номер которого находится в поле SWR.TOP. Если содержимое тега не равно 11b (регистр не пустой), то устанавливаются биты SWR.SF и SWR.IE и выполнение команды заканчивается. Если содержимое тега равно 11b, анализируется тип операнда и выполняются следующие действия:

- если операнд источник является вещественным значением в памяти размером 32 или 64 бита, то оно преобразуется в вещественное число в расширенном формате, после чего записыва-

ется в вершину стека;

– если операнд источник является вещественным значением в памяти размером 80 бит или регистром стека, то он копируется в вершину стека.

Значение в памяти должно иметь формат вещественного числа. В качестве операнда допускается использовать любой регистр стека, в том числе и ST(0). В последнем случае содержимое ST(0) попадет также в ST(1). Особое внимание нужно уделять процессу заполнения стека, чтобы избежать исключения недействительной операции вследствие переполнения стека. Необходимо следить за тем, чтобы регистр ST(7) был пустым. Для принудительного его освобождения можно использовать, например, команду FFREE ST(7).

FST/FSTP dest

Сохранение вещественного значения из регистра ST(0).

Если dest является ячейкой памяти (32- или 64-битовой), то перед сохранением производится округление числа в вершине стека до значения; соответствующего размеру мантиссы dest. Порядок числа также при необходимости приводится к размеру порядка dest. Режим округления выбирается исходя из значения в поле CWR.RC. Затем округленное значение из регистра ST(0) копируется в место (ячейку памяти или регистр), указанное операндом dest. После сохранения исходное значение в ST(0) остаётся на своём месте в стеке сопроцессора. Следует заметить, что копирование возможно и в непустой регистр стека сопроцессора.

Последнее действие FSTP — выталкивание значения из вершины стека, то есть увеличение его указателя вершины SWR.TOP = SWR.TOP + 1.

Команды передачи данных в целочисленном формате

FILD src

Целочисленная загрузка.

Преобразование целого значения из операнда src в вещественное расширенное представление, после чего уменьшение на 1 указателя вершины стека сопроцессора (поле SWR.TOP). Результат преобразования помещается в регистр ST(0). Исходное целочисленное значение может быть адресом ячейки памяти размером 16,32 или 64 бита. Нужно следить за тем, чтобы значение в той ячейке было целым, иначе команда хотя и будет выполнена, но результат операции будет неопределённым. Также необходимо исключить ситуации, когда все регистры стека сопроцессора заняты. Для страховки можно применять ко-

манду FFREE.

FIST/FISTP dest

Целочисленное сохранение.

Исходя из значения в поле RC (управление округлением) регистра CWR, команда округляет число в вершине стека до соответствующего целого значения:

- RC = 00b — до ближайшего целого;
- RC = 01b — до ближайшего меньшего целого;
- RC = 10b — до ближайшего большего целого;
- RC = 11b — дробная часть числа отбрасывается.

Результат преобразования помещается в ячейку памяти, адрес которой указан операндом dest. Если величина преобразованного значения превышает по модулю максимально представимое число в операнде dest, то в нём формируется наибольшее отрицательное число — 8000h или 80000000h. Дополнительное действие для FISTP — выталкивание значения из вершины стека.

Команда FIST округляет вещественное значение и записывает его в ячейку памяти. Далее результат может использоваться в командах основного процессора. Размер целочисленного результата преобразования может быть только 16 или 32 бита.

Команда FISTP позволяет выполнить преобразование значения из вершины стека в соответствующее округленное целое значение для того, чтобы с ним могли работать команды основного процессора. Дополнительный эффект по сравнению с командой FIST в том, что производится выталкивание значения из вершины стека. В отличие от FIST в команде FISTP результат целочисленного преобразования может составлять не только 16 и 32 бита, но и 64 бита.

Команды передачи данных в десятичном формате

FBLD src

Загрузка десятичного числа в стек сопроцессора.

Команда преобразует операнд src, который содержится в памяти в формате двоично-десятичного числа, в расширенный формат и помещает его в вершину стека сопроцессора — регистр ST(0). Команда не контролирует правильность подаваемых упакованных десятичных цифр, поэтому использование других символов в поле двоично-десятичного числа приведёт к неопределённому результату. Так как результат размещается в стеке сопроцессора, то в нём должен быть хотя бы один свободный регистр. Иначе возникнет исключение недействительной операции.

FBSTP src

Сохранение десятичного значения в памяти с выталкиванием.

Исходный операнд src находится в регистре ST(0). Команда преобразует исходный операнд в двоично-десятичный формат и записывает его в область памяти приемника размером 10 байт (18 упакованных десятичных цифр). Последнее действие команды — выталкивание исходного операнда из вершины стека.

Если в стеке было не целое число, то оно округляется в соответствии со значением в поле CWR.RC.

Команды загрузки констант

Основным назначением сопроцессора является поддержка вычислений с плавающей точкой. В математических вычислениях довольно часто встречаются предопределённые константы, и сопроцессор хранит значения некоторых из них. Другая причина использования этих констант заключается в том, что для определения их в памяти (в расширенном формате) требуется 10 байт, а это для хранения, например, единицы расточительно (сама команда загрузки константы, хранящейся в сопроцессоре, занимает два байта). В формате, отличном от расширенного, эти константы хранить не имеет смысла, так как теряется время на их преобразование в тот же расширенный формат. Для каждой предопределённой константы существует специальная команда, которая производит загрузку её на вершину регистрового стека сопроцессора:

FLD1/FLDL2T/FLDL2E/FLDLG2/FLDLN2/FLDPI/FLDZ

Загрузка константы.

- FLD1 — загрузка вещественной единицы: SWR.TOP = SWR.TOP – 1; ST(0) = +1,0.
- FLDL2T — загрузка двоичного логарифма от 10: SWR.TOP = SWR.TOP – 1; ST(0) = $\log_2 10$.
- FLDL2E — загрузка двоичного логарифма числа *e*: SWR.TOP = SWR.TOP – 1; ST(0) = $\log_2 e$.
- FLDLG2 — загрузка десятичного логарифма числа 2: SWR.TOP = SWR.TOP – 1; ST(0) = lg 2.
- FLDLN2 — загрузка натурального логарифма числа 2: SWR.TOP = SWR.TOP – 1; ST(0) = ln 2.
- FLDPI — загрузка числа π: SWR.TOP = SWR.TOP – 1; ST(0) = π.
- FLDZ — загрузка нуля: SWR.TOP = SWR.TOP – 1; ST(0) = +0,0.

Команды загрузки констант позволяют сократить объём выделяемой памяти для данных программы, так как хранение вещественных констант в памяти требует минимум 32 бита.

Команды обмена

`FXCH [src]`

Обмен содержимым регистров стека.

Команда FXCH обменивает содержимое регистра вершины стека и другого регистра стека сопроцессора. Команда имеет два варианта расположения операндов. При неявном задании исходные значения находятся в регистрах стека ST(0) и ST(1). В команде FXCH с одним операндом исходные значения содержатся в регистрах ST(0) и ST(i). Команда полезна для подготовки выполнения действий над операндами внутри стека, так как большинство команд сопроцессора ориентировано на работу с его вершиной.

Команды условной пересылки Pentium II/III

`FCMOVcc dest, src`

Условная пересылка данных между регистрами сопроцессора.

Команда проверяет состояние флагов в регистре EFLAGS (табл. 5.1). Если состояние флагов соответствует условиям выполнения команды, то она пересылает значение из источника (ST(i)) в приемник (ST(0)). В противном случае выполнение команды заканчивается без пересылки.

Таблица 5.1

Состояние флагов для пересылки значения из источника в приёмник

Мнемокод	Состояние флагов
FMOVB ST(0), ST(i)	CF = 1
FMOVE ST(0), ST(i)	ZF = 1
FMOVBE ST(0), ST(i)	CF = 1 или ZF = 1
FMOVU ST(0), ST(i)	PF = 1
FMOVNB ST(0), ST(i)	CF = 0
FMOVNE ST(0), ST(i)	ZF = 0
FMOVNBE ST(0), ST(i)	CF = 0 или ZF = 0
FMOVNU ST(0), ST(i)	PF = 0

Данная команда появилась впервые в системе команд процессора Pentium Pro. Она позволяет выполнить пересылку вещественных данных между регистрами сопроцессора с учётом некоторых условий, определяемых состоянием определенных флагов в регистре EFLAGS.

Это удобно и существенно повышает гибкость процесса программирования на языке ассемблера, одновременно уменьшая объём кода за счёт устранения команд сравнения и условной передачи управления. Кроме того, команда FCMOVcc дополняет систему команд сопроцессора, так как в последней отсутствовала инструкция пересылки данных из произвольного регистра стека ST(i) сопроцессора в регистр ST(0). Для определения факта поддержки этой команды необходимо использовать команду CPUID с EAX = 1: в регистре EDX будет сформирована информация о возможностях процессора, где за интересующее нас свойство отвечает бит 15 (должен быть единичным). На случай если текущий процессор не поддерживает команду FCMOVcc, желательно иметь альтернативный участок программы на основе традиционных команд.

Команды сравнения данных

Команды сравнения данных сравнивают значение числа в вершине стека и операнда, указанного в команде.

Команды сравнения вещественных данных

FCOM/FCOMP/FCOMPP

Сравнение вещественных чисел.

Поддерживаются два формата: FCOM/FCOMP [oper] и FCOMPP. Выполняется сравнение значения в регистре ST(0) и значения операнда, указанного в команде (m32/64/ST(i)) или принимаемого по умолчанию (регистр ST(1)). Результат сравнения определяется состоянием битов C3, C2 и C0 регистра SWR. Последняя операция для FCOMP — выталкивание значения из ST(0). Последняя операция FCOMPP — выталкивание значений из ST(0) и ST(1).

Для программирования реакции на результаты сравнения необходимо анализировать состояние битов C3, C2 и C0 регистра SWR сопроцессора. До появления команд FCOMI сопроцессор не имел для этого специальных средств, за исключением команды FSTSW AX, которая позволяет записать содержимое регистра SWR в регистр AX. Затем командой SAHF полученное содержимое регистра AH записывается в младший байт регистра EFLAGS/FLAGS. По местоположению биты C3, C2 и C0 соответствуют флагам ZF, PF и CF. Таким образом, после применения SAHF становится возможным реагировать на результаты сравнения значений в сопроцессоре командами условного перехода основного процессора. Команда FCOM формирует исключение недействительной операции в случае нахождения в стеке непредставимых значений (C3C2C0 = 111).

