МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южный федеральный университет»
Инженерно-технологическая академия

С.В. Скороход
В.В. Селянкин
С. Н. Дроздов
Д.П. Калачев
Н.Ш. Хусаинов

Основы программирования микропроцессоров Intel для встраиваемых систем

Учебное пособие

Для студентов бакалаврской подготовки
по направлениям 02.03.03, 09.03.04

Таганрог
Издательство Южного федерального университета
2016

УДК 004.4 (075, 8)
ББК 32.973.26-04я73
С 753

Печатается по решению редакционно-издательского совета Южного федерального университета

Рецензенты:

профессор кафедры ПМиИТ Таганрогского института управления и экономики, доктор технических наук ***В. П. Карелин;***

доцент кафедры САиТ Института компьютерных технологий и информационной безопасности Южного федерального университета, кандидат технических наук ***А. С. Свиридов***

Скороход, С. В.

С 753 Основы программирования микропроцессоров Intel для встраиваемых систем : учебное пособие / **Скороход С.В., Селянкин В. В., Дроздов С. Н., Калачев Д. П., Хусаинов Н. Ш.**. : Южный федеральный университет. – Таганрог: Издательство Южного федерального университета, 2016. – 82 с.

ISBN 978-5-9275-2223-1

Рассматриваются вопросы программирования процессоров Intel на языке ассемблера для встраиваемых систем. Учебное пособие состоит из семи разделов. В первом разделе рассматривается последовательность разработки простейшей арифметической программы. Второй раздел посвящен разработке циклических программ. В третьем разделе обсуждаются вопросы использования логических команд. В четвертом разделе излагаются технология обработки символьной информации. Пятый раздел предназначен для изучения операций с двоично-десятичной арифметикой. Шестой раздел посвящен применению подпрограмм. В седьмом разделе обсуждаются вопросы программирования арифметического сопроцессора для реализации вычислений с вещественными числами. Пособие предназначено для студентов, обучающихся по направлениям 02.03.03, 09.04.04.

ISBN 978-5-9275-2223-1 УДК 004.4 (075, 8)
ББК 32.973.26-04я73

Введение

Встраиваемая вычислительная система – это специализированная вычислительная система, непосредственно взаимодействующая с объектом контроля или управления и объединенная с ним единой конструкцией. Взаимодействие с обслуживаемым объектом выполняется посредством специализированных датчиков и исполнительных устройств.

Особенностями встраиваемых систем являются:

- непосредственное подключение к объекту;
- работа в режиме реального времени;
- поддержка алгоритмов автоматического регулирования;
- повышенные требования к надежности и безопасности функционирования;
- жесткие условия эксплуатации (широкий диапазон температур, помехи и т.п.);
- использование в малогабаритных, автономных и переносных системах (предъявляет требования невысокого энергопотребления, малых габаритов, минимального числа вспомогательных элементов).

Как правило, встроенная система является частью более крупной системы или встраивается непосредственно в объект управления. Встроенные системы – это системы «глубоко интегрированные» с объектами физического мира. Их элементы практически всегда ограничены по ресурсам. Это системы длительного жизненного цикла, часто автономные. Масштаб этих систем по размерам и сложности меняется в очень широких пределах. Эти системы рассчитаны на непрофессиональных пользователей и вместе с тем часто выполняют критически важные функции.

Встраиваемые вычислительные системы можно классифицировать:

- по области применения и назначению;
- по различному соотношению информационных и управляющих функций, т.е. система преимущественно информационная (система сбора данных) или управляющая система автоматического управления);
- по пространственной локализации аппаратных блоков:
  а) пространственно локализованные,
  б) пространственно рассредоточенные;
- по различному соотношению вычислительной (обработка данных) и коммуникационной (функция ввода/вывода данных) составляющих;
- по степени участия человека:
  а) автоматические системы – системы, в которых оператор выполняет только функции начальной настройки и оперативной корректировки параметров и режимов работы системы. Функции сбора данных,

передачи и исполнения команд управления, оперативной выработки команд управления происходят без участия человека;

б) автоматизированные системы – системы, в которых оператор частично или в полном объеме обеспечивает оперативную обработку данных и формирование команд управления исполнительными устройствами (например, телеуправление);

- по организации обработки данных и вычислений (централизованные или децентрализованные);
- по распараллеливанию на уровне задач и/или функций между физическими/логическими модулями системы.

Диапазон применения встроенных вычислительных систем очень велик. В него попадают и простейшие устройства уровня домашнего таймера, и сложнейшие распределенные иерархические системы, управляющие критически важными объектами на огромных территориях.

- Телекоммуникационные системы, сетевое оборудование (коммутаторы, маршрутизаторы, ADSL-модемы и т.п.).
- Бытовая электроника (сотовые телефоны, КПК, игровые консоли, цифровые фотоаппараты, электрочайники, микроволновые печи, посудомоечные машины и пр.).
- Современное медицинское и спортивное оборудование.
- Транспортная автоматика (от автомобильных до авиационных систем), авионика, системы управления городским дорожным движением.
- Системы телемеханики (системы управления наружным освещением, контроля и учета электроэнергии и других энергоресурсов, управления и мониторинга энергообъектов).
- Системы мониторинга, навигации, слежения, бортовые системы для военных и космических применений.
- «Умный дом» («интеллектуальное здание») на основе технологий сенсорных сетей.

Проектируя встроенную вычислительную систему, разработчик всегда создает специализированную вычислительную систему независимо от степени соотношения готовых и заново создаваемых решений. В сферу его анализа попадают все уровни организации системы. Он имеет дело не с созданием приложения в готовой операционной среде при наличии мощных и удобных инструментальных средств, а с созданием новой специализированной ВС в условиях жестких ограничений самого разного плана. Это обуславливает необходимость наиболее полного использования возможностей имеющихся аппаратных ресурсов и особенностей их архитектуры, что возможно только при применении языка ассемблера специализированного процессора, на базе которого создается встроенная система.

Компания Intel предлагает к использованию во встраиваемых решениях процессоры семейства Atom, Core 2 Duo, Core i3/i5/i7, а также одноядерные и двухъядерные процессоры семейства Intel Atom с частотами 1.1 и 1.6 ГГц для решения задач начального и среднего уровня. Настоящее пособие посвящено изучению основ программирования на языке ассемблера семейства процессоров Intel в реальном режиме, применяемом при создании и программировании встроенных систем.

Пособие разработано в рамках выполнения базовой части государственного задания в сфере научной деятельности (проект № 3442 "Информационно-алгоритмическое обеспечение систем цифрового управления, автономной высокоточной навигации и технического зрения для перспективных летательных аппаратов: разработка теоретических основ проектирования, алгоритмов, способов эффективной и надежной программной реализации, использование высокопроизводительной вычислительной инфраструктуры для экспериментального моделирования").

1. Разработка линейных арифметических программ

Регистры процессора

Регистры процессора разделяются на следующие группы [1]:

- регистры общего назначения;
- индексные регистры и указатели;
- сегментные регистры;
- регистр флагов;
- указатель команд.

Регистры общего назначения (РОН) включают 4 универсальных регистра, оптимизированных для выполнения каких-либо операций:

- AX (EAX, RAX) – регистр-аккумулятор. Предназначен для хранения операнда и записи результата выполнения команды;
- BX (EBX, RBX) – базовый регистр. Может использоваться для формирования базового адреса данных в памяти;
- CX (ECX, RCX) – регистр-счетчик. Используется как счетчик при организации циклов и сдвигов.
- DX (EDX, RDX) – дополнительный регистр. Используется для хранения промежуточных данных.

Соответствие разрядности и обозначений регистров изображено в табл. 1.1.

Таблица 1.1

<table>
<tr><td>63</td><td>31</td><td>15</td><td colspan="2">7 0</td></tr>
<tr><td rowspan="2">RAX</td><td rowspan="2">EAX</td><td colspan="3">AX</td></tr>
<tr><td>AH</td><td colspan="2">AL</td></tr>
<tr><td rowspan="2">RBX</td><td rowspan="2">EBX</td><td colspan="3">BX</td></tr>
<tr><td>BH</td><td colspan="2">BL</td></tr>
<tr><td rowspan="2">RCX</td><td rowspan="2">ECX</td><td colspan="3">CX</td></tr>
<tr><td>CH</td><td colspan="2">CL</td></tr>
<tr><td rowspan="2">RDX</td><td rowspan="2">EDX</td><td colspan="3">DX</td></tr>
<tr><td>DH</td><td colspan="2">DL</td></tr>
</table>

В процессорах архитектуры x64 имеются дополнительные 64-разрядные РОН R8 – R15 и их младшие байты [2]:

- R8D – R15D – 32-разрядные;
- R8W – R15W – 16-разрядные;
- R8B – R15B – 8-разрядные.

Индексные регистры и указатели предназначены для формирования адреса в памяти и включают четыре регистра [3]:

- BP (EBP, RBP) – указатель базы. Используется для хранения некоторого начального (базового) адреса;
- SP (ESP, RSP) – указатель стека. Указывает на вершину стека;
- DI (EDI, RDI), SI (ESI, RSI) – индексные регистры. Содержат смещение относительного базового адреса.

Соответствие разрядности и обозначений индексных регистров изображено в табл. 1.2.

Таблица 1.2

63	31	15 0
RBP	EBP	BP
RSP	ESP	SP
RSI	ESI	SI
RDI	EDI	DI

Регистр флагов содержит биты условий, называемых флагами, сигнализирующих о состоянии процессора после последней выполненной команды [4]. Перечень наиболее важных флагов приведен в табл. 1.3.