Алгоритм работы команды FCOMP аналогичен алгоритму FCOM, за исключением того, что FCOMP в качестве своего последнего действия выполняет выталкивание значения из вершины стека сопроцессора. Порядок использования результатов сравнения аналогичен тому, как это делается для FCOM.

Команда FCOMPP применяется в случаях, когда после операции сравнения значения в регистрах стека сопроцессора ST(0) и ST(1) должны быть удалены. В остальном порядок использования её результатов аналогичен таковому для команд FCOM и FCOMP. Команда FCOMPP, в отличие от команд FCOM и FCOMP, не имеет операндов и поэтому всегда работает только с регистрами ST(0) и ST(1).

`FUCOM/FUCOMP/FUCOMPP`

Неупорядоченное сравнение вещественных значений.

Команды FUCOM/FUCOMP/FUCOMPP сравнивают между собой значения в регистрах стека. Команда имеет два варианта расположения операндов:

- FUCOM/FUCOMP/FUCOMPP;
- FUCOM src.

В варианте без операндов сравниваемые значения находятся в регистрах стека ST(0) и ST(1).

В команде FUCOM с одним операндом сравниваемые значения находятся в регистрах стека ST(0) и ST(i). Команда выполняет сравнение значений, по результатам которого устанавливаются флаги C3, C2 и C0 в регистре SWR (табл. 5.2).

Таблица 5.2

Установка битов C3, C2 и C0 в регистре SWR по результатам сравнения регистров

Значения в регистрах ST(0) и ST(1)/ST(i)	Состояния битов C3, C2, C0 в регистре SWR
ST(0) и ST(1)/ST(i) несравнимы	C3 = 1, C2 = 1, C0 = 1
ST(0) = ST(1)/ST(i)	C3 = 1, C2 = 0, C0 = 0
ST(0) < ST(1)/ST(i)	C3 = 0, C2 = 0, C0 = 1
ST(0) > ST(1)/ST(i)	C3 = 0, C2 = 0, C0 = 0

Для программирования реакции на установку флагов C3, C2 и C0 необходимо анализировать их состояние. Для этого можно использовать команду FSTSW AX, которая позволяет записать содержимое регистра SWR в регистр AX. Затем командой SAHF содержимое регистра AH, в котором и находятся биты C3, C2, C0, нужно загрузить в

младший байт регистра EFLAGS/FLAGS. Местоположение битов C3, C2 и C0 соответствует позиции флагов ZF, PF и CF. В отличие от команды FCOM команда FUCOM не формирует исключений в случае нахождения в стеке непредставимых значений (C3C2C0 = 111).

Команда FUCOMP выполняет те же действия, что и FUCOM, и одно дополнительное действие — выталкивание значения из вершины стека.

Команда FUCOMPP выполняет действия, аналогичные FUCOM, но кроме того, выталкивает значения из регистров ST(0) и ST(1).

Команды сравнения целочисленных данных

`FICOM/FICOMP oper`

Сравнение целого и вещественного значений.

Сравнение значения в регистре ST(0) и целочисленного операнда в ячейке памяти m16/32(int). Результат сравнения определяется со стоянием битов C3, C2 и C0 регистра SWR сопроцессора (табл. 5.3).

Таблица 5.3

Результат сравнения операндов по состоянию битов C3, C2 и C0 регистра SWR

Результат сравнения	Биты C3, C2, C0 регистра SWR
Операнды несравнимы	C3 = 1, C2 = 1, C0 = 1
Второй операнд больше первого	C3 = 0, C2 = 0, C0 = 1
Второй операнд меньше первого	C3 = 0, C2 = 0, C0 = 0
Второй операнд равен первому	C3 = 1, C2 = 0, C0 = 0

Последнее действие FICOMP — выталкивание значения из вершины стека. Для программирования реакции на результаты сравнения необходимо анализировать состояние битов C3, C2 и C0 и регистра SWR. Обычно это делается с помощью команды FSTSW AX, которая позволяет записать содержимое регистра SWR в регистр AX. Затем командой SAHF содержимое регистра AH, в котором и находятся биты C3, C2, C0, записывается в младший байт регистра EFLAGS/FLAGS. При этом биты C3, C2 и C0 разместятся в полях флагов ZF, PF и CF. Для дальнейшего анализа можно использовать команды условного перехода основного процессора.

Команда FICOMP, подобно команде FICOM, выполняет сравнение вещественного значения в вершине стека сопроцессора с целочисленным значением в ячейке памяти, адрес которой указан в команде на месте поля операнд. Дополнительный эффект работы этой команды заключается в выталкивании значения из стека.

Специальные команды сравнения

FXAM

Определение типа операнда в регистре ST(0).

Команда определяет тип операнда в ST(0) и, исходя из него, устанавливает биты C3, C2, C0 (табл. 5.4). В бит C1 помещается знак операнда в ST(0).

Таблица 5.4

Определение типа операнда по состоянию битов C3, C2 и C0 в регистре SWR после выполнения команды FXAM

Состояние битов C3, C2, C0 в регистре SWR	Тип операнда
C3 = 0, C2 = 0, C0 = 0	Неизвестный тип
C3 = 0, C2 = 0, C0 = 1	Не число
C3 = 0, C2 = 1, C0 = 0	Корректное вещественное число
C3 = 0, C2 = 1, C0 = 1	Бесконечность
C3 = 1, C2 = 0, C0 = 0	Ноль
C3 = 1, C2 = 0, C0 = 1	Пусто
C3 = 1, C2 = 1, C0 = 0	Денормализованное число

FTST

Сравнение значения в регистре ST(0) с нулём.

Значение в вершине стека сравнивается с нулём. По результатам работы устанавливаются биты C3, C2, C0 в регистре SWR (табл. 5.5).

Таблица 5.5

Установка битов C3, C2 и C0 в регистре SWR по состоянию регистра ST(0)

Состояние регистра ST(0)	Состояния битов C3, C2, C0 в регистре SWR
Не определено	C3 = 1, C2 = 1, C0 = 1
ST(0) = 0	C3 = 1, C2 = 0, C0 = 0
ST(0) < 0	C3 = 0, C2 = 0, C0 = 1
ST(0) > 0	C3 = 0, C2 = 0, C0 = 0

Наличие команды FTST в системе команд сопроцессора позволяет экономить память и код для такой, достаточно часто встречающейся, операции, как сравнение некоторого значения с нулём. В результате работы команды устанавливаются биты C3, C2, C0 в регистре SWR. Для программирования реакции на установку этих битов необходимо

анализировать их состояние. Обычно это делается командой FSTSW AX, которая позволяет выгрузить содержимое регистра SWR в регистр AX. Затем командой SAHF содержимое регистра AH, в котором находятся биты C3, C2, C0, записывается в младший байт регистра EFLAGS/FLAGS. Между положением битов C3, C2, C0 и флагов ZF, PF и CF имеется однозначное соответствие.

Команды условного сравнения Pentium II/III

`FCOMI/FCOMIP/FUCOMI/FUCOMIP oper1, oper2`

Сравнение вещественных значений и установка EFLAGS:

- Команда проверяет, поддерживаются ли форматы чисел в операндах oper1 и oper2.
- Для команд FCOMI/FCOMIP если один или оба операнда — не-числа (NaN) или числа в неподдерживаемом сопроцессором формате, то возбуждается исключение недействительной операции сопроцессора, в результате чего управление передается соответствующему обработчику. Если бит CWR.IM = 1, устанавливаются флаги ZF = PF = CF = 1. Если оба операнда — корректные вещественные числа, выполнение команды продолжается.
- Для команд FUCOMI/FUCOMIP если один или оба операнда — не-числа (QNaN, но не SNaN) или числа в неподдерживаемом сопроцессором формате, то устанавливаются флаги ZF = PF = CF = 1. Иначе (один или оба операнда — не-числа SNaN или числа в неподдерживаемом формате) возбуждается исключение недействительной операции и, если CWR.IM = 1, устанавливаются флаги ZF = PF = CF = 1. Если оба операнда — корректные вещественные числа, выполнение команды продолжается.
- Непосредственно вычитание oper1 – oper2.
- По результатам вычитания установка флагов ZF, PF, CF в регистре EFLAGS (табл. 5.6). Последняя операция для FCOMIP/FUCOMIP — выталкивание значения из ST(0).

Команда FCOMI по действию аналогична команде FCOM, но имеет отличия, суть которых в следующем. Во-первых, последняя не позволяет сравнивать между собой регистры стека сопроцессора. Во-вторых, FCOMI представляет результат сравнения в более удобном виде, устанавливая сразу флаги в регистре EFLAGS.

Команда FCOM изменяет значения битов C3, C2, C0 регистра SWR сопроцессора. Для программирования реакции на результаты сравнения необходимо анализировать состояние этих битов. Команда FCOMI

Таблица 5.6

Установка флагов ZF, PF, CF в регистре EFLAGS по результатам сравнения командами FCOMIP/FUCOMIP

Результат сравнения	ZF	PF	CF
oper1 > oper2	0	0	0
oper1 < oper2	0	0	1
oper1 = oper2	1	0	0
Неподдерживаемые форматы	1	1	1

позволяет сделать то же самое, что FCOM (выгрузка флагов SWR в AX и загрузка их из AH в EFLAGS/FLAGS с помощью SAHF), но без лишних действий.

Для команд FCOMIP и FCOMP применимы те же замечания, что и для FCOMI и FCOM, за исключением дополнительного действия — выталкивания значения из стека.

Команды FUCOMI/FUCOMIP выполняют те же операции, что и FCOMI/FCOMIP. Разница в том, что FUCOMI/FUCOMIP возбуждают исключение «недействительный арифметический операнд» только тогда, когда оба операнда являются SNaN значениями или представлены в неподдерживаемом формате; числа QNaN приводят к установке битов C0, C2, C3 в 1 без генерации исключения, Команды FCOMI/FCOMIP возбуждают исключение «недействительный арифметический операнд» в случае, если один или оба операнда — нечисла любого вида или значения в неподдерживаемом формате.

Арифметические команды

Команды сопроцессора, входящие в группу арифметических команд, реализуют четыре основные арифметические операции — сложение, вычитание, умножение и деление. Имеется также несколько дополнительных команд, предназначенных для повышения эффективности использования основных арифметических команд. С точки зрения типов операндов арифметические команды сопроцессора можно разделить на команды, работающие с вещественными и целыми числами.

Целочисленные арифметические команды

Целочисленные арифметические команды предназначены для работы на тех участках вычислительных алгоритмов, где в качестве исходных данных используются целые числа в памяти в формате слово и короткое слово, имеющие размерность 16 и 32 бита.

Схема расположения операндов вещественных команд традиционна для команд сопроцессора. Один из операндов располагается в вершине стека сопроцессора — регистре ST(0), куда после выполнения команды записывается и результат, а второй операнд может быть расположен либо в памяти, либо в другом регистре стека сопроцессора. Допустимыми типами операндов в памяти являются все перечисленные ранее вещественные форматы за исключением расширенного.

В отличие от целочисленных арифметических команд, вещественные арифметические команды допускают большее разнообразие в сочетании местоположения операндов и самих команд для выполнения конкретного арифметического действия. Так, например, можно выделить три возможных варианта команды сложения. В дополнение к этим трем вариантам существует еще одна команда сложения, производящая дополнительное действие — удаление значения из стека.

FADD/FADDP/FIADD

Сложение двух операндов.

Команды имеют следующие варианты расположения операндов:

- FADD слагаемое_1 [, слагаемое_2];
- FADDP [слагаемое_1, слагаемое_2];
- FIADD слагаемое_1.

Выполняемые действия соответственно:

- без операндов исходные значения: ST(0) = слагаемое_1, ST(1) = слагаемое_2. Команда выполняет сложение: ST(1) = ST(0) + ST(1). Последнее действие выталкивание значения из регистра ST(0). Результат сложения — в ST(0);
- с одним операндом — исходные значения: слагаемое_1 — m32real/m64real или m16(32)int, ST(0) = слагаемое_2. Команда выполняет сложение: ST(0) = ST(0) + слагаемое_1;
- с двумя операндами — слагаемое_1 содержится в ST(0)/ST(i), слагаемое_2 — в ST(i)/ST(0). Результат сложения помещается в регистр ST(0) для команды FADD ST(0), ST(i) или в регистр ST(i) для команды FADD ST(i), ST(0). Последнее действие команды FADDP — выталкивание значения из регистра ST(0), после чего результат сложения оказывается в регистре ST(i-1).

При описании чисел в памяти следует представлять их в одной из двух возможных форм записи вещественного числа — с десятичной точкой или в экспоненциальной форме: 0.05 или 5e2.

Команда FIADD выполняет операцию сложения операндов в двух форматах: целом и вещественном, но допускает только один вариант расположения операндов — один операнд в памяти, другой в вершине

стека — регистре ST(0).

FSUB/FISUB/FSUBP

Вычитание операндов.

Команды выполняют операцию вычитания операндов, находящихся в вершине стека и в другом регистре стека или в памяти. Команды имеют следующие варианты расположения операндов:

- FSUBP;
- FSUB/FISUB вычитаемое;
- FSUB/FSUBP уменьшаемое, вычитаемое.

При этом:

- для команды без операндов (FSUBP) в регистре ST(0) находится вычитаемое, в регистре ST(1) — уменьшаемое. Команда выполняет вычитание ST(1) = ST(1) – ST(0). Последнее действие — выталкивание значения из регистра ST(0). Таким образом, результат операции остаётся в регистре ST(0);
- для команды с одним операндом (вычитаемое) в регистре ST(0) находится уменьшаемое, в области памяти m32(64)real или m16(32)int — вычитаемое. Вычитание производится по формуле ST(0) = ST(0) – вычитаемое;
- для команды с двумя операндами (уменьшаемое и вычитаемое) уменьшаемое находится в ST(0)/ST(i), вычитаемое — в ST(i)/ST(0). Результат помещается в регистр ST(0) для команды FSUB ST(0), ST(i) или в ST(i) для команды FSUB ST(i), ST(0).

Команда FSUBP выполняет операцию вычитания операндов, находящихся в вершине стека и в другом регистре стека или в памяти с последующим выталкиванием значения из ST(0).

Команда FISUB допускает только один вариант расположения операндов — вычитаемое в памяти (в целочисленном формате), уменьшаемое в вершине стека — регистре ST(0) (в вещественном формате).

FSUBR/FISUBR/FSUBRP

Инверсное вычитание операндов.

Команды имеют следующие варианты расположения операндов:

- FSUBRP;
- FSUBR/FISUBR уменьшаемое;
- FSUBR/FSUBRP вычитаемое, уменьшаемое.

При этом:

- для команды без операндов (FSUBRP) вычитаемое находится в регистре ST(1), уменьшаемое — в регистре ST(0). Команда

выполняет вычитание ST(1) = ST(0) – ST(1). Последнее действие — выталкивание значения из ST(0), после чего результат вычитания оказывается в регистре ST(0);

- для команды с одним операндом (уменьшаемое) уменьшаемое находится в области памяти m32(64)real или m16(32)int, вычитаемое — в регистре ST(0). Команда выполняет вычитание ST(0) = уменьшаемое – ST(0);
- для команды с двумя операндами (вычитаемое и уменьшаемое) уменьшаемое помещается в регистр ST(i)/ST(0), вычитаемое — в регистр ST(0)/ST(i). Результат заносится в регистр ST(0) для команды FSUBR ST(0), ST(i) или в ST(i) для команды FSUBR ST(i), ST(0).

Команда FSUBR выполняет операцию вычитания операндов, находящихся в вершине стека и в другом регистре стека или в памяти. Отличие команды FSUBR от команды FSUB в обратном расположении уменьшаемого и вычитаемого.

Команда FSUBRP выполняет операцию вычитания операндов, находящихся в вершине стека и в другом регистре стека или в памяти, с последующим выталкиванием значения из ST(0). Команда допускает все три варианта расположения операндов.

Команда FISUBR допускает только один вариант расположения операндов — уменьшаемое в целом формате в памяти, вычитаемое в вещественном формате на вершине стека — в регистре ST(0).

FMUL/FIMUL/FMULP

Умножение.

Команда имеет три варианта расположения операндов.

- FMULP;
- FMUL/FIMUL множитель_1;
- FMUL/FMULP множитель_1, множитель_2.

При этом:

- для команды без операндов (только FMULP) первый множитель находится в регистре ST(1), второй — в регистре ST(0). Команда выполняет умножение ST(1) = ST(1) · ST(0). Последнее действие — выталкивание значения из регистра ST(0). Результат умножения — в ST(0);
- для команды с одним операндом первый множитель (операнд) находится в ячейке памяти m32/m64 или m16(32)int, второй — в регистре ST(0). Команда выполняет умножение ST(0) = множитель_1 · ST(0);
- для команды с двумя операндами первый множитель (операнд

множитель_1) хранится в ST(0)/ST(i), второй множитель (операнд множитель_2) — в ST(i)/ ST(0). Команда перемножает множитель_1 на множитель_2. Результат помещается либо в регистр ST(0) для команды FMUL ST(0), ST(i), либо в регистр ST(i) для команды FMUL/FMULP ST(i), ST(0), то есть на место множителя_1.

Операнд в команде может быть только адресом ячейки памяти размером 16 или 32 бита. Необходимо следить за тем, чтобы значение в этой ячейке было целым, иначе результат операции будет неверным.

FDIV/FIDIV/FDIVP

Деление двух чисел.

Команды FDIV/FDIVP/FIDIV имеют несколько вариантов расположения операндов:

- FDIVP;
- FDIV/FIDIV делитель;
- FDIV/FDIVP делимое, делитель.

При этом:

- для команды без операндов делимое находится в регистре ST(1), делитель — в регистре ST(0). Результат деления помещается в ST(1): ST(1) = ST(1)/ST(0). Последнее действие — выталкивание значения из ST(0). Таким образом, окончательный результат попадает в регистр ST(0);
- для команды с оценим операндом (делитель) делимое находится в регистре ST(0), делитель — в ячейке памяти m32(64)real или m16(32)int. Результат помещается в ST(0);
- для команды с двумя операндами делимое и делитель хранятся в двух регистрах стека, один из которых — ST(0). Выполняется деление делимое = (делимое/делитель).

Команда FDIV выполняет операцию деления над операндами. При необходимости описания чисел в памяти они должны быть представлены в одной из двух возможных форм записи вещественного числа— с десятичной точкой или в экспоненциальной форме.

Для критичных по времени выполнения программ, содержащих много операции деления и не требующих большой точности вычислении, можно воспользоваться следующей возможностью команды FDIV. Вычисление можно ускорить в два раза, если сформировать соответствующее значение поля PC (управление точностью) в регистре CWR. Поле CWR.PC позволяет определить один из трёх типов точности вычислений: 24 бита (PC = 00), 53 бита (PC = 10), 64 бита (PC = 11). Точность в 64 бита принята по умолчанию. Для изменения значе-

ния поля PC регистра CWR используйте команду FLDCW источник_в_памяти, которая загружает 16-битовое значение из ячейки памяти в регистр CWR.

Команда FDIVP выполняет операцию деления над операндами с последующим выталкиванием значения из вершины стека. Регистр ST(0) нельзя использовать в качестве приёмника. При необходимости можно установить точность вычислений.

Команда FIDIVP выполняет операцию деления вещественного операнда на целочисленный. Операндом в команде FIDIVP может быть только адрес ячейки памяти размером 16 или 32 бита. Необходимо следить за тем, чтобы значение в этой ячейке было целым, иначе команда все же будет выполнена, но результат операции будет неверным. Аналогично другим командам деления, с помощью поля CWR.PC можно установить точность вычислений: 24 бита (CWR.PC = 00), 53 бита (CWR.PC = 10), 64 бита (CWR.PC = 11). Точность в 64 бита принята по умолчанию. Изменить значение в поле CWR.PC можно, используя команду FLDCW источник_в_памяти, которая загружает 16-битовое значение из ячейки памяти в регистр CWR.

FDIVR/FIDIVR/FDIVRP

Деление в обратном порядке.

Предусматриваются три варианта расположения операндов.

- FDIVRP;
- FDIVR/FIDIVR делимое;
- FDIVR/FDIVRP делитель, делимое.

При этом:

- для команды без операндов делимое находится в регистре ST(0), делитель — в регистре ST(1). Команда выполняет деление: ST(1) = ST(0)/ST(1). Поскольку последнее действие — выталкивание значения из регистра ST(0), то результат операции — в регистре ST(0);
- для команды с одним операндом делимое находится в ячейке памяти m32(64)real или m16(32)int, делитель в регистре ST(0). Команда выполняет деление: ST(0) = делимое/ST(0);
- для варианта с двумя операндами делимое и делитель хранятся в двух регистрах стека, один из которых — ST(0). Команда выполняет деление делитель = (делимое/делитель).

Команда FDIVR является альтернативой команде FDIV и выполняет операцию деления над операндами. При необходимости описания чисел в памяти они должны быть представлены в одной из двух возможных форм записи вещественного числа в программе — с десятич-

ной точкой или в экспоненциальной форме.