Таблица 1.3

Бит №	Имя флага	Назначение
0	CF	Флаг переноса. Устанавливает, был ли перенос из старшего разряда или заем в старший разряд при выполнении арифметических операций
1	PF	Флаг четности. Устанавливается в 1, если результат операции содержит четное количество единиц
4	AF	Флаг вспомогательного переноса. Используется в операциях над упакованными двоично-десятичными числами
6	ZF	Флаг нуля. Устанавливается 1, если результат операции равен 0
7	SF	Флаг знака. Показывает знак результата операции (1– отрицательный, 0 – положительный)
8	TF	Флаг трассировки. Обеспечивает возможность работы процессора в пошаговом режиме
9	IF	Флаг внешних прерываний. Если IF=1, прерывание разрешается, IF=0 – блокируется
10	DF	Флаг направления. Используется командами обработки строк. DF=1 – прямое направление (от меньших адресов к большим). DF=0 – обратное направление
12	OF	Флаг переполнения. Устанавливается в 1, если произошел выход результата операции за пределы допустимого диапазона значений

Сегментные регистры являются 16-разрядными. Адресуемый сегментным регистром участок памяти называется текущим сегментом. Сегмент всегда выровнен по границе параграфа (т.е. находится по адресу, кратному 16). Поэтому содержимое сегментного регистра всегда предполагает наличие четырех нулевых битов в младших разрядах, которые в регистре не хранятся [5].

Список сегментных регистров приведен в табл. 1.4.

Таблица 1.4

Сегментный регистр	Назначение
CS	Регистр сегмента кода. Содержит начальный адрес сегмента кода
DS	Регистр сегмента данных
SS	Регистр сегмента стека
ES, FS, GS	Дополнительные сегментные регистры

Указатель команд IP (EIP, RIP) содержит смещение команды, которая должна быть выполнена. Пара регистров CS+IP содержит адрес следующей команды [6].

Режимы адресации

Режимы адресации операндов команд процессора приведены в табл. 1.5. Здесь использованы следующие сокращения: R – регистр, V – переменная, C – константа [14].

Таблица 1.5

Название	Обозначение	Содержание	Пример
Регистровая прямая	R	Операнд находится в регистре	mov AX, SI переслать содержимое регистра SI в регистр AX
Непосред-ственная	C	Непосредственный операнд (константа) присутствует в ко-манде	mov AX, 093Ah занести константу 093Ah в ре-гистр AX
Прямая	V + C или V - C	Исполнительный адрес операнда присутствует в команде	mov BX, WW+2 переслать в BX слово памяти, отстоящее от переменной с именем WW на 2 байта

Окончание табл. 1.5

Название	Обозначение	Содержание	Пример
Косвенная регистровая	[R] где R – BP, BX, SI, DI	Регистр содержит адрес операнда	mov [BX], CL переслать содержимое регистра CL по адресу, находящемуся в регистре BX
Косвенная регистровая относитель-ная	V[R] , C[R] , [R+V] , [R+C] , где R – SI,DI (индексная) BX,BP (базовая)	Адрес операнда вычисляется как сумма содержимого реги-стра и смещения	mov M[BX], CL переслать содержимое регистра CL по адресу,отстоящему от переменной M на BX байт
Индексно-базовая	[BR][IR] , V[BR][IR] , [BR][IR]C ,где IR – SI, DI, BR – BX, BP	Адрес операнда вычисляется как сумма содержимых базового и индексного реги-стров и возможного смещения	mov [BX][SI]3, AL переслать содержимое регистра AL по адресу, сумме регистров BX, SI и константы 3

Структура простейшей программы

Ниже приведена схема простейшей программы на языке ассемблера с использованием модели памяти small, в которой допускается наличие единственного сегмента кода и единственного сегмента данных [14].

```
        model small
        stack 100h
        dataseg
        . . .           ; описание данных
        codeseg
Start:
        startupcode
        . . .           ; код программы
Quit:   exitcode 0
        end Start
```

Для выделения памяти в сегменте данных под константы, переменные, массивы используются директивы определения данных [7]:

```
        db                      ; 1 байт
        dw                      ; 2 байта
<Имя>   dd   <выражение>        ; 4 байта
        dq                      ; 8 байт
        dt                      ; 10 байт
```

<Имя> – идентификатор, обозначающий смещение данных относительно начала сегмента.

<Выражение> определяет значения, заносимые в выделяемую область памяти. Виды выражений и их примеры приведены в табл. 1.6.

Таблица 1.6

Вид выражения	Пример
Константа	c4 db 17
Знак вопроса – отсутствие значения	ef1 dw ?
Несколько констант, разделенных запятыми	ef2 db 11, 14, 25, 17
Повторитель dup	fl1 db 10 dup (?) ; 10 байт без значений
Символьную константу	fl3 db ‘+’
Символьную строку	fl4 db ‘abcde’ ; 5 байт с кодами символов
Последовательная комбинация предыдущих видов	fl5 db ‘abcde’,10 dup (0), 0Ah, ’!’

Перечень простейших арифметических команд приведен в табл. 1.7 [14].

Таблица 1.7

Команда	Изменение флагов OSZAPC	Действие
mov DST, SRC	------	Пересылка, DST←SRC
xchg DST, SRC	------	Обмен, DST←SRC
add DST, SRC	xxxxxx	Сложение, DST←DST+SRC
adc DST, SRC	xxxxxx	Сложение с переносом, DST←DST+SRC+CF
inc OPND	xxxxx-	Увеличить на единицу, OPND←OPND+1
sub DST, SRC	xxxxxx	Вычитание, DST←DST-SRC
sbb DST, SRC	xxxxxx	Вычитание с заемом, DST←DST-SRC-CF
dec OPND	xxxxx-	Уменьшение на единицу, OPND←OPND-1
neg OPND	xxxxxx	Изменение знака, OPND←0-OPND
rcl DST, CONT	x----x	Циклический сдвиг влево через CF

Окончание табл. 1.7

Команда	Изменение флагов OSZAPC	Действие
rcr DST, CONT	x----x	Циклический сдвиг вправо через CF
rol DST, CONT	x----x	Циклический сдвиг влево
ror DST, CONT	x----x	Циклический сдвиг вправо
sal DST, CONT	xxxuxx	Арифметический сдвиг влево
sar DST, CONT	xxxuxx	Арифметический сдвиг вправо
shl DST, CONT	xxxuxx	Логический сдвиг влево
shr DST, CONT	xxxuxx	Логический сдвиг вправо
mul SRC	x----x	Беззнаковое умножение AX на регистр или ячейку памяти (результат – в DX:AX)
imul SRC	x----x	Знаковое умножение AX на регистр или ячейку памяти (результат – в DX:AX)
div SRC	------	Беззнаковое деление пары регистров DX:AX на регистр или ячейку памяти (результат: AX– частное, DX– остаток)
idiv SRC	------	Знаковое деление пары регистров DX:AX на регистр или ячейку памяти (результат: AX– частное, DX– остаток)

Пример составления программы

Дана формула: X = 3A + (B + 5)/2 – C – 1.

A, B, C, X – целые знаковые числа длиной в слово. Написать программу, реализующую данную формулу.

Распишем формулу по отдельным операциям в виде табл. 1.8 [14].

Таблица 1.8

AX ← A	Занести A в регистр AX
AX ← 2*(AX)	2A в AX
AX ← (AX) + A	3A в AX
BX ← B	B в BX
BX ← 5 + (BX)	B+5 в BX
BX ← (BX)/2	(B+5)/2 в BX
AX ← (BX) + (AX)	3A+(B+5)/2 в AX
AX ← (AX) – C	3A+(B+5)/2–C в AX
AX ← (AX) – 1	3A+(B+5)/2–C–1 в AX
X ← (AX)	3A+(B+5)/2–C–1 в X

Напишем программу, реализующую последовательность шагов, описанных в табл. 1.8 [14].

```
model  SMALL
stack  100h
dataseg
A       dw     10
B       dw     20
C       dw      5
X       dw      ?
codeseg
startupcode
mov AX, A         ;значение A в регистр AX
sal    AX, 1      ; 2A в AX
add     AX, A     ; 3A в AX
mov     BX, B     ; B в BX
add     BX, 5     ; B+5 в BX
sar     BX, 1     ; (B+5)/2 в BX
add     AX, BX    ; 3A+(B+5)/2 в AX
sub     AX, C     ; 3A+(B+5)/2-C в AX
dec     AX        ; 3A+(B+5)/2-C-1 в AX
mov     X, AX     ; 3A+(B+5)/2-C-1 в X
;Конец работы
QUIT:   exitcode  0
end
```

Задания для самостоятельного выполнения

Разработать программу, реализующую указанную формулу, исполнить программу с несколькими наборами исходных данных, проверить правильность результатов.

1. X = –4A + (B + C)/ 4 + 2
2. X = (A – B) / 4 – 2C + 5
3. X = (A/2 + B) / 4 + C – 1
4. X = (7A – 2B – 100) / 2 + C
5. X = – (C + 2A + 4B + 8)
6. X = –A/2 + 4(B + 1) – 3C
7. X = A – 5(B – 2C) + 2
8. X = 6C + (B – C + 1)/2
9. X=-7(C – A/4) + 3 B – 5
10. X=5(A – B) + C/2 +1
11. X=(–3A – 5B + 7C)/4
12. X= –9A + 3B/4 – C/2

13. X= 7(A + 3B – 20) +C/2
14. X=3(A + B –5C/4) + 2
15. X=25 – 7(A – 2 B) +C/4

Контрольные вопросы

1. Из каких полей состоит строка программы на ассемблере ?
2. Какие поля обязательны, а какие можно опустить ?
3. Назначение директив stack**,** dataseg и codeseg.
4. Назначение макрокоманд startupcode, exitcode.
5. Назначение директив db, dw.
6. Назначение оператора dup в директивах db, dw.
7. Назначение директивы end.
8. В чем различие между командами:
   mov AX, BX;
   mov AX, [BX];
   mov [AX], BX ?
9. Какая директива завершает текст программы ?
10. В чем различие между командой mov A, 1 и директивой A dw 1?
11. Как изменится содержимое AL и флагов после выполнения команд
    shr AL, 1 ;
    sar AL, 1 ;
    shl AL, 1 ;
    sal AL, 1 ,
    если AL = F2h ?