Вычисления можно ускорить в два раза, используя поле CWR.PC, как это описано для команды FDIV.

Команда FDIVRP выполняет операцию деления в обратном порядке операндов с последующим выталкиванием значения из вершины стека.

Команда FIDIVR выполняет операцию деления значения целочисленного операнда на вещественный операнд. Операнд в команде может быть только адресом ячейки памяти размером 16 или 32 бита. Значение в этой ячейке должно быть целым, иначе результат операции будет неверным. Аналогично другим командам деления, с помощью поля CWR.PC можно установить точность вычислений: 24 бита (CWR.PC = 00), 53 бита (CWR.PC = 10), 64 бита (CWR.PC = 11). Точность в 64 бита принята по умолчанию. Изменить значение в поле CWR.PC можно, используя команду FLDCW источник_в_памяти, которая загружает 16-битовое значение из ячейки памяти в регистр CWR.

Дополнительные арифметические команды

FSQRT

Вычисление квадратного корня.

Вычисление квадратного корня из содержимого регистра ST(0) и запись результата в ST(0).

FABS

Получение абсолютного значения (модуля) числа в ST(0).

Сброс знакового бита ST(0).

FCHS

Инвертирование знака значения в ST(0).

Команда FCHS выполняет замену значения в вершине стека сопроцессора на противоположное.

FXTRACT

Выделение порядка и мантиссы значения в ST(0).

Значение порядка выделяется и записывается в регистр ST(1); значение мантиссы помещается на вершину стека сопроцессора — в регистр ST(0). Разделение на составные части вещественного числа в вершине стека, в свою очередь, является составляющей полной поддержки функции $\log_b x$. Эта команда вместе с F2XM1 необходима для использования общей функции возведения в степень, нужна для масштабирования при преобразовании числа в десятичное представление

(для его визуализации на устройстве отображения) и бывает удобна в отладочных целях и при обработке исключений.

FPREM

Вычисление частичного остатка от деления ST(0) на ST(1).

Делимое хранится в ST(0), делитель — в ST(1). Команда вычитает значения полей порядков в ST(0) и ST(1): d = ST(0) – ST(1):

- если полученная разность d меньше 64, выполняется деление I = ST(0)/ST(1), результат которого округляется путём усечения к ближайшему меньшему целому: в ST(0) записывается новое значение, равное ST(0) = ST(0) – (ST(1) · I); бит SWR.C2 сбрасывается (это означает, что в ST(0) получен истинный остаток, меньше делителя, как это и требовалось); биты C0, C1, C3 в SWR устанавливаются равными значениям трёх младших битов значения частного I —I2, I1, I0;
- если полученная разность d больше 64, бит SWR.C2 устанавливается в 1 (это означает, что в ST(0) получен пока только частичный остаток, не удовлетворяющий требованию «остаток < делителя», и нужно повторить обращение к команде FPREM для получения истинного остатка); переменной n присваивается значение из диапазона 32–63; выполняется деление I = (ST(0) / ST(1)) / 2(d – n), затем округление I путём усечения к ближайшему меньшему целому; в ST(0) записывается значение, равное ST(0) – (ST(1) · I · $2^{(d-n)}$).

Чтобы лучше понять принцип работы команды FPREM, необходимо вспомнить процесс деления чисел в столбик. Когда числа не делятся нацело, то остаётся остаток. Этот остаток может быть двух типов. Остаток первого типа получается в процессе деления и численно больше делителя. Такой остаток условно назовем промежуточным. Последний промежуточный остаток, который численно меньше делителя, назовем конечным.

Команда FPREM носит название «частичный остаток», так как результат формируется итеративно. При этом характеристика значения в ST(0) не может быть уменьшена более чем на 63 при однократном выполнении инструкции. Если команде удается получить конечный остаток, меньший делителя в ST(1), то выполнение команды считается завершённым. Об этом можно судить по нулевому состоянию бита C2 в регистре SWR. В противном случае C2 устанавливается в 1, и результат в ST(0) является промежуточным. Характеристика частичного остатка в ST(0) (после выполнения FPREM) будет отличаться от зна-

чения в ST(0), которое было до выдачи команды FPREM, по меньшей мере на 32.

Для получения частичного остатка программа должна циклически выполнять команду FPREM до тех пор, пока бит C2 не очистится. К команде FPREM часто обращаются для уменьшения аргументов периодических функций. При этом в качестве делителя используется значение π/4. Когда такое уменьшение завершено, то есть получен конечный (частичный) остаток, особое значение имеют биты C3, C1 и C0 регистра SWR. В них команда FPREM помещает 3 младших бита частного. Их значение позволяет определить один из восьми секторов круга, в котором находится значение исходного угла. Команда FPREM поддерживается ради совместимости с сопроцессорами 8087 и 80287, поскольку не соответствует стандарту IEEE 754. Для получения остатка от деления с плавающей запятой в стандарте IEEE 754 следует использовать команду FPREM1.

FPREM1

Вычисление частичного остатка от деления ST(0) на ST(1) в соответствии со стандартом IEEE 754.

Делимое хранится в регистре ST(0), делитель — в регистре ST(1). Команда вычитает значения полей порядков в ST(0) и ST(1): d = ST(0) – ST(1):

- если полученная разность d меньше 64, то выполняется деление I = ST(0)/ST(1), результат которого округляется к ближайшему целому: в ST(0) записывается новое значение, равное ST(0) = ST(0) – (ST(1) · I); бит SWR.C2 сбрасывается (это означает, что в ST(0) получен истинный остаток, удовлетворяющий требованию «остаток < делителя»); биты C0, C1, C3 в SWR устанавливаются равными значениям трёх младших битов частного I — I2, I1, I0;
- если полученная разность d больше 64, то бит SWR.C2 устанавливается в 1 (это говорит, что в ST(0) получен пока только частичный остаток, не удовлетворяющий требованию «остаток < делителя», и нужно повторить обращение к команде FPREM1 для получения истинного остатка); переменной n присваивается значение из диапазона 32–63; выполняется деление I = (ST(0) / ST(1)) / 2(d – n), затем округление I путём усечения к ближайшему меньшему целому; в ST(0) записывается значение, равное ST(0) – (ST(1) · I · $2^{(d-n)}$).

Команда FPREM1 используется для точного деления чисел в соответствии с требованиями стандарта на вычисления с плавающей запя-

той IEEE 754, согласно которому величина остатка должна быть меньше половины модуля (делителя). Во всем остальном принцип её работы аналогичен таковому для команды FPREM1. Отличие команды FPREM1 от FPREM в том, что характеристика значения в ST(0) не может быть уменьшена более чем на 63 при однократном выполнении инструкции. Если команде удаётся получить конечный остаток, который меньше чем делитель в ST(1), то выполнение команды считается завершённым. Об этом можно судить по нулевому состоянию бита C2 в регистре SWR. В противном случае C2 устанавливается в 1, и результат в ST(0) является промежуточным остатком. Характеристика частичного остатка в ST(0) (после выполнения FPREM1) будет отличаться от значения в ST(0), которое было до выдачи команды FPREM1, по меньшей мере, на 32.

Для получения частичного остатка программа должна циклически выполнять команду FPREM1, пока бит C2 не очистится. Команду FPREM1 часто используют для уменьшения аргументов периодических функций. При этом в качестве делителя выступает значение π/4. Когда такое понижение завершено, то есть получен конечный (частичный) остаток, особое значение имеют биты C3, C1 и C0 регистра SWR. В них команда FPREM помещает 3 младших бита частного. Их значение позволяет определить один из восьми секторов круга, в котором находится значение исходного угла.

FSCALE

Масштабирование значения в ST(0).

Исходные значения: ST(0) = *x*, ST(1) = *y*. Значение *y* округляется к ближайшему меньшему целому (обозначим это значение *d*) и затем вычисляется выражение $ST(0) = x \cdot 2^d$. Команда не выталкивает из стека регистр ST(1).

Команда FSCALE умножает или делит значения в ST(0) на степень числа два. Значение показателя степени (порядка) находится в регистре ST(1) и представляет собой целое число со знаком, вследствие чего и становится возможным осуществлять такое быстрое умножение/деление. Команда FSCALE может быть применена как обратная для команды FXTRACT. (Разбить последней командой число па мантиссу и порядок, изменить порядок самостоятельно и заново сформировать число.)

Поскольку значение порядка остается в стеке (ST(1)), после команды FSCALE следует добавить команду FSTP или FFREE.

FRNDINT

Округление значения в регистре ST(0) до целого.

Выполняется округление значения в ST(0) в соответствии со значением поля CWR.RC:

- CWR.RC = 00b — до ближайшего целого;
- CWR.RC = 01b — до ближайшего меньшего целого;
- CWR.RC = 10b — до ближайшего большего целого;
- CWR.RC = 11b — с отбрасыванием дробной части числа.

Результат записывается в регистр ST(0). Программист должен самостоятельно устанавливать нужный режим округления. Для этого он может использовать как команды для работы со средой сопроцессора, так и команды чтения/записи регистра CWR (FSTCW и FLDCW).

Команды трансцендентных функций

Сопроцессор имеет ряд команд, предназначенных для вычисления значений тригонометрических функций, таких как синус, косинус, тангенс, арктангенс, а также значений логарифмических и показательных функций. Наличие этих команд значительно облегчает жизнь программисту, вынужденному интенсивно заниматься разработкой вычислительных алгоритмов. Выигрыш налицо. Во-первых, отпадает необходимость самому разрабатывать соответствующие подпрограммы. Во-вторых, точность результатов выполнения трансцендентных команд очень высока.

Значения аргументов в командах, вычисляющих результат тригонометрических функций, должны задаваться в радианах.

Команды для вычисления тригонометрических функций

`FSIN`

Вычисление значения синуса угла, заданного в радианах.

Если значение x в регистре ST(0) находится в диапазоне $-2^{63} \le x \le +2^{63}$, то присвоить значения CWR.C2 = 0, ST(0) = sin$x$. В обратном случае оставить регистр в вершине стека без изменений и установить в 1 бит CWR.C2. Необходимо следить за тем, чтобы величина угла x, помещаемая в вершину стека сопроцессора, находилась в границах: $-2^{63} \le x \le +2^{63}$. Для этого нужно проверять состояние бита C2 регистра SWR после выполнения команды. Если C2 = 0, то значение исходного операнда находилось в указанном диапазоне. Если операнд вне диапазона, то необходимо циклически (FPREM, FPREM1) вычитать значение 2π, пока значение в ST(0) не станет корректным.

`FCOS`

Вычисление косинуса угла, заданного в радианах.