2. Разработка циклических программ

Команды проверки условий и переходов

Программирование разветвляющихся вычислений на языке ассемблера связано с использованием команд условных переходов. Каждая из этих команд проверяет некоторый код условия или их комбинацию и в случае выполнения условия выполняет переход по указанному адресу. При невыполнении условия управление передается следующей команде программы.

Для выработки кода условия можно воспользоваться командами cmp**,** test.

Команда cmp op1, op2 выполняет вычитание второго операнда из первого без сохранения результата. Результатом команды являются флаги процессора, устанавливаемые по значению полученной разности.

Команда test op1, op2 выполняет поразрядную конъюнкцию над парой своих операндов без запоминания результата. Как и в предыдущем случае, результатом команды являются флаги процессора, устанавливаемые по значению полученной конъюнкции.

Кроме того, коды условия вырабатываются арифметическими и логическими командами.

Команды условных переходов по значению флагов и регистров приведены в табл. 2.1 [8].

Таблица 2.1

Мнемокод	Аналог	Проверяемые флаги (условие перехода)	Используется для организации перехода, если...
jc	jb	CF=1	есть перенос
jnc	jnb	CF=0	нет переноса
jz	je	ZF=1	результат=0
jnz	jne	ZF=1	результат<>0
js	–	SF=1	результат <0
jns	–	SF=0	результат >=0
jo	–	OF=1	есть переполнение
jno	–	OF=0	нет переполнения
jp	jpe	PF=1	в результате четное число единиц
jn	jpo	PF=0	в результате нечетное число единиц
jcxz	–	CX=0	CX(счетчик цикла)=0

Команды переходов, используемые при сравнении беззнаковых чисел, приведены в табл. 2.2 [9].

Таблица 2.2

Мнемокод	Аналог	Проверяемые флаги (условие перехода)	Используется для организации перехода, если...
jb	jnae, jc	CF=1	первый операнд меньше второго
jnb	jae, jnc	CF=0	первый операнд больше или равен второму
jbe	jna	CF or (ZF = 1)	первый операнд меньше или равен второму
jnbe	ja	CF or (ZF = 0)	первый операнд больше второго

Команды переходов, используемые при сравнении знаковых чисел, приведены в табл. 2.3 [10].

Таблица 2.3

Мнемокод	Аналог	Проверяемые флаги (условие перехода)	Используется для организации перехода, если...
jl	jnge	SF ⊕ OF = 1	первый операнд меньше второго
jnl	jge	SF ⊕ OF = 0	первый операнд больше или равен второму
jle	jng	(SF ⊕ OF) or ZF = 1	первый операнд меньше или равен второму
jnle	jg	(SF ⊕ OF) or ZF = 1	первый операнд больше второго

Команда безусловного перехода jmp выполняет переход к заданной ячейке без проверки каких-либо условий.

Условные и безусловные переходы разделяются на прямые и косвенные [11]. Если в команде задается метка команды, на которую надо перейти, то переход является прямым:

```
 L:   mov AX, BX
. . .
      jmp L              ; переход на метку L.
```

Если в команде задается регистр или адрес ячейки памяти, то переход является косвенным. Процессор перейдет к команде, адрес которой содержится в регистре или заданной ячейке:

```
jle   BX   ; переход к команде, адрес которой содержится в BX
```

```
jle     [BX] ; регистр содержит адрес ячейки памяти
             ; в сегменте данных, которая содержит адрес команды,
             ; к которой будет выполнен переход.
```

Команды для организации циклов

Для организации циклических вычислений можно использовать команды условного перехода, однако в случае организации цикла по счетчику удобнее воспользоваться командой loop и ее разновидностями, приведенными в табл. 2.4. Следует помнить, что эти команды выполняют короткий переход, т.е. имеют ограниченное смещение перехода в диапазоне от –128 до 127 байтов [12].

Таблица 2.4

Мнемокод	Аналог	Выполняемая последовательность действий
loop	–	CX = CX–1; переход, если CX<>0
loopz	loope	CX = CX–1; переход, если (CX<>0) and (ZF=1)
loopnz	loopne	CX = CX–1; переход, если (CX<>0) and (ZF=0)

Одно из важнейших применений циклов – обработка массивов [14]. В языке ассемблера существует возможность описывать только одномерные массивы. Для этого используют директивы описания данных db**,** dw**,** dd и др. При выполнении цикла часто требуется при каждом новом повторении обращаться к следующему элементу массива. Фактически это означает необходимость увеличения адреса текущего элемента. Очевидно, это можно сделать, если для обращения к элементу массива применять индексный или базовый режим адресации.

Важным моментом при программировании циклов является проверка условия окончания цикла. Есть несколько возможных вариантов организации такой проверки. Если число повторений заранее известно, то можно в одном из регистров (лучше в CX) вести счетчик повторений, тогда условием окончания будет достижение счетчиком заданного значения. Иногда вместо счетчика удобнее использовать значение адреса обрабатываемого элемента массива, в этом случае за условие окончания следует принять выход адреса за пределы массива.

Следует очень внимательно относиться к выбору конкретного значения счетчика или адреса, при котором заканчивается цикл. Практика программирования показывает, что одним из самых распространенных типов

ошибок в программах является выполнение на одно повторение цикла больше или меньше, чем нужно.

Пример циклической программы

Дан массив из десяти слов, содержащих целые числа. Требуется найти максимальное значение [14].

```
        model  SMALL
        stack  100h

        dataseg
MAX     dw     ?
MASS    dw     10h,20h,30h,5h,40h,15h,20h,70h,35h,34h

        codeseg
        startupcode

         lea    BX, MASS        ; Загрузить адрес массива
         mov    CX, 10          ; Установить счетчик
         mov    AX, [BX]        ; Первый элемент массива в аккумулятор
BEG:    cmp    [BX], AX    ; Сравнить текущий элемент массива с
                           ; максимумом
        jl     NO                  ; он меньше
        mov    AX, [BX]            ; он больше
NO:     add    BX, 2               ; Следующий элемент массива
        loop   BEG         ; Возврат, если счетчик CX не пуст
        mov    MAX, AX

;Конец работы
QUIT:   exitcode  0
        end
```

Задания для самостоятельного выполнения

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

1. Найти количество отрицательных чисел. Массив байтов.
2. Найти суммы всех положительных и отрицательных чисел. Массив слов.
3. Найти среднее арифметическое чисел. Массив слов.
4. Найти количество чисел, больших 10h. Массив слов.
5. Найти сумму абсолютных величин. Массив байтов.
6. Найти количество положительных чисел. Массив слов.
7. Поменять местами пары соседних чисел. Массив слов.

8. Переставить числа в обратном порядке. Массив байтов.
9. Заменить все отрицательные числа нулями. Массив слов.
10. Найти максимальный элемент массива. Массив байтов.
11. Найти минимальный элемент массива. Массив слов.
12. Найти количество четных чисел. Массив байтов.
13. Найти сумму нечетных чисел. Массив слов.
14. Найти сумму отрицательных чисел. Массив слов.
15. Заменить все положительные числа единицами. Массив байтов.

Контрольные вопросы

1. Для чего нужен оператор ptr?
2. В чем отличие команд mov AX, offset MASS и lea AX, MASS?
3. В чем отличие команд mov AX, BX и mov AX, [BX]?
4. В чем отличие команд mov AX, [BP] и mov AX, [BX] ?
5. В чем отличие команд mov AX, [BX+2] и mov AX, [BX]+2 ?
6. В чем отличие команд mov AX, [BX][SI] и mov AX, [SI][BX] ?
7. Для организации каких вычислений служат команды loop**,** loope**,** loopne?
8. Модифицирует ли какие-нибудь регистры команда loop ?
9. Можно ли организовать цикл по счетчику, не используя loop-команды?
10. Можно ли организовать цикл **while** с помощью одной из loop-команд?
11. Что такое прямой и косвенный переходы?
12. Какие команды выполняют переход по флагу нуля?
13. Какие команды выполняют переход по флагу знака?
14. Какие команды выполняют переход по флагу переноса?
15. Какие команды выполняют переход по флагу переполнения?
16. Какие команды выполняют переход при сравнении беззнаковых чисел?
17. Какие команды выполняют переход при сравнении знаковых чисел?
18. Что делает команда jle [BX]?

3. Использование логических команд

Логические команды и команды сдвигов

К логическим командам, или точнее, командам, оперирующим с отдельными битами, можно отнести команды, приведенные в табл. 3.1 [14].

Таблица 3.1

Команда	Флаги OSZAPC	Действие
and DST, SRC	xxxxxx	Поразрядное И, DST←DST & SRC
or DST, SRC	xxxxxx	Поразрядное ИЛИ, DST←DST V SRC
xor DST, SRC	xxxxxx	Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ, DST←DST⊕SRC
test DST, SRC	xxxxxx	Поразрядное И без запоминания результата
not OPND	xxxxx-	Поразрядная инверсия, OPND ← ⌉OPND

Операция сдвига битов целого числа может выполняться несколькими способами [13].

- Логический сдвиг, при котором освобождающиеся разряды заполняются нулями.
- Арифметический сдвиг. При арифметическом сдвиге вправо освобождающиеся разряды заполняются первоначальным значением знакового разряда. Арифметический сдвиг влево эквивалентен логическому сдвигу влево.
- Циклический сдвиг. При его выполнении разряды, перемещаемые за разрядную сетку, помещаются в освобождаемые разряды.
- Циклический сдвиг с переносом. Разряд, перемещаемый за разрядную сетку, помещается в флаг переноса CF, а значение флага CF – в освобождающийся разряд.