Значение угла в радианах x ($-2^{63} \le x \le +2^{63}$) хранится в регистре

ST(0). Команда преобразует число из ST(0) в значение косинуса и записывает его обратно в регистр ST(0).

Необходимо следить за тем, чтобы величина угла x в радианах, предварительно помещаемая в вершину стека сопроцессора, находилась в границах: $-2^{63} \le x \le +2^{63}$. Контроль осуществляется по выходному состоянию бита C2 регистра SWR. Если C2 = 0, то значение операнда принадлежит этому диапазону.

FSINCOS

Вычисление значений синуса и косинуса угла, заданного в радианах.

Если значение x находится в диапазоне $-2^{63} \le x \le +2^{63}$, то присвоить значения CWR.C2 = 0; ST(0) = sin$x$; CWR.TOP = CWR.TOP – 1; ST(0) = cos$x$. Если x не принадлежит указанному диапазону, оставить регистр в вершине стека без изменений и установить в единицу бит CWR.C2. Конечный результат сохраняется в регистрах: ST(0) = cos$x$, ST(1) = sin$x$. Необходимо следить за тем, чтобы величина угла x, помещаемая в вершину стека сопроцессора, находилась в означенных выше границах. Для этого нужно проверять состояние бита C2 регистра SWR. Если C2 = 0, то значение операнда находится в диапазоне $|x| < 2^{63}$. Если операнд вне диапазона, то следует циклически вычитать значение 2π либо использовать FPREM с делителем 2π, пока значение в ST(0) не станет корректным.

FPTAN

Вычисление частичного тангенса угла в радианах из регистра ST(0).

Если значение в ST(0) находится в диапазоне $-2^{63}\ldots2^{63}$, бит C2 сбрасывается, вычисляется тангенс, результат записывается в ST(0), а в стек помещается вещественная единица. Окончательное содержимое регистров: ST(0) = 1,0, ST(1) — тангенс угла. Если значение в ST(0) не принадлежит диапазону $-2^{63}\ldots2^{63}$, бит C2 устанавливается в 1. Если исходный операнд в ST(0) выходит за границы диапазона $-2^{63}\ldots2^{63}$, необходимо вычесть из исходного значения $2\pi N$, где N — множитель, достаточный для нормирования значения в ST(0).

В ранних моделях процессоров (8087, 80287) диапазон допустимых значений был значительно уже — от 0 до $\pi/4$. Если операнд не попадал в диапазон, то его требовалось втискивать в эти рамки, используя команду FPREM с делителем (модулем) равным $\pi/4$.

Зачем нужна единица в стеке? Этот шаг введен в алгоритм работы команды для обеспечения совместимости с младшими моделями процессоров. В них команда FPTAN размещала в стеке два значения, с помощью которых можно было вычислить остальные тригонометрические функции, такие как sin, cos, ctg. Соответствующие команды для

вычисления sin и cos появились лишь в процессоре i386. Что же касается программ для 8087 и 80287, то здесь определение значений функций sin и cos производилось с помощью формул, компонентами которых были два числа, получаемые в результате вычисления тангенса нужного угла. В программах для процессоров старше i386 включение единицы также полезно, например, для вычисления по известным формулам значений других тригонометрических функций. Так, для расчета котангенса можно после FPTAN использовать команду инверсного деления FDIVR.

Заполнение регистра ST(0) требует от программиста действий по контролю содержимого стека для предотвращения его переполнения.

FPATAN

Вычисление частичного арктангенса угла в радианах.

Исходные операнды: значение x в регистре ST(0); значение y в регистре ST(1). Команда вычисляет значение арктангенса частного x/y; выталкивает значения x и y из стека сопроцессора; записывает результат вычисления арктангенса в вершину стека — регистр ST(0).

Для процессора 80287 исходные значения x и y должны отвечать условию $0 \le |y| \le |x| < +\infty$. Если это не так, необходимо использовать следующие формулы приведения:

- $\mathrm{arctg}(x) = -\mathrm{arctg}(-x)$;
- $\mathrm{arctg}(x) = \pi/2 - \mathrm{arctg}(1/x)$.

Для более старших моделей процессоров ограничений нет. Значение угла в команде FPATAN задается в радианах.

Команды для вычисления логарифмов и степеней

F2XM1

Вычисление $2^x - 1$.

Команда F2XM1 вычисляет значение $2^x - 1$, а результат заносит в вершину стека сопроцессора. Исходный операнд (значение степени x в диапазоне $-1 \le x \le 1$, для младших моделей 8087 и 80287 условие более жёсткое — $0 \le x \le 0{,}5$) должен быть предварительно помещен в регистр ST(0). Если значение x выходит за рамки указанного диапазона, результат операции не определён. Команда может быть использована при вычисления y^x для произвольного положительного x. Тогда изменение степени осуществляется по формуле $x^y = 2^{y\log_2 x}$.

FYL2XP1

Вычисление выражения $y\log_2(x + 1)$.

Команда FYL2XP1 вычисляет значение выражения $y\log_2(x + 1)$ и

помещает результат в регистр ST(0). Операнд *y* хранится в ST(1), *x* — в ST(0). Значение в ST(1) должно находиться в диапазоне от –∞ до +∞, а в ST(0) — в диапазоне от $-\left(1-\frac{\sqrt{2}}{2}\right)$ до $\left(1-\frac{\sqrt{2}}{2}\right)$. Если значение в ST(0) лежит вне указанного диапазона, то результат операции не определён. Команда обеспечивает более высокую точность по сравнению с командой FYL2X при вычислении логарифмов чисел, очень близких к 1.

FYL2X

Вычисление выражения $y\log_2 x$.

Команда FYL2X вычисляет значение выражения $y\log_2 x$. Значение *y* хранится в ST(1), *x* — в ST(0). Команда помещает результат вычисления в регистр ST(0). Операнд *x* должен быть положительным, иначе возникает исключение недействительной операции.

Команда рассчитана на применение при вычислении логарифмов с произвольным положительным основанием. При этом делается преобразование $\log_b x = \frac{\log_2 x}{\log_2 b}$.

Команды управления сопроцессором

В эту группу входят команды, предназначенные для общего управления работой сопроцессора. Они позволяют работать с управляющими регистрами сопроцессора. Описание этих и других команд приведено в главе 17 Учебника («Архитектура и программирование сопроцессора»).

Содержание работы

1. Ознакомиться с теоретическим материалом.
2. В соответствии со своим вариантом вычислить значение выражения с помощью TASM, используя команды сопроцессора. Все переменные считать вещественными размером 4 байта.
3. Включить в отчёт все необходимые листинги программ.

Пример решения задачи

Вычислить значение выражения: $\frac{4a}{cb+18}$.

Напишем программу, решающую поставленную задачу, на TASM. Пусть значения переменных задаются в сегменте данных в форме ве-

щественных чисел.

```
MASM
MODEL SMALL
.STACK 100h
.DATA
        a   DD 1.71
        b   DD 2.37
        c   DD -0.83
        res DD ?
.CODE
main PROC
        MOV AX, @DATA
        MOV DS, AX

        .486
        FINIT                    ;инициализация сопроцессора
        FLD   a                  ;ST(0)=a
        MOV   res, DWORD PTR 4   ;res=4
        FIMUL res                ;ST(0)=4a
        FLD   c                  ;ST(0)=c ST(1)=4a
        FMUL  b                  ;ST(0)=cb
        MOV   res, DWORD PTR 18  ;res=18
        FIADD res                ;ST(0)=cd+18
        FDIVP                    ;ST(0)=(4a)/(cb+18)
        FST   res                ;res=(4a)/(cb+18)

        MOV AX, 4C00h
        INT 21h
main ENDP
END main
```

Варианты заданий

1. $\dfrac{2c-d+\sqrt{23a}}{\dfrac{a}{4}-1}$
2. $\dfrac{c+4d-\sqrt{12}}{1-\dfrac{a}{2}}$
3. $\dfrac{-2c+82d}{\operatorname{tg}\left(\dfrac{a}{4}-1\right)}$
4. $\dfrac{\lg(2c-a)+d-152}{\dfrac{a}{4}+c}$
5. $\dfrac{\operatorname{tg}\left(a+\dfrac{c}{4}\right)-12d}{ab-1}$
6. $\dfrac{-2c-\sin\dfrac{a}{d}+53}{\dfrac{a}{4}-b}$
7. $\dfrac{2c-\lg\dfrac{d}{4}}{a^2-1}$
8. $\dfrac{\operatorname{tg}c-23d}{2a-1}$
9. $\dfrac{2c-23d}{\lg\left(1-\dfrac{a}{4}\right)}$

10. $\dfrac{4c+d-1}{c-\operatorname{tg}\dfrac{a}{2}}$

11. $\dfrac{2c-\sqrt{42d}}{c+a-1}$

12. $\dfrac{\sqrt{\dfrac{25}{c}}-d+2}{d+a-1}$

13. $\dfrac{\operatorname{arctg}\left(c-\dfrac{d}{2}\right)}{2a-1}$

14. $\dfrac{4\lg c-\dfrac{d}{2}+23}{a^2-1}$

15. $\dfrac{c\operatorname{tg}(b+23)}{\dfrac{a}{2}-4d-1}$

16. $\dfrac{\dfrac{c}{d}+\ln\dfrac{3a}{2}}{c-a+1}$

17. $\dfrac{2c+51\lg d}{d-a-1}$

18. $\dfrac{2c+\ln\dfrac{d}{4}+23}{a^2-1}$

19. $\dfrac{42c-\dfrac{d}{2}+1}{a^2-\ln(b-5)}$

20. $\dfrac{\dfrac{\operatorname{arctg}\,2c}{d}+2}{d-a-1}$

21. $\dfrac{\operatorname{arctg}\dfrac{12}{c}+73}{a^2-1}$

22. $\dfrac{\dfrac{2c}{a}-d^2}{d+\operatorname{tg}(a-1)}$

23. $\dfrac{\sqrt{\dfrac{53}{a}}+d-4a}{1+ac}$

24. $\dfrac{\sqrt{15a}+b-\dfrac{a}{4}}{cd-1}$

25. $\dfrac{-\dfrac{25}{a}+c-\operatorname{tg}b}{1+\dfrac{cd}{2}}$

Контрольные вопросы

1. Организация программной модели математического сопроцессора.
2. В чём состоит принцип работы с регистрами стека сопроцессора?
3. Как представляются в двоичном виде вещественные числа?
4. Какие специальные численные значения выделяются среди численных значений, обрабатываемых сопроцессором?
5. Классификация команд сопроцессора.
6. Как сопроцессор выполняет математические операции?
7. Как и для чего осуществляется доступ к управляющим регистрам сопроцессора?
8. Какие исключительные ситуации могут возникать при выполнении команд сопроцессора?
9. Как выполняется отладка программ, включающих команды сопроцессора, средствами TASM?