При всех видах сдвигов разряд, перемещаемый за пределы разрядной сетки, заносится в флаг переноса CF.

Форматы команд сдвига приведены в табл. 1.7.

Примеры использования логических команд и команд сдвига

Установить 3-й и 0-й биты в регистре AL, остальные биты не изменять:
or AL, 00001001b .

Сбросить 4-й и 6-й биты в регистре AL, остальные биты не изменять:
and AL, 10101111b .

Проинвертировать 4-й и 2-й биты в регистре AL, остальные биты не изменять:
xor AL, 00010100b .

Перейти на метку LAB, если установлен 4-й бит регистра AL, в противном случае продолжить выполнение программы

```
        test AL, 00010000b
        jnz LAB
; продолжаем
        ...
LAB:
```

Подсчитать число единиц в байте – регистре AL

```
    mov CX, 8
    xor BX, BX
LL:     shl AL, 1
    jnc NO
    inc BX
NO:     loop LL
```

Пусть содержимое регистра AX содержит битовые поля со значениями дня, месяца и года в соответствии с табл. 3.2.

Таблица 3.2

Поле	Назначение	Диапазон значений
0–4 бит	День	0 – 31
5–8 бит	Месяц	0 –15
9–15 бит	Год	0 –127 (смещение относительно 1980)

Выделим значения каждого из полей.

```
mov  BX, AX     ; сохраняем копию AX
and  AX, 1Fh    ; в AX остался только день
…
mov  AX, BX     ; восстанавливаем исходное содержимое AX
shr  AX, 5      ; сдвигаем так, чтобы с нулевого бита
                ; начиналось поле месяца
and  AX, Fh     ; в AX остался только месяц
…
mov  AX, BX     ; восстанавливаем исходное содержимое AX
shr  AX, 9      ; сдвигаем так, чтобы с нулевого бита
                ; начиналось поле года
and  AX, 7Fh    ; в AX остался только год
```

Установим значение каждого из полей.

```
mov  BX, Day            ; занесли день в BX
and  AX, 0FFE0h         ; обнуляем день в AX
or AX, BX               ; установили день в AX
…
mov  BX, Month ; занесли месяц в BX
```

```
shl    BX, 5              ; сдвигаем значение месяца в свои биты BX
and    AX, 0FE1Fh         ; обнуляем месяц в AX
or AX, BX                 ; установили месяц в AX
…
mov  BX, Year             ; занесли год в BX
shl    BX, 9              ; сдвигаем значение года в свои биты BX
and    AX, 1FFh           ; обнуляем год в AX
or AX, BX                 ; установили год в AX
```

Пример программы с использованием логических команд

Дан массив из 10 байтов. Все байты имеют нулевые старшие биты. Необходимо каждый байт, содержащий 1 в 0-м бите, дополнить до четного числа единиц установкой 7-го бита байта, каждый байт, содержащий 0 в 0-м бите, дополнить до нечетного числа единиц установкой 7-го бита байта [14].

```
      model  SMALL
      stack  100h

      dataseg
MB      db     04h,07h,11h,23h,04h,38h,3Fh,2Ah,0Dh,34h

      codeseg
      startupcode

      lea    BX, MB       ; BX – текущий адрес массива MB
      mov    CX, 10       ; CX – счетчик числа итераций
BEG:
      mov    AL, [BX]     ; считать очередной байт массива
      test   AL, 00000001b; установлен ли бит 0 ?
      jz     BIT0CLR      ; нет, бит 0 сброшен
                          ; бит 0 установлен
      test   AL, 0FFh     ; четное число единиц ?
      jp     OK           ; да, больше ничего делать не надо
      or     AL, 80h      ; нечетное, дополнить до четного
      jmp    short OK
BIT0CLR:                  ; бит 0 сброшен
      test   AL, 0FFh     ; четное число единиц ?
      jnp    OK           ; нет, больше ничего делать не нужно
      or     AL, 80h      ; четное, дополнить до нечетного
OK:
      mov    [BX], AL     ; записать измененный байт массива

      inc    BX           ; BX<-адрес очередн. элемента массива
```

```
    loop   BEG

;Конец работы
QUIT:   exitcode  0
    end
```

Задания для самостоятельного выполнения

1. Дан массив из 10 байтов. Посчитать количество байтов, в которых сброшены 6 и 4 биты.
2. Дан массив из 8 байтов. Рассматривая его как массив из 64 бит, посчитать количество единиц.
3. Дан массив из 8 байтов. Рассматривая его как массив из 64 бит, посчитать длину самой длинной последовательности единиц.
4. Дан массив из 8 байтов. Рассматривая его как массив логических значений x0 x1 x2 x3 x4 x5 x6 x7 (true – есть ненулевые биты в байте, false – все биты нулевые), вычислить логическую формулу f = (x7&~x6&x3&x1) V (x6&x4&x2&x1&~x0) V (~x7&x6&x3&x1) .
5. Дан массив из 5 байтов. Рассматривая его как массив из 10 тетрад, найти «исключающее или» всех 10 тетрад.
6. Дан массив из 10 байтов. Посчитать количество байтов, в которых сброшены 5 или 1 биты.
7. Рассматривая байт как набор логических значений x7 x6 x5 x4 x3 x2 x1 x0 (true – 1, false – 0), вычислить логическую формулу f = (x7&~x6&x3&x1) V (x6&x4&x2&x1&~x0) V (~x7&x6&x3&x1) .
8. Дан массив из 10 байтов. Посчитать количество байтов, в которых установлены 3 и 7 биты.
9. Дан массив из 10 слов, содержащий даты в формате, заданном табл. 3.2. Найти дату с наименьшим днем.
10. Дан массив из 10 слов, содержащий даты в формате, заданном табл. 3.2. Найти дату с наибольшим месяцем.
11. Дан массив из 10 слов, содержащий даты в формате, заданном табл. 3.2. Найти дату с наибольшим годом.
12. Дан массив из 10 слов, содержащий даты в формате, заданном табл. 3.2. Найти количество дат, приходящихся на первую декаду месяца.
13. Дан массив из 10 слов, содержащий даты в формате, заданном табл. 3.2. Найти количество дат, относящихся к зимним месяцам.
14. Дан массив из 10 слов, содержащий даты в формате, заданном табл. 3.2. Найти количество дат 20-го столетия.
15. Дан массив из 10 слов, содержащий даты в формате, заданном табл. 3.2. Найти самую младшую дату.

Контрольные вопросы

1. В чем отличие команд test DST, SRC и and DST, SRC?
2. Как сбросить 5-й бит переменной – байта BB?
3. Как установить 5-й бит переменной – байта BB?
4. Как проинвертировать 5-й бит переменной – байта BB?
5. Как проверить, установлен ли 5-й бит переменной – байта BB?
6. Как проверить, четным или нечетным является количество установленных бит в переменной – байте BB?
7. Какие флаги условий модифицируются после выполнения команд and, or, xor?
8. В чем основное отличие команд логических и арифметических сдвигов?
9. В чем отличие циклических сдвигов от циклических сдвигов с переносом?
10. Укажите максимальное число двоичных разрядов, на которое можно сдвинуть операнд с помощью одной команды сдвига.
11. В паре строк
    mov CL, 8
    "сдвиг" BL, CL
    какие команды можно подставить вместо "сдвиг", чтобы содержимое BL не изменилось ?
12. В паре строк
    "сдвиг влево" BL, 1
    "сдвиг вправо" BL, 1
    какие команды можно подставить вместо "сдвиг влево" и "сдвиг вправо", чтобы содержимое BL не изменилось?
13. Как извлечь значение битового поля?
14. Как установить значение битового поля?

4. Обработка символьной информации

Ввод/вывод символьной информации

Операционная система DOS предоставляет программисту, работающему на языке ассемблера, большой набор подпрограмм, выполняющих различные полезные действия. Все эти подпрограммы оформлены как подпрограммы обработки прерываний, и для каждой подпрограммы в руководстве по DOS указан номер соответствующего ей прерывания. Для вызова системной подпрограммы следует использовать команду программного прерывания int. Сокращенно принято сами системные подпрограммы называть «прерываниями», хотя это не очень правильно. Некоторые прерывания относятся к ROM BIOS (подсистеме управления вводом/выводом, находящейся в ПЗУ), другие прерывания обслуживаются подпрограммами DOS, загружаемыми в ОЗУ. Особую роль играет прерывание с номером 21h. В зависимости от значения, содержащегося при вызове прерывания в регистре AH, DOS выполняет при этом одну из нескольких десятков подпрограмм, которые принято называть функциями DOS [14].

Для каждого прерывания и каждой функции DOS в руководстве описан набор входных и выходных параметров, передаваемых через регистры, а также перечень возможных ошибок. В данном разделе будут описаны прерывания и функции DOS, относящиеся к работе с клавиатурой и экраном ПЭВМ.

Функция 01h (т.е. прерывание 21h при значении AH=01h) выполняет ввод с клавиатуры одного символа. Если в момент вызова функции в буфере клавиатуры были символы (т.е. были нажатия клавиш до этого), то берется символ из буфера, в противном случае система ждет, пока не будет нажата клавиша. Код введенного символа помещается в регистр AL. Введенный символ отображается на экране (как говорят, выполняется эхо-отображение).

Например, пусть на клавиатуре была нажата клавиша «F». Тогда после вызова:

```
mov ah,01h
int 21h
```

в регистре AL будет содержаться число 46h, которое является кодом буквы F в коде ASCII.

Функция 01h проверяет также, не нажимал ли пользователь в ходе работы программы комбинацию клавиш Ctrl+C (или Ctrl+Break). В этом случае управление передается на подпрограмму обработки Ctrl+C, которая обычно прекращает выполнение программы пользователя.

Если на клавиатуре была нажата одна из клавиш, которым не соответствует никакой код ASCII (будем называть такие клавиши несимвольными; к ним относятся, например, Home, Ins, Page Up, F1, F10, стрелка) или комбинация одной из клавиш Alt, Ctrl, Shift с другой клавишей, то функция 01h возвращает в регистре AL значение 0. В этих случаях следует

еще раз вызвать ту же функцию, тогда будет выдан так называемый расширенный код данной клавиши или комбинации, согласно специальной таблице расширенных кодов.

Функция 08h работает аналогично 01h, за исключением того, что не выполняется эхо-отображение введенного символа.

Функция 07h работает аналогично 01h, за исключением того, что не выполняется эхо-отображение и не проверяется нажатие Ctrl+C.

Функция 06h может выполнять как ввод с клавиатуры, так и вывод на экран. Если в момент вызова регистр DL содержит значение 0FFh, то данная функция выполняет ввод без ожидания. Это означает следующее. Если буфер клавиатуры содержит какие-либо символы (т.е. клавиши ранее нажимались), то флаг нуля ZF сбрасывается в 0, а символ из буфера заносится в AL. Если же буфер пуст (нажатий не было), то устанавливается ZF=1, при этом значение в AL не играет роли. Таким образом, эта функция не ждет, пока будет нажата клавиша, а сразу выдает какой-то результат.

Функция 06h не выполняет эхо-отображения и не проверяет нажатие Ctrl+C.

Функция 0Bh не выполняет ввод символа, а только проверяет, есть ли символы в буфере. Если есть, то устанавливается AL=0ffh, если нет, то – AL=00h. Выполняется также проверка на Ctrl+C.

Функция 0Ah выполняет буферизованный ввод строки с клавиатуры. При этом символы вводятся один за другим, как при многократном применении функции 01h до тех пор, пока не будет введен код 0Dh (код клавиши Enter), завершающий строку. В ходе ввода строки пользователь может редактировать строку и, в частности, использовать BackSpace. При вводе выполняется также проверка на Ctrl+C.

При вызове функции 0Ah требуется, чтобы в регистре DX содержался адрес (в сегменте данных) области памяти (буфера), в которую система поместит введенную строку. В первом байте этого буфера должна быть записана его длина, т.е. максимальное число символов (включая 0Dh), которое можно записать в буфер. Эта длина должна быть, по крайней мере, на 2 меньше, чем число зарезервированных байтов. После окончания ввода строки функция помещает во второй байт буфера действительное число введенных символов (не считая 0Dh), а начиная с третьего байта буфера, размещаются введенные символы. Последним всегда будет код 0Dh.

Например. Пусть требуется ввести строку длиной не более 10 символов. При этом в сегменте данных можно описать буфер так:

```
BUFFER  db  11          ;Первый байт буфера
ENTERED db  (?)         ;Число введенных символов
STRING  db  11 dup (?)  ;Введенные символы
```

Сам же ввод выполняется командами:

```
lea DX,BUFFER   ;Адрес буфера – в DX
```

```
    mov AH,0Ah      ;Номер функции – в AH
    int 21h         ;Вызов функции
```

Пусть при этом пользователь набрал 6 символов: «Hallo!» и нажал клавишу Enter. После вызова функции в байте entered будет число 6, в первых 6 байтах массива STRING будут коды введенных символов, в седьмом байте – код 0Dh, а оставшиеся 4 байта будут иметь неопределенные значения.

Функция 0Ch вначале очищает буфер клавиатуры (т.е. «забывает» предыдущие нажатия клавиш), а потом выполняет любую из функций 01h**,** 06h**,** 07h**,** 08h или 0Ah. Номер этой функции задается в регистре AL. Если задано иное число, то выполняется только очистка буфера. Кроме того, всегда выполняется проверка на Ctrl+C.

Имеются две функции для вывода на экран: одна функция для вывода одного символа, а другая – для вывода строки символов.

Функция 02h выдает в текущую позицию экрана символ, код которого содержится в регистре DL. Известно, что для машин типа IBM PC почти каждому из 256 возможных кодов соответствует какое-то графическое изображение, однако при выводе по функции 02h некоторые коды не выдаются на экран, а служат управляющими. В частности:

- 0Dh (CR) – перевод курсора в начало текущей строки;
- 0Ah (LF) – перевод курсора вниз на 1 строку;
- 08h (BS) – перевод курсора влево на 1 позицию;
- 07h (BEL) – звонок.

Функция 09h выдает, начиная с текущей позиции экрана, строку символов, адрес начала которой (в сегменте данных) содержится в регистре DX. Строка может содержать управляющие символы (CR, LF и т.п.) и, таким образом, на экране занимать несколько строк. Концом выдаваемой строки служит символ «$» (код 24h).

Например. Чтобы вывести на экран с новой строки текст: «Привет! Вот как надо выдавать текст на экран!» – и затем перевести курсор в следующую строку, следует в сегменте данных описать строку:

```
PRIVET    db  0Dh,0Ah
          db  "Привет! Вот как надо выдавать текст на экран!"
          db  0Dh,0Ah,"$"
```

а в сегменте команд записать команды:

```
    lea  DX,PRIVET   ; Адрес строки – в DX
    mov  AH,09h      ; Номер функции – в AH
    int  21h         ; Вызов функции
```

Пример. Ввести строку с клавиатуры, посчитать количество всех десятичных цифр во введенной строке, посчитанные значения вывести на экран [14].

```
    model  SMALL
```

```
        stack  100h

        dataseg
AskCont db 0Ah,0Dh
        db 'Завершить работу – Esc, продолжить – Любая клавиша '
        db '$'
Ask     db 0Ah,0Dh,'Введите строку:','$'
COUNT   db 10 dup( ? ) ; счетчики количества цифр
CIFR    db '0123456789ABCDEF' ; таблица преобразования
                      ; шестнадцатеричных цифр для вывода
INPSTR  db 80, ?, 82 dup( ? ) ; буфер ввода
OUTSTR  db 0Dh,0Ah, ?, ' ', ?, ?, '$' ; буфер вывода

        codeseg
        startupcode

BEGIN:
;Ввод строки
        lea    DX, Ask
        mov    AH, 09h
        int    21h
        lea    DX, INPSTR
        mov    AH, 0Ah
        int    21h

;Обработка
        xor    AX, AX
        lea    BX, INPSTR+2  ;адрес начала введенной строки
        xor    CX, CX
        mov    CL, INPSTR+1  ;кол-во введенных символов строки
BB:
        mov    AL, [BX]      ;очередной символ строки
        cmp    AL, '0'       ;код символа меньше, чем код нуля ?
        jb     NC            ;да, т.е. не цифра
        cmp    AL, '9'       ;код символа больше, чем код девятки ?
        ja     NC            ;да, т.е. не цифра
;символ – десятичная цифра
        sub    AL, '0'       ;получаем дв. значение цифры, т.е.
        mov    SI, AX        ; индекс в массиве счетчиков COUNT
        inc    COUNT[SI]     ;увеличиваем соответств. счетчик
NC:     inc    BX            ;получить очередной символ строки
        loop   BB
```

```
;Вывод результатов
        lea    DX, OUTSTR
        xor    SI, SI        ;Счетчик цифр
OO:     xor    AX, AX
        mov    AL, '0'
        add    AX, SI        ;ASCII-код очередной цифры в SI
        mov    OUTSTR+2, AL  ; в буфер вывода
        mov    AL, COUNT[SI] ;AL<-значение счетчика
                             ; очередной цифры
        mov    CL, 4         ;получаем
        shr    AL, CL        ; в DI
        mov    DI, AX        ; значение старшей шестн. цифры
        mov    AL, CIFR[DI]  ; счетчика преобразуем в ASCII-код
        mov    OUTSTR+4, AL  ;пересылаем в буфер вывода
        mov    AL, COUNT[SI] ;AL<-знач. счетчика очередн. цифры
        and    AL, 0Fh       ;Получаем в DI значение
        mov    DI, AX        ; младшей шестн. цифры счетчика
        mov    AL, CIFR[DI]  ; преобразуем ASCII-код
        mov    OUTSTR+5, AL  ; пересылаем в буфер вывода
        mov    AH, 09h       ;Вывод сформированной в буфере
        int    21h           ; строки
        inc    SI            ;Счетчик очередной цифры
        cmp    SI, 10
        jl     OO

;Запрос на продолжение работы
        lea    DX, AskCont
        mov    AH, 09h
        int    21h
        mov    AH, 08h
        int    21h
        cmp    AL, 27
        je     QUIT
        jmp    BEGIN

;Конец работы
QUIT:   exitcode  0
        end
```

Преобразование десятичных чисел

При вводе десятичное число записано в форме ASCII-числа. Алгоритм преобразования состоит в том, что, начиная с самого правого байта ASCII-числа, выполняется такая последовательность шагов:

1. Устанавливается в 0 левый полубайт каждого байта ASCII-числа.
2. ASCII-цифры умножаются на 1, 10, 100, и результаты складываются.

Например, преобразование числа 2459 выглядит так:

0 * 0 + 2 = 2
2 * 10 + 4 = 24
24 * 10 + 5 = 245
245 * 10 + 9 = 2459

При вводе чисел общего вида нужно учесть 2 момента:

- наличие или отсутствие знака;
- введенное число может не поместиться в отведенную для него разрядную сетку.