Лабораторная работа № 6
Команды передачи управления

Цель работы: изучить команды передачи управления и особенности их применения для реализации алгоритмов разветвляющейся и циклической структуры.

Основные понятия

Команды передачи управления можно подразделить на следующие группы:

- команды безусловной передачи управления;
- команды условного перехода;
- команды управления циклами.

Команды безусловной передачи управления

JMP target

```
11101011    disp 8                          oper: 8
11101001    disp 16/32                      oper: 16/32
11111111    mod 100 r/m                     oper: 16/32
11101010    disp 16/32   seg 16             oper: 16:16/32
11111111    mod 101 r/m                     oper: 16:16/32
```

Безусловный переход к другой точке потока команд.

Возможные варианты задания операнда target:

- короткий относительный переход — значение disp трактуется как знаковое и является смещением перехода относительно следующей за JMP команды в сегменте кода, т.е. адрес цели равен IP/EIP+disp 8/16/32;
- близкий абсолютный косвенный переход — значением target является 16/32-разрядный регистр или ячейка памяти, содержащая адрес перехода в текущем сегменте кода;
- дальний абсолютный переход — значением target является полный адрес в виде четырёх- или шестиразрядного указателя;
- дальний абсолютный косвенный переход — значением target является адрес ячейки памяти размером 32/48 бит, содержащей компоненты адреса перехода.

```
JMP SHORT m1   ;короткий переход в пределах -127...+128 байт
относительно
               ;следующей за JMP команды
JMP m2         ;переход на m2 в пределах текущего сегмента
JMP BX         ;переход по адресу CS:[BX]
JMP adr_m1     ;переход на метку adr_m1 в пределах текущего
сегмента
JMP FAR Target ;межсегментный переход по адресу Target
```

Команды условного перехода

JXX met

Переход на метку met, если выполнено условие XX.

Команды условного перехода, в зависимости от своей мнемоники анализируют флаги, и если проверяемое условие истинно, производится переход к ячейке, обозначенной операндом. Если проверяемое условие ложно, выполняется следующая за командой перехода команда. Идентификатор met преобразуется ассемблером в непосредственные значения короткого перехода, которые во время выполнения добавляются к текущему значению IP/EIP. Мнемоника команд условного перехода представлена в табл. 6.1. Логические условия «больше» и «меньше» («Greater» и «Less») относятся к сравнениям целочисленных значений со знаком, а «выше» и «ниже» («Above» и «Below») — к сравнениям целочисленных значений без знака.

Команды условного перехода используются для проверки различных условий, возникающих в ходе выполнения программы. Многие команды формируют признаки результатов своей работы в регистре FLAGS/EFLAGS. На анализе флагов этого регистра построена логика условных переходов всех команд, кроме JCXZ/JECXZ. В отличие от других команд условного перехода она выполняет переход в случае если значение регистра CX/ECX = 0. Её удобно использовать для обхода цикла, если счётчик цикла равен нулю.

Команды управления циклами

LOOP target

```
11100010    disp 8                          oper: 8
```

LOOPE/LOOPZ target

```
11100001    disp 8                          oper: 8
```

LOOPNE/LOOPNZ target

```
11100000    disp 8                          oper: 8
```

Управление циклом со счётчиком в регистре CX/ECX.

Команда LOOP выполняет декремент содержимого регистра CX/ECX, затем если CX/ECX = 0, то передаёт управление следующей команде, иначе передаёт управление команде, метка которой указана в качестве операнда. Смещение метки относительно текущего значения регистра IP/EIP должно быть в диапазоне –128…+127 байт.

Команды LOOPE/LOOPZ и LOOPNE/LOOPNZ выполняют декремент содержимого регистра CX/ECX, затем если CX/ECX = 0, то передаёт управление следующей команде, иначе проверяет флаг ZF. Если ZF = 0, то для команды LOOPE/LOOPZ это означает выход из цикла, а для команд LOOPE/LOOPNZ — переход к началу цикла. Если ZF = 1,

Таблица 6.1

Мнемоники команд условного перехода

Мнемокод	Машинный код		Значения флагов, по которым происходит переход	Условие перехода
	при 8-разрядном смещении	при 16/32-разрядном смещении		
JO	01110000	00001111 10000000	OF = 1	переполнение
JNO	01110001	00001111 10000001	OF = 0	нет переполнения
JB/JC/JNAE	01110010	00001111 10000010	CF = 1	ниже
JAE/JNB /JNC	01110011	00001111 10000011	CF = 0	не ниже
JE/JZ	01110100	00001111 10000100	ZF = 1	равно
JNZ/JNE	01110101	00001111 10000101	ZF = 0	не равно
JBE/JNA	01110110	00001111 10000110	CF = 1 или ZF = 1	не выше
JA/JNBE	01110111	00001111 10000111	CF = 0 и ZF = 0	выше
JS	01111000	00001111 10001000	SF = 1	отрицательно
JNS	01111001	00001111 10001001	SF = 0	не отрицательно
JP/JPE	01111010	00001111 10001010	PF = 1	паритет
JPO/JNP	01111011	00001111 10001011	PF = 0	не паритет
JL/JNGE	01111100	00001111 10001100	SF ≠ OF	меньше
JGE/JNL	01111101	00001111 10001101	SF = OF	не меньше
JLE/JNG	01111110	00001111 10001110	ZF = 1 или SF ≠ OF	не больше
JG/JNLE	01111111	00001111 10001111	ZF = 0 и SF = OF	больше
JCXZ /JECXZ	11100011			

то для команд LOOPE/LOOPZ происходит переход к началу цикла, а для команд LOOPE/LOOPNZ — выход из цикла.

Команда LOOP используется для организации цикла со счётчиком. Количество повторений цикла задаётся значением в регистре CX/ECX перед входом в последовательность команд, составляющих тело цикла.

Более подробно о командах передачи управления изложено в главе 10 Учебника («Команды передачи управления»).

С помощью команд передачи управления можно разрабатывать программы нелинейной разветвляющейся и циклической структуры. Особенности разработки таких структур описаны в главе 11 Учебни-

ка («Программирование типовых управляющих структур»).

Содержание работы

1. Ознакомиться с теоретическим материалом.
2. Исследовать выполнение всех команд передачи управления с любыми возможными типами операндов.
3. Решить следующую задачу: в соответствии со своим вариантом вычислить и сохранить в стеке *n* значений переменной *X* при фиксированном значении *b*, изменяя значение *a* с шагом *h*.
4. Решить поставленную задачу с помощью DEBUG. Все переменные считать целыми числами размером 1 байт.
5. Решить поставленную задачу с помощью TASM. Переменную *n* считать целым беззнаковым числом, остальные переменные — вещественными числами.
6. В отчёт включить все необходимые листинги.

Пример решения задачи

Вычислить и сохранить в стеке *n* значений переменной *X* при фиксированном значении *b*, изменяя значение *a* с шагом *h*:

$$X = \begin{cases} a - b, & \text{если } a > b \\ -1, & \text{если } a = b \\ a + b, & \text{если } a < b \end{cases}$$

Решим задачу с помощью DEBUG. Разместим в начале кода значения переменных *a*, *b*, *h*, и *n*, значение *X* после вычисления будет в регистре BX.

```
-a
17C0:0100 db   2          ;значение a
17C0:0101 db   3          ;значение b
17C0:0102 db   1          ;значение h
17C0:0103 db   3          ;значение n
17C0:0104 XOR  CX, CX     ;CX=0
17C0:0106 MOV  CL, [103] ;CL=n
17C0:010A MOV  AL, [101] ;AL=b
17C0:010D CBW             ;AX=b
17C0:010E MOV  DX, AX     ;DX=b
17C0:0110 MOV  AL, [100] ;AL=a
17C0:0113 CBW             ;AX=a
17C0:0114 MOV  BX, AX     ;BX=a
17C0:0116 CMP  BX, DX     ;сравнение a и b
17C0:0117 JE   120        ;переход, если a=b
17C0:011A JG   125        ;переход, если a>b
                          ;действия в случае a<b
17C0:011C ADD  BX, DX     ;BX=a+b
17C0:011E JMP  127        ;переход
```

```
                        ;действия в случае a=b
17C0:0120 MOV  BX, -1   ;BX=-1
17C0:0123 JMP  127      ;переход
                        ;действия в случае a>b
17C0:0125 SUB  BX, DX   ;BX=a-b
17C0:0127 PUSH BX       ;занесение значения BX в стек
17C0:0128 ADD  AL, [102]
17C0:012C ADC  AH, 0    ;AX=a+h
17C0:012F LOOP 114      ;CX=CX-1, если CX>0, то переход
17C0:0131 NOP
17C0:0132
```

Напишем программу для TASM. Зададим в сегменте данных начальное значения переменных *a*, *b*, *h* и *n*.

```
MASM
MODEL SMALL
.STACK 100h
.DATA
        a DD -1.5
        b DD 0.5
        h DD 0.5
        n DB 7
        X DD ?
.CODE
main PROC
        MOV AX, @DATA
        MOV DS, AX

        XOR CX, CX
        MOV CL, n  ;CX=количество повторений цикла
        .486
        FLD a      ;ST(0)=a
cycle:
        FCOM  b  ;сравнение чисел a и b
        FSTSW AX
        SAHF     ;загрузка в регистр флагов значений регистров
сопроцессора
        JP m3 ;если числа несравнимы, то переход на m3
        JC m1 ;если a<b, то переход на m1
        JZ m2 ;если a=b, то переход на m2
;действия в случае a>b
        FLD   b            ;ST(0)=b ST(1)=a
        FSUBR ST(0), ST(1) ;ST(0)=a-b
        FSTP  X            ;X=a-b ST(0)=a
        JMP   endc         ;переход на endc
m1: ;действия в случае a<b
        FLD  b            ;ST(0)=b ST(1)=a
        FADD ST(0), ST(1) ;ST(0)=a+b
        FSTP X            ;X=a-b ST(0)=a
        JMP  endc         ;переход на endc
m2: ;действия в случае a=b
```

```
        FLD1       ;ST(0)=1 ST(1)=a
        FCHS       ;ST(0)=-1
        FSTP X     ;X=-1 ST(0)=a
        JMP  endc ;переход на endc
m3: ;действия в случае, если a и b несравнимы
        MOV X, 0 ;занесём в качестве результата значение 0
endc: ;действия, которые выполняются в цикле
      ;при любых значениях a и b
        PUSH X     ;занесение значения X на текущем шаге в стек
        FADD h     ;ST(0)=a+h — новое значение a
        LOOP cycle ;CX=CX-1, если CX>0, то переход на cycle,
                   ;иначе — выход
        MOV AX, 4C00h
        INT 21h
main ENDP
END main
```

Варианты заданий

1. $X = \begin{cases} 2 + \frac{b}{a}, & \text{если } a > b \\ 25, & \text{если } a = b \\ \frac{a - 5}{b}, & \text{если } a < b \end{cases}$

2. $X = \begin{cases} \frac{b}{a} - 2, & \text{если } a > b \\ 2, & \text{если } a = b \\ \frac{a^3 + 1}{b}, & \text{если } a < b \end{cases}$

3. $X = \begin{cases} \frac{b}{a} + 5, & \text{если } a > b \\ -5, & \text{если } a = b \\ \frac{a^2}{b} - 1, & \text{если } a < b \end{cases}$

4. $X = \begin{cases} \frac{a}{b} + 10, & \text{если } a > b \\ -51, & \text{если } a = b \\ b - \frac{4}{a}, & \text{если } a < b \end{cases}$

5. $X = \begin{cases} \frac{a}{b} - 1, & \text{если } a > b \\ -25, & \text{если } a = b \\ \frac{b^3 - 5}{a}, & \text{если } a < b \end{cases}$

6. $X = \begin{cases} \frac{a}{b} - 1, & \text{если } a > b \\ -25, & \text{если } a = b \\ \frac{b^3 - 5}{a}, & \text{если } a < b \end{cases}$

7. $X = \begin{cases} \frac{52b}{a} + b, & \text{если } a > b \\ -125, & \text{если } a = b \\ \frac{a - 5}{b}, & \text{если } a < b \end{cases}$

8. $X = \begin{cases} b - \frac{1}{a}, & \text{если } a > b \\ 255, & \text{если } a = b \\ \frac{a - 5}{b}, & \text{если } a < b \end{cases}$

9. $X = \begin{cases} 1 - \frac{b}{a}, & \text{если } a > b \\ -10, & \text{если } a = b \\ \frac{a - 5}{b}, & \text{если } a < b \end{cases}$

10. $X = \begin{cases} \frac{a}{b} + 31, & \text{если } a > b \\ -25, & \text{если } a = b \\ \frac{5b - 1}{a}, & \text{если } a < b \end{cases}$

11. $X = \begin{cases} \frac{2+b}{a}, & \text{если } a > b \\ -2, & \text{если } a = b \\ \frac{a-5}{b}, & \text{если } a < b \end{cases}$

12. $X = \begin{cases} \frac{b}{a} + 1, & \text{если } a > b \\ 25, & \text{если } a = b \\ \frac{a^3-5}{b}, & \text{если } a < b \end{cases}$

13. $X = \begin{cases} \frac{b}{a} + 61, & \text{если } a > b \\ -5, & \text{если } a = b \\ 1 - \frac{a}{b}, & \text{если } a < b \end{cases}$

14. $X = \begin{cases} \frac{a}{b} + 1, & \text{если } a > b \\ -2, & \text{если } a = b \\ 1 - \frac{b}{a}, & \text{если } a < b \end{cases}$

15. $X = \begin{cases} \frac{3a-5}{b}, & \text{если } a > b \\ -4, & \text{если } a = b \\ a^2 + \frac{b}{a}, & \text{если } a < b \end{cases}$

16. $X = \begin{cases} \frac{b}{a} - 1, & \text{если } a > b \\ -295, & \text{если } a = b \\ \frac{a-235}{b}, & \text{если } a < b \end{cases}$

17. $X = \begin{cases} \frac{2a}{b} + 1, & \text{если } a > b \\ -445, & \text{если } a = b \\ \frac{b+5}{a}, & \text{если } a < b \end{cases}$

18. $X = \begin{cases} \frac{a}{b} + 1, & \text{если } a > b \\ a + 25, & \text{если } a = b \\ b - \frac{2}{a}, & \text{если } a < b \end{cases}$

19. $X = \begin{cases} \frac{b}{a} + 10, & \text{если } a > b \\ 3425, & \text{если } a = b \\ \frac{2a-5}{b}, & \text{если } a < b \end{cases}$

20. $X = \begin{cases} a - \frac{b}{a}, & \text{если } a > b \\ -a, & \text{если } a = b \\ a - \frac{1}{b}, & \text{если } a < b \end{cases}$

21. $X = \begin{cases} \frac{b+1}{a}, & \text{если } a > b \\ -b, & \text{если } a = b \\ \frac{a-5}{b}, & \text{если } a < b \end{cases}$

22. $X = \begin{cases} \frac{a}{b} - 1, & \text{если } a > b \\ 25 - a, & \text{если } a = b \\ \frac{b-5}{a}, & \text{если } a < b \end{cases}$

23. $X = \begin{cases} \frac{b}{a} + 2, & \text{если } a > b \\ -11, & \text{если } a = b \\ \frac{a-8}{b}, & \text{если } a < b \end{cases}$

24. $X = \begin{cases} \frac{a}{b} + 2, & \text{если } a > b \\ 8, & \text{если } a = b \\ \frac{b-9}{a}, & \text{если } a < b \end{cases}$

25. $X = \begin{cases} \frac{b+5}{a}, & \text{если } a > b \\ -5, & \text{если } a = b \\ 1 - \frac{a}{b}, & \text{если } a < b \end{cases}$

Контрольные вопросы

1. Классификация команд передачи управления.
2. Как выполняются команды условного и безусловного перехода?
3. Какие виды переходов поддерживаются командой JMP?
4. Что такое «метка»?
5. Как организовать на ассемблере выполнение разветвляющихся и циклических конструкций? вложенных конструкций?

Лабораторная работа № 7
Команды поразрядной обработки данных

Цель работы: изучить команды поразрядной обработки данных и особенности их применения.

Основные понятия

Команды поразрядной обработки данных можно подразделить на следующие группы:

- логические команды;
- команды сдвига;
- команды циклического сдвига;
- команды для работы с отдельными битами.

Логические команды

NOT oper

```
1111011w    mod 010 r/m                     oper: 8/16/32
```

Инвертирование всех битов операнда.

Команда изменяет значения всех битов операнда. Операнд может располагаться в регистре или в памяти.

AND dest, src

```
0010010w    data 8/16/32                    dest: AL/AX/EAX
100000sw    mod 100 r/m     data 8/16/32    oper: 8/16/32
001000dw    mod reg r/m                     oper: 8/16/32
```

Логическое И.

Команда выполняет операцию логического И над соответствующими парами битов операндов. Биты операнда dest устанавливаются только в том случае, если в обоих операндах они равны 1. Эта команда удобна для принудительного сброса определённых битов операнда.

TEST dest, src

```
1010100w    data 8/16/32                    dest: AL/AX/EAX
```

```
1111011w    mod 000 r/m      data 8/16/32     oper: 8/16/32
1010010w    mod reg r/m                       oper: 8/16/32
```

Логическое сравнение.

Команда аналогична команде AND, но изменяет только значения флагов.

OR dest, src

```
0000110w    data 8/16/32                      dest: AL/AX/EAX
100000sw    mod 001 r/m      data 8/16/32     oper: 8/16/32
000010dw    mod reg r/m                       oper: 8/16/32
```

Логическое ИЛИ.

Команда выполняет операцию логического ИЛИ над соответствующими парами битов операндов. Биты операнда dest сбрасываются только в том случае, если в обоих операндах они равны 0. Эта команда удобна для установки определённых битов операнда.

XOR dest, src

```
0011010w    data 8/16/32                      dest: AL/AX/EAX
100000sw    mod 110 r/m      data 8/16/32     oper: 8/16/32
001100dw    mod reg r/m                       oper: 8/16/32
```

Логическое исключающее ИЛИ.

Команда выполняет операцию логического исключающего ИЛИ над соответствующими парами битов операндов. Биты операнда dest устанавливаются, если значения соответствующих битов операндов различны, и сбрасываются в противном случае. Эта команда удобна для инвертирования или сравнения определённых битов операндов.

Команды сдвига

SHL oper, cnt

```
1101000w    mod 100 r/m                       oper: 8/16/32
                                              cnt : 1
1101001w    mod 100 r/m                       oper: 8/16/32
                                              cnt : CL
1100000w    mod 100 r/m   data 8              oper: 8/16/32
```

Логический сдвиг операнда влево.

SHR oper, cnt

```
1101000w    mod 101 r/m                       oper: 8/16/32
                                              cnt : 1
1101001w    mod 101 r/m                       oper: 8/16/32
                                              cnt : CL
1100000w    mod 101 r/m   data 8              oper: 8/16/32
```

Логический сдвиг операнда вправо.

Эти команды сдвигают все биты операнда влево/вправо на количество разрядов, указанное операндом cnt, при этом выдвигаемый слева/справа бит становится значением флага CF. Одновременно с проти-

воположной стороны вдвигается бит с нулевым значением. Для всех команд сдвига значение операнда cnt ограничено диапазоном от 0 до 31. Если текущее значение флага CF и значение старшего бита результата различны, то флаг OF устанавливается, иначе — сбрасывается.

Команды SHL/SHR удобно использовать для умножения/деления беззнакового числа на степени двойки.

SHLD dest, src, cnt (386)

```
00001111    10100100     data 8              oper: 16/32
                                             cnt : 8
00001111    10100101                         oper: 16/32
                                             cnt : CL
```

Сдвиг двойного слова влево.

SHRD dest, src, cnt (386)

```
00001111    10101100     data 8              oper: 16/32
                                             cnt : 8
00001111    10101101                         oper: 16/32
                                             cnt : CL
```

Сдвиг двойного слова вправо.

Эти команды аналогичны командам SHL/SHR, но сдвигается число dest:src. При этом операнд src не изменяется. Первый операнд может находиться в регистре или в памяти, второй — только в регистре.

SAL oper, cnt

```
1101000w    mod 100 r/m                      oper: 8/16/32
                                             cnt : 1
1101001w    mod 100 r/m                      oper: 8/16/32
                                             cnt : CL
1100000w    mod 100 r/m   data 8             oper: 8/16/32
```

Арифметический сдвиг операнда влево.

SAR oper, cnt

```
1101000w    mod 111 r/m                      oper: 8/16/32
                                             cnt : 1
1101001w    mod 111 r/m                      oper: 8/16/32
                                             cnt : CL
1100000w    mod 111 r/m   data 8             oper: 8/16/32
```

Арифметический сдвиг операнда вправо.