Схематично алгоритм такого преобразования выглядит так (для 2-байтового знакового числа):

```
If [ВхБуфер]='-' then begin
        Знак='-';
        Inc(ВхБуфер)
end
else if [ВхБуфер]='+' then begin
        Знак='+';
        Inc(ВхБуфер)
end
else
        Знак='+';
Результат=0;
While ([ВхБуфер]>='0') and ([ВхБуфер]<='9') do begin
        Результат = Результат * 10;
        Результат =  Результат + цифра из буфера;
        If CF=1 then
                goto переполнение;
        Inc(ВхБуфер);
end;
if Знак='-' then
        if Результат > 32768
                goto переполнение;
        else
                Результат = - Результат;
else
```

if Результат > 32767

goto переполнение;

Преобразование при выводе выполняется путем последовательного деления на 10. Остаток – соответствующая цифра. Цифры получаются в последовательности от младших к старшим разрядам.

Для преобразования 2-байтового числа со знаком нужно делить 5 раз (так как число содержит не более 5 цифр). Возможны 2 варианта действий:

- всегда делить 5 раз – могут получиться ведущие нули;
- делить, пока результат не станет равен нулю – тогда нулей не будет.

Команды обработки строк

Цепочечный примитив – это команда, предназначенная для обработки одного элемента строки (массива). Отдельный примитив обрабатывает один элемент строки. В общем случае примитивы работают с двумя областями памяти.

Область, из которой данные поступают на обработку, называется источником. Источник всегда адресуется парой регистров DS:SI. Область, в которую данные помещаются после обработки, называется приемником. Приемник всегда адресуется парой регистров ES:DI.

В некоторых примитивах может использоваться только источник или только приемник.

После выполнения любого из примитивов содержимое индексных регистров DI и SI автоматически увеличивается или уменьшается на одну и ту же величину – величину длины обрабатываемого элемента строки (1, 2 или 4). Направление изменения (увеличение или уменьшение) зависит от значения флага DF и описано в табл. 4.1.

Таблица 4.1

Элемент строки	Флаг DF	
	0	1
Байт	+1	–1
Слово	+2	–2
Двойное слово	+4	–4

Примитивы копирования строк:

movsb – копирование байта
movsw – копирование слова
movsd – копирование двойного слова

} mem[ES : DI] ← mem[DS : SI]

Флаги не модифицируются.

Примитивы сравнения строк:

cmpb – сравнение байт
cmpw – сравнение слов
cmpd – сравнение двойных слов
} mem[ES : DI] - mem[DS : SI]

Флаги модифицируются аналогично команде cmp.

Примитивы сканирования строк:

scasb – сравнение байт
scasw – сравнение слов
scasd – сравнение двойных слов
} (AL, AX, EAX) - mem[ES : DI]

Команда выполняет сравнение содержимого аккумулятора с элементом строки по адресу ES:DI. Устанавливает флаги аналогично cmp.

Примитивы загрузки строк:

lodsb – загрузка байта
lodsw – загрузка слова
lodsd – загрузка двойного слова
} (AL, AX, EAX) ← mem[DS : SI]

Команда выполняет копирование элемента строки из памяти в аккумулятор. Флаги не меняются.

Примитивы выгрузки строк:

stosb – выгрузка байта
stosw – выгрузка слова
stosd – выгрузка двойного слова
} mem[ES : DI] ← (AL, AX, EAX)

Команда выполняет копирование элемента строки из аккумулятора в ячейку памяти. Флаги не меняются.

Префикс повторения обеспечивает выполнение одного цепочечного примитива несколько раз. Количество повторений определяется содержимым регистра CX. Таким образом, префикс повторения определяет цикл из одной команды – цепочечного примитива.

Существуют 3 префикса повторения, описанные в табл. 4.2.

Таблица 4.2

Префикс	Действие	Цепочечный примитив
rep	Выполнять, пока CX<>0	movs, lods, stos
repe, repz	Выполнять, пока CX<>0 и ZF=1	cmps, scas
repne, repnz	Выполнять, пока CX<>0 и ZF=0	cmps, scas

Пример. Подсчет количества слов во фрагменте текста.

```
        dataseg
s1      db    ‘ text  string  for  example  $‘
len     dw    32
        codeseg
        mov AX, @DATA
```

```
        mov DS, AX
        mov ES, AX
        mov CX, len     ; размер строки
        lea DI, s1      ; адрес первого символа строки
        mov AL, ‘ ‘     ; разделитель слов
        xor BX, BX      ; счетчик слов
        cld
next:
        repe scasb      ; пропускаем пробелы
        je exit         ; кроме пробелов ничего нет – закончить
        inc BX          ; нарастить счетчик
        repne scasb; ищем конец слова
        jne exit        ; строка закончилась – закончить
        jmp next
exit:                   ; BX – счетчик слов
```

Задания для самостоятельного выполнения

1. Ввести с клавиатуры строку, состоящую из целых чисел, разделенных пробелами. Найти минимальное число и вывести на экран.
2. Ввести с клавиатуры строку, состоящую из целых чисел, разделенных пробелами. Подсчитать количество чисел строки, которые входят в заданный константами интервал.
3. Ввести с клавиатуры строку, состоящую из целых чисел, разделенных пробелами. Найти количество четных чисел.
4. Ввести с клавиатуры строку, состоящую из целых чисел, разделенных пробелами. Найти количество чисел больших, чем первое число строки.
5. Ввести с клавиатуры строку. Сжать строку, т.е. удалить пробелы и табуляции. Вывести результаты на экран.
6. Ввести с клавиатуры строку. Преобразовать все малые буквы в большие. Вывести результаты на экран.
7. Ввести с клавиатуры строку. Посчитать количество слов в строке. Подумать, что является разделителем слов. Вывести результаты на экран.
8. Ввести с клавиатуры строку. Ввести с клавиатуры коротенькую строку – шаблон. Найти шаблон во введенной строке. Вывести на экран "ДА", если шаблон есть, и "НЕТ", если его нет.
9. Ввести с клавиатуры две строки. Сравнить их. Вывести на экран номер позиции, в которой строки различаются.
10. Ввести с клавиатуры строку. Если она длиннее некоторой величины, то обрезать, если короче – растянуть, вставив нужное число пробелов между словами. Вывести результаты на экран.
11. Ввести с клавиатуры строку, состоящую из нескольких слов. Вывести

каждое слово на экран в отдельной строке, т.е. выдать слова в столбик.

12. Ввести с клавиатуры строку. Переставить в ней символы, поменяв местами первый с последним, второй с предпоследним и т.д. Вывести результаты на экран.
13. Ввести с клавиатуры строку, состоящую из целых чисел, разделенных пробелами. Найти сумму чисел и вывести на экран.
14. Ввести с клавиатуры строку, состоящую из целых чисел, разделенных пробелами. Найти максимальное число и вывести на экран.
15. Ввести с клавиатуры строку. Удалить из нее пробелы так, чтобы осталось по одному пробелу между словами.

Контрольные вопросы

1. Что такое программное прерывание?
2. Какие возможности работы с клавиатурой имеются у программиста?
3. Чем отличаются друг от друга различные функции DOS, выполняющие ввод с клавиатуры?
4. Что такое расширенный код клавиатуры?
5. Можно ли выдать на экран текст «$1.00 = 25.00 rub.», используя функцию DOS 09h?
6. Какие режимы адресации удобно использовать при работе с одномерными массивами?
7. Что означают выражения в поле операндов в строках примера:
   lea BX, INSTR+2
   mov OUTSTR+3, AL ?
8. Как выделить младшую тетраду байта?
9. Как выделить старшую тетраду байта?
10. В чем отличие команд
    lea BX, STR
    mov BX, offset STR ?
11. Как преобразовать десятичное число из символьной формы в двоичную?
12. Как преобразовать целое двоичное число в десятичное символьное представление?
13. Что такое цепочечный примитив?
14. Какие цепочечные примитивы предназначены для копирования и сканирования строк и их действие?
15. Какие цепочечные примитивы предназначены для сравнения строк и их действие?
16. Какие цепочечные примитивы предназначены для загрузки и выгрузки строк и их действие?
17. Какие префиксы повторения используются с цепочечными примитивами и их действие?

5. Работа с двоично-десятичной арифметикой

Форматы представления десятичных чисел

В процессорах Intel существует семейство форматов представления двоично-десятичных чисел:

- числа в формате ASCII;
- неупакованные двоично-десятичные (BCD) числа;
- упакованные двоично-десятичные числа.

Представление чисел в формате ASCII удобно в тех случаях, когда необходимо выполнять ввод чисел с консоли или вывод на какое-либо устройство, например дисплей или принтер. В ASCII-числах старший (левый) полубайт каждого байта содержит значение 3h, а младший (правый) полубайт– значение десятичного разряда. Например, число 6591 в формате ASCII представлено как 36353931h, при этом самый старший байт содержит значение 36h, а самый младший – 31h.

Упакованные BCD-числа хранятся по две цифры в байте в виде четырехбитовых групп, называемых тетрадами, причем каждая тетрада представляет собой двоичную комбинацию, соответствующую одной десятичной цифре, т.е. двоичное число в диапазоне 0000b– 1001b.

Неупакованное BCD-число содержит одну десятичную цифру в младшей тетраде байта, старшая тетрада должна быть нулевой, однако для команд сложения и вычитания содержимое старшей тетрады несущественно.

Арифметические операции с неупакованными числами

Сложение одноразрядных ASCII чисел выполняется в 3 этапа.

1. Сложение командой add/adc, результат должен быть в AX.
2. Коррекция регистра AX командой aaa. Результат – в AX правильное неупакованное двузначное BCD-число.
3. Установка (при необходимости) значения 3 в старшие полубайты AX.

Для реализации сложения многоразрядных ASCII-чисел нужно организовать цикл, складывающий соответствующие разряды от младших к старшим с учетом переноса.

Вычитание одноразрядных ASCII чисел выполняется в 3 этапа.