Эти команды выполняют сдвиг всех битов операнда влево/вправо на количество разрядов, указанное операндом cnt, при этом выдвигаемый слева/справа бит становится значением флага CF. Одновременно для команды SAL справа в операнд вдвигается нулевой бит. Для команды SAR по мере сдвига вправо освобождающиеся места заполняются значением знакового бита.

Команды циклического сдвига

ROL oper, cnt

```
1101000w    mod 000 r/m                     oper: 8/16/32
                                            cnt : 1
1101001w    mod 000 r/m                     oper: 8/16/32
                                            cnt : CL
1100000w    mod 000 r/m   data 8            oper: 8/16/32
```

Циклический сдвиг операнда влево.

ROR oper, cnt

```
1101000w    mod 001 r/m                     oper: 8/16/32
                                            cnt : 1
1101001w    mod 001 r/m                     oper: 8/16/32
                                            cnt : CL
1100000w    mod 001 r/m   data 8            oper: 8/16/32
```

Циклический сдвиг операнда вправо.

Каждый раз при циклическом сдвиге разрядов операнда влево/вправо его старший/младший выдвигаемый бит вдвигается в операнд справа/слева и становится одновременно значением младшего/старшего бита операнда и флага CF.

RCL oper, cnt

```
1101000w    mod 010 r/m                     oper: 8/16/32
                                            cnt : 1
1101001w    mod 010 r/m                     oper: 8/16/32
                                            cnt : CL
1100000w    mod 010 r/m   data 8            oper: 8/16/32
```

Циклический сдвиг операнда влево через флаг переноса CF.

RCR oper, cnt

```
1101000w    mod 011 r/m                     oper: 8/16/32
                                            cnt : 1
1101001w    mod 011 r/m                     oper: 8/16/32
                                            cnt : CL
1100000w    mod 011 r/m   data 8            oper: 8/16/32
```

Циклический сдвиг операнда вправо через флаг переноса CF.

Каждый раз при циклическом сдвиге разрядов операнда влево/вправо его старший/младший бит становится значением флага CF. Старое содержимое CF вдвигается в операнд справа/слева и становится значением его младшего/старшего бита.

Команды для работы с отдельными битами

BSF dest, src (386)

```
00001111    10111100     mod reg r/m          oper: 16/32
```

Определение номера позиции в операнде src самого младшего единичного бита.

BSR dest, src (386)

```
00001111    10111101     mod reg r/m          oper: 16/32
```

Определение номера позиции самого старшего единичного бита в операнде src.

Эти команды просматривают биты операнда src, находящегося в регистре или памяти, начиная с младшего/старшего. Номер позиции первого единичного бита слева записывается в регистр dest, флаг ZF сбрасывается. Если единичных битов нет, то флаг устанавливается, а значение в dest не определено.

BT src, index (386)

```
00001111    10100011     mod reg r/m           oper: 16/32
00001111    10111010     mod 100 r/m   data 8   oper: 16/32
```

Определение значения конкретного бита в операнде src.

BTC src, index (386)

```
00001111    10111011     mod reg r/m           oper: 16/32
00001111    10111010     mod 111 r/m   data 8   oper: 16/32
```

Определение и дополнение значения заданного бита в операнде src.

BTR src, index (386)

```
00001111    10110011     mod reg r/m           oper: 16/32
00001111    10111010     mod 110 r/m   data 8   oper: 16/32
```

Определение значения заданного бита в операнде src и сброс этого бита.

BTS src, index (386)

```
00001111    10101011     mod reg r/m           oper: 16/32
00001111    10111010     mod 101 r/m   data 8   oper: 16/32
```

Определение значения заданного бита в операнде src и установка его в 1.

Эти команды проверяют бит операнда src с номером index. После выполнения команды в флаг CF заносится значение проверяемого бита, а сам бит при этом не изменяется/инвертируется/сбрасывается/устанавливается. Операнд src должен находиться в регистре или памяти, а index — регистр или непосредственные данные.

Более подробно о командах поразрядной обработки данных изложено в главе 9 Учебника («Логические команды и команды сдвига»).

Содержание работы

1. Ознакомиться с теоретическим материалом.
2. Исследовать выполнение всех команд поразрядной обработки данных с любыми возможными типами операндов.
3. В соответствии со своим вариантом решить поставленные за-

дачи с помощью DEBUG и TASM, используя команды поразрядной обработки данных.

4. В отчёт включить все необходимые листинги.

Задачи к выполнению лабораторной работы

1. В слове установить 0, 2 и 5 биты, сбросить 7, 11 и 13 биты, инвертировать 3, 8 и 15 биты. Остальные биты оставить без изменения.
2. Двухразрядное упакованное двоично-десятичное число представить ASCII-кодами его цифр.
3. Двузначное десятичное число, представленное ASCII-кодами своих цифр, представить двоично-десятичным кодом в упакованном формате.
4. Обменять вторую и третью тетраду в слове.
5. Организовать беззнаковое деление слова на 8.
6. В слове обнулить крайний справа единичный бит: например, двоичное число 01011000 преобразуется в 01010000, а число 00001111 — в число 00001110 (обнулённый бит подчёркнут).
7. В слове установить крайний справа нулевой бит: например, двоичное число 01010111 преобразуется в 01011111, а число 00001111 — в число 00011111 (установленный бит подчёркнут).
8. В слове оставить установленным только крайний справа единичный бит: например, двоичное число 01011000 преобразуется в 00001000, а число 00001111 — в число 00000001 (установленный бит подчёркнут).
9. В слове выделить крайний справа нулевой бит: например, двоичное число 01011000 преобразуется в 00000001, а число 00001111 — в число 00010000 (выделенный бит подчёркнут).
10. В слове выделить крайние справа нулевые биты: например, двоичное число 01011000 преобразуется в 00000111 (выделенные биты подчёркнуты), а число 00001111 — в нуль 00000000, т. к. справа не было нулевых битов.

Пример решения задачи

Выполнить сдвиг двойного слова, находящегося в двух регистрах, на два разряда влево.

Пусть заданное двойное слово находится в паре регистров AX:BX.

Решим задачу в DEBUG. Выполним логический сдвиг регистра BX влево, а затем циклический сдвиг с учётом флага переноса регистра AX. Эти действия выполним два раза.

```
-a
17BA:0100 SHL BX, 1
```

```
17BA:0102 RCL AX, 1
17BA:0104 SHL BX, 1
17BA:0106 RCL AX, 1
17BA:0108 NOP
```

Напишем программу для TASM. Используем команду SHLD.

```
MASM
MODEL SMALL
.STACK 100h
.CODE
main PROC
        .386
        SHLD AX, BX, 2
        SHL  BX, 2

        MOV AX, 4C00h
        INT 21h
main ENDP
END main
```

Варианты заданий

Номер варианта	Номера задач	Номер варианта	Номера задач
1	1, 6	14	3, 9
2	1 ,7	15	3, 10
3	1 ,8	16	4, 6
4	1, 9	17	4 ,7
5	1, 10	18	4 ,8
6	2, 6	19	4, 9
7	2 ,7	20	4, 10
8	2 ,8	21	5, 6
9	2, 9	22	5 ,7
10	2, 10	23	5 ,8
11	3, 6	24	5, 9
12	3, 7	25	5, 10
13	3, 8		

Контрольные вопросы

1. Классификация команд поразрядной обработки данных.
2. Какие способы адресации возможны для операндов в командах поразрядной обработки данных?
3. Что нужно сделать, чтобы установить, сбросить или инвертировать значение определённого бита регистра?
4. Как можно выполнить сдвиг целого числа, если оно целиком не помещается ни в один из доступных регистров?

Приложение

Значения поля КОП для некоторых двухоперандных команд ассемблера

Мнемоника команды ассемблера	Код операции	
	без непосредственной адресации операндов	при непосредственной адресации операнда
MOV	100010 101000 – MOV AX, [disp]	1011 1100011 000
XCHG	10010 1000011	——
ADD	000000	0000010 100000 000
ADC	000100	0001010 100000 010
SUB	001010	0010110 100000 101
SBB	000110	0001110 100000 011
CMP	001110	0011110 100000 111
AND	001000	0010010 100000 100
TEST	1010010	1010100 1111011 000
OR	000010	0000110 100000 001
XOR	001100	0011010 100000 110

Значения поля reg

Код	Регистр
000	AL/AX
001	CL/CX
010	DL/DX
011	BL/BX
100	AH/SP
101	CH/BP
110	DH/SI
111	BH/DI

Значения поля r/m

Код	Адрес
000	BX + SI + disp
001	BX + DI + disp
010	BP + SI + disp
011	BP + DI + disp
100	SI + disp
101	DI + disp
110*	BP + disp
111	BX + disp

* при mod=00 адрес [disp], а не [BP]

Библиографический список

1. *Юров, В. И.* Assembler: учебник для вузов. — 2-е изд. / В. И. Юров. — СПб.: Питер, 2006. — 637 с.: ил. — ISBN 5-94723-581-1

2. *Юров, В. И.* Assembler: практикум. — 2-е изд. / В. И. Юров. — СПб.: Питер, 2006. — 399 с.: ил. — ISBN 5-94723-671-0

3. *Юров, В. И.* Assembler. Специальный справочник. — 2-е изд. / В. И. Юров. — СПб.: Питер, 2005. — 412 с.: ил. — ISBN 5-469-00003-6

4. *Корсунов, Н. И.* Организация электронных вычислительных машин и систем: методические указания к выполнению лабораторных работ / сост.: Н. И. Корсунов, М. С. Розанов, В. В. Румбешт. — Белгород: Изд-во БГТУ, 2006. — Ч. 1. — 103 с.

5. *Голубь, Н. Г.* Искусство программирования на Ассемблере: лекции и упражнения. — 2-е изд., испр. и доп. / Н. Г. Голубь — СПб.: ООО «ДиаСофтЮП». 2002. — 656 с. — ISBN 5-9377 6-3

Учебное издание

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ
Основы программирования
на языке Ассемблер

Методические указания к выполнению лабораторных работ
для студентов специальности 230105 — Программное обеспечение
вычислительной техники и автоматизированных систем

Составители: Гарибов Александр Искендерович
Куценко Дмитрий Александрович

Подписано в печать 08.12.09. Формат 60×84/16. Усл.печ.л. 5,4. Уч.-изд.л. 5,8.
Тираж 121 экз. Заказ Цена
Отпечатано в Белгородском государственном технологическом университете
им. В. Г. Шухова
308012, г. Белгород, ул. Костюкова, 46