1. Вычитание командой sub/sbb, результат должен быть в AX.
2. Коррекция регистра AX командой aas. Результат – в AX правильное неупакованное двузначное BCD-число.
3. Установка (при необходимости) значения 3 в старшие полубайты AX.

Для реализации вычитания многоразрядных ASCII-чисел нужно организовать цикл, вычитающий соответствующие разряды от младших к старшим с учетом флага переноса (заема).

Умножение одноразрядных ASCII чисел выполняется в 4 этапа.

1. Преобразование ASCII-чисел в BCD-числа.

2. Умножение командой mul, результат должен быть в AX.
3. Коррекция регистра AX командой aam. Результат – в AX правильное неупакованное двузначное BCD-число.
4. Установка (при необходимости) значения 3 в старшие полубайты AX.

Для реализации умножения многоразрядных ASCII-чисел нужно организовать цикл умножения «в столбик» с получением промежуточных произведений и их последующим сложением.

Деление одноразрядных ASCII чисел выполняется в 4 этапа.

1. Преобразование ASCII-чисел в BCD-числа.
2. Коррекция двухбайтового делимого в регистре AX командой aad.
3. Деление командой div. Результат – в AL неупакованное двузначное BCD-число – частное, в AH неупакованное BCD-число – остаток.
4. Установка (при необходимости) значения 3 в старшие полубайты AX.

Для деления многоразрядных чисел реализуется алгоритм «в столбик».

Арифметические операции с упакованными числами

Для упакованных десятичных чисел допустимы только операции сложения и вычитания. Каждая операция выполняется в 2 этапа.

На первом выполняется операция – сложение или вычитание (add, adc, sub, sbb) двух упакованных десятичных чисел, первое из которых должно находиться в регистре AL, на втором – десятичная коррекция результата в регистре AL (daa, das).

Рассмотрим подробнее одну из команд коррекции – daa, коррекция после сложения BCD-чисел.

После первого этапа – двоичного сложения правильных BCD-чисел – возможно появление неправильного BCD-результата в двух ситуациях:

1) получена недопустимая тетрада, т.е. тетрада, двоичный эквивалент которой больше 9;
2) получена допустимая тетрада, но при сложении из нее возник двоичный перенос с весом 16, в то время как правильный вес единицы переноса должен быть равен 10.

Отметим, что перенос из младшей тетрады фиксируется флагом AF, а из старшей – CF.

Алгоритм выполнения команды daa состоит из двух шагов:

1) если AF=1 или младшая тетрада регистра AL содержит запрещенную комбинацию, к содержимому AL прибавляется 06 и флаг AF устанавливается в 1;
2) если CF=1 или старшая тетрада регистра AL содержит запрещенную комбинацию, к содержимому AL прибавляется 60h и флаг CF устанавливается в 1.

Пример. Содержимое регистров AL=65h и BL=28h, что соответствует десятичным числам 65 и 28. Выполним их сложение

```
addAL, BL  ; AL=8Dh, AF=0, CF=0, ZF=0
daa        ; AL=93h, AF=0, CF=0, ZF=0
```
и вычитание
```
subAL, BL  ; AL=3Dh, AF=1, CF=0, ZF=0
das        ; AL=37h, AF=1, CF=0, ZF=0.
```
В комментариях показаны значения регистра AL и флагов после выполнения соответствующей команды.

Пример обработки BCD-чисел

Написать программу сложения двух десятиразрядных неупакованных десятичных чисел [14].

```
model  SMALL
stack  100h
dataseg
Ask1    db 0Ah,0Dh,'Введите первое слагаемое (не более 10 цифр):$'
Ask2    db 0Ah,0Dh,'Введите второе слагаемое (не более 10 цифр):$'
Buf1    db 11
Len1    db ?
Opnd1   db 12 dup( ? )
Buf2    db 11
Len2    db ?
Opnd2   db 12 dup( ? )
ResTdb 0Ah,0Dh,'Сумма'
Res     db 12 dup(' '),'$'
AskContdb 0Ah,0Dh
Db 'Завершить работу – Esc, продолжить – Любая клавиша'
db '$'
codeseg
startupcode
push    DS
pop     ES       ; ES <- DS
BEGIN:
;Ввод первого слагаемого
B1:     leaDX, Ask1
mov     AH, 09h
int     21h
lea     DX, Buf1
mov     AH, 0Ah
int     21h
cmp     Len1, 0
je      B1
;проверка 0–9 и очистка старшей тетрады
lea     BX, Opnd1
xor     CX, CX
```

```
mov     CL, Len1
xor     SI, SI
T1:     mov     AL, [BX][SI]
cmp     AL, '0'
jb      B1          ; ошибка
cmp     AL, '9'
ja      B1          ; ошибка
and     AL, 0Fh
mov     [BX][SI], AL
inc     SI
loop    T1
;прижать к правому краю
mov     CL, Len1
cmp     CL, 10
je      E1
mov     DI, 9
mov     SI, CX
dec     SI
P1:     mov     AL, [BX][SI]
mov     [BX][DI], AL
dec     DI
dec     SI
loop    P1
;обнулить лишнее
xor     DI, DI
mov     CL, 10
sub     CL, Len1
N1:     mov     byte ptr [BX][DI], 0
inc     DI
loop    N1
E1:
;Ввод второго слагаемого
B2:     leaDX, Ask2
mov     AH, 09h
int     21h
lea     DX, Buf2
mov     AH, 0Ah
int     21h
cmp     Len2, 0
je      B2
;проверка 0–9 и очистка старшей тетрады
lea     BX, Opnd2
xor     CX, CX
mov     CL, Len2
xor     SI, SI
```

```
T2:     mov     AL, [BX][SI]
cmp     AL, '0'
jbB2          ; ошибка
cmp     AL, '9'
jaB2          ; ошибка
and     AL, 0Fh
mov     [BX][SI], AL
inc     SI
loop    T2
;прижать к правому краю
mov     CL, Len2
cmp     CL, 10
je      E2
mov     DI, 9
mov     SI, CX
dec     SI
P2:     mov     AL, [BX][SI]
mov     [BX][DI], AL
dec     DI
dec     SI
loop    P2
;обнулить лишнее
xor     DI, DI
mov     CL, 10
sub     CL, Len2
N2:     mov     byte ptr [BX][DI], 0
inc     DI
loop    N2
E2:
;Сложение
mov     CX, 10
clc
lea     SI, Opnd1+9
lea     DI, Opnd2+9
lea     BX, Res+10
A1:     mov     AL, [SI]
adc     AL, [DI]
aaa
mov     [BX], AL
dec     SI
dec     DI
dec     BX
loop    A1
mov     AL, 0
adc     AL, 0
```

```
mov     [BX], AL
;Преобразование результата в ASCII
mov CX, 11
A2:     or      byte ptr [BX], 30h
inc     BX
loop    A2
;Вывод результата
lea     DX, ResT
mov     AH, 09h
int     21h
;Запрос на продолжение работы
lea     DX, AskCont
mov     AH, 09h
int     21h
mov     AH, 08h
int     21h
cmp     AL, 27  ;ESC
je      QUIT
jmp     BEGIN
;Конец работы
QUIT:   exitcode  0
end
```

Задания для самостоятельного выполнения

Имеются две группы заданий стандартной (варианты 1–7) и повышенной сложности (варианты 6–11), выберите самостоятельно любой вариант из какой-либо группы.

1. Введите два десятичных числа разрядностью не более 10 цифр, выполните преобразование в упакованный BCD-формат, сложите их и выведите результат.
2. Введите два десятичных числа разрядностью не более 10 цифр, выполните преобразование в упакованный BCD-формат, вычтите второе из первого и выведите результат.
3. Введите два десятичных числа разрядностью не более 10 цифр, выполните преобразование в упакованный BCD-формат, перемножьте их и выведите результат.
4. Введите два десятичных числа, первое разрядностью не более 10 цифр, второе – из одной цифры выполните преобразование в упакованный BCD-формат, поделите первое на второе и выведите результат.
5. Введите два десятичных числа, первое разрядностью не более 10 цифр, второе – из одной цифры выполните преобразование в неупакованный BCD-формат, перемножьте их и выведите результат.
6. Введите два десятичных числа разрядностью не более 10 цифр,

выполните преобразование в неупакованный BCD-формат, вычтете второе из первого и выведите результат.

7. Введите два десятичных числа, первое разрядностью не более 10 цифр, второе – из одной цифры выполните преобразование в неупакованный BCD-формат, поделите первое на второе и выведите результат.
8. Напишите программу-калькулятор, выполняющую действия + –, внутреннее представление чисел – упакованный BCD-формат.
9. Напишите программу-калькулятор, выполняющую действия * /, внутреннее представление чисел – упакованный BCD-формат.
10. Напишите программу-калькулятор, выполняющую действия + – *, внутреннее представление чисел – неупакованный BCD-формат.
11. Напишите программу-калькулятор, выполняющую действия + – /, внутреннее представление чисел – неупакованный BCD-формат.

Контрольные вопросы

1. Какие форматы двоично-десятичных чисел используются в процессорах Intel?
2. Чем отличаются упакованный и неупакованный BCD-форматы представления десятичных чисел?
3. Что такое десятичная коррекция результата арифметической операции?
4. Почему используются различные команды десятичной коррекции для различных арифметических операций?
5. Как организовать выполнение операций сложения и вычитания над многоразрядными операндами?
6. Зачем нужны команды десятичной арифметики?
7. Почему коррекция для деления выполняется перед операцией, а для остальных операций – после?
8. Как выполняется сложение одноразрядных ASCII-чисел?
9. Как выполняется вычитание одноразрядных ASCII-чисел?
10. Как выполняется умножение одноразрядных ASCII-чисел?
11. Как выполняется деление одноразрядных ASCII-чисел?
12. Как выполняется сложение многоразрядных ASCII-чисел?
13. Как выполняется вычитание многоразрядных ASCII-чисел?
14. Как выполняется умножение многоразрядных ASCII-чисел?
15. Как выполняется деление многоразрядных ASCII-чисел?
16. Как выполняется сложение упакованных BCD-чисел?
17. Как выполняется вычитание упакованных BCD-чисел?
18. Как выполняется сложение и вычитание многоразрядных BCD-чисел?

6. Подпрограммы

Описание и вызов подпрограмм

Описание подпрограммы в языке ассемблера имеет следующую структуру [14]:

```
имя   proc тип
      . . .
      операторы тела подпрограммы
      . . .
      ret
имя   endp
```

Здесь «тип» – одно из слов NEAR (ближняя) или FAR (дальняя). Если тип не задан, по умолчанию принимается NEAR.

Процедура NEAR должна вызываться из того же сегмента кода, в котором она описана. Процедура FAR может вызываться из других сегментов кода (с другим значением регистра CS). Такие процедуры обычно используются как отдельные объектные модули или в составе библиотек.

Команда ret выполняет возврат из процедуры в вызывающую программу. Она не обязана быть последней по тексту процедуры, но является последней по порядку выполнения. Команда ret также имеет ближний и дальний варианты в зависимости от типа подпрограммы, внутри описания которой встретилась команда.

Допускается вложение описания подпрограммы внутрь описания другой подпрограммы.

В заголовке подпрограммы рекомендуется комментировать ее характеристики. Как правило, следует отразить следующие моменты: действие, выполняемое подпрограммой; входные и выходные параметры; ограничения и особенности подпрограммы.

Вызов подпрограммы выполняется командой call. Вызов также бывает ближний или дальний. При ближнем вызове в стеке запоминается текущее значение регистра IP, используемое затем командой ret (ближней) для возврата в точку вызова. При дальнем вызове в стек заносится также значение сегментного регистра CS, что позволяет команде ret (дальней) выполнить возврат в другой сегмент.

Тип вызова определяется типом операнда команды. Если в качестве операнда указано имя подпрограммы, то тип FAR или NEAR выбирается в зависимости от описания подпрограммы. Если в качестве операнда используется слово или двойное слово памяти, то выполняется косвенный, соответственно ближний или дальний вызов подпрограммы по адресу, хранящемуся в памяти. При этом в двойном слове младшее слово содержит смещение, старшее слово – сегмент из адреса подпрограммы.

Пример. Пусть в сегменте данных описаны переменные:

```
FADDR dd  ?
NADDR dw ?
```
а в сегменте кода описаны подпрограммы:
```
FPROC proc FAR
. . .
FPROC endp
NPROC proc
. . .
NPROC endp
```
Возможны следующие варианты команд вызова:
```
call FPROC              ;Дальний прямой вызов п/п FPROC
call NPROC              ;Ближний прямой вызов п/п NPROC
call FADDR              ;Дальний прямой вызов п/п, чей адрес – в FADDR
call NADDR              ;Ближний прямой вызов п/п, чей адрес – в NADDR
call BX                 ;Ближний косвенный вызов п/п, чей адрес –в BX
call word ptr [BX]      ;Ближний косвенный вызов п/п, чей адрес –
                        ; в слове, адрес которого  – в BX
calld dword ptr [BX]    ;Дальний косвенный вызов п/п, чей адрес –
                        ; в двойном слове, адрес которого – в BX
```

Передача параметров в подпрограмму

Программист имеет полную свободу в выборе способа передачи входных параметров в подпрограмму и выходных – из подпрограммы, важно лишь, чтобы обработка параметров в подпрограмме была согласована с заданием параметров в вызывающей программе [14].

Чаще всего применяется передача параметров через регистры или через стек.

При передаче через регистры программа перед вызовом заносит входные параметры в некоторые регистры процессора, а после возврата выбирает из регистров значения результатов. При передаче через стек, программа перед вызовом заносит параметры в стек с помощью команды push. Обычно при этом считается, что подпрограмма имеет только входные параметры (как функция в языке Си). Чтобы подпрограмма могла изменять значения параметров, следует передавать ей не сами значения, а адреса параметров.

Для доступа к параметрам, переданным в стеке, в начале подпрограммы обычно выполняются команды:
```
push  BP
mov  BP, SP
```
После этого можно адресовать величины в стеке, указывая их смещения относительно вершины стека, адрес которой находится в регистре BP. При вычислении смещения нужно учитывать, что команда call, как отмечалось

выше, помещает в стек адрес возврата (одно или два слова). Удобно для адресации параметров описать соответствующую структуру данных.

Можно применять смешанные способы передачи параметров. В частности, для подпрограмм-функций удобно возвращать результат в регистре, даже если входные параметры получены в стеке.

Пример. Пусть подпрограмма типа near имеет два параметра длиной в слово передаваемых через стек. В этом случае после вызова подпрограммы, сохранения и загрузки регистра BP, стек будет выглядеть, как показано на рис. 6.1.

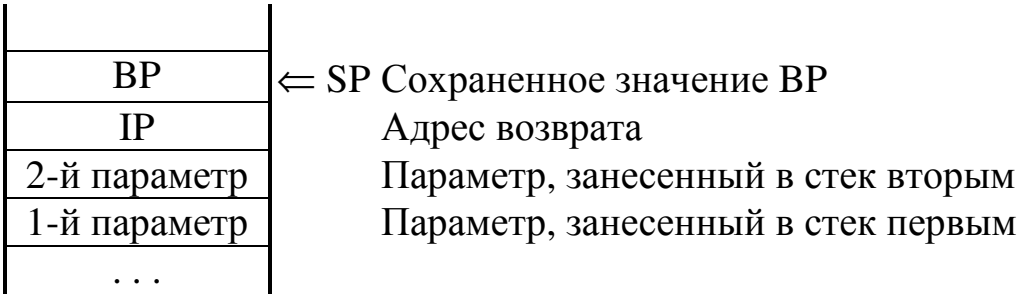

Рис. 6.1. Состояние стека после вызова подпрограммы

Если описать следующую структуру:

```
__arg  struc
    __saveBP   dw ?
    __retAddr  dw ?
    __Param2   dw ?
    __Param1   dw ?
__arg  ends,
```

то доступ к параметрам можно осуществить с помощью команд:

```
movAX, __Param1[BP]; загрузить в AX значение первого параметра
movBX, __Param2[BP]; загрузить в BX значение второго параметра.
```

Для облегчения очистки стека от переданных параметров используется разновидность команды ret с операндом – числом байтов, которые нужно убрать из стека сразу после возврата. Это позволяет вызывающей программе не заботиться об удалении параметров из стека. Для нашего примера команда возврата из подпрограммы может выглядеть следующим образом:

```
ret 4 .
```

Каждая подпрограмма должна либо сохранять значения всех регистров процессора (кроме тех, которые используются для возврата результатов), либо, в крайнем случае, в описании подпрограммы должно быть четко указано, какие регистры она изменяет. Для сохранения регистров используется стек. Команды push служат для помещения регистров в стек, а pop – для их восстановления перед возвратом из подпрограммы. Сохранение регистров должно выполняться после загрузки указателя стека в регистр BP.

Локальные переменные подпрограммы

Переменные, размещенные в сегменте данных, являются статическими [14]. Конечно, их можно рассматривать как локальные переменные подпрограмм, обеспечив локализацию области действия с помощью директивы locals (см. ниже). Однако такое статическое распределение памяти под локальные переменные не соответствует понятию локальных переменных в блочных языках типа Pascal или C, поскольку время существования таких переменных совпадает с временем существования программы. Для того чтобы решить данную проблему, т.е. обеспечить динамическое распределение памяти под локальные переменные, следует выделять для них память в стеке (как это делается в языках Pascal или C).

Предположим, что в подпрограмме должно быть две локальные переменные длиной в слово. Чтобы обеспечить выделение памяти, для них перед командами сохранения регистров следует добавить команду
sub SP, 4 ,
которая резервирует в стеке два слова.

После выполнения этой команды стек будет выглядеть в соответствии с изображением на рис. 6.2.

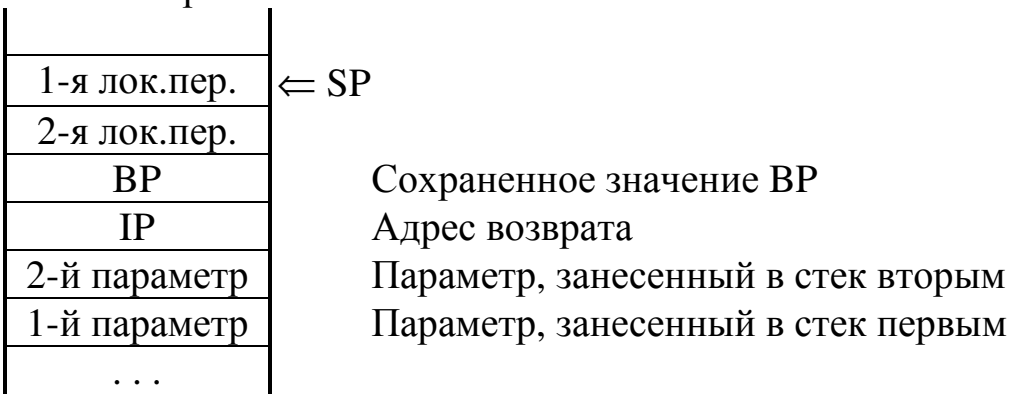

Рис. 6.2. Локальные переменные в стеке

И если определить структуру:

```
__locvars  struc
    __var1   dw ?
    __var2   dw ?
__locvars  ends,
```

то доступ к локальным переменным можно осуществить с помощью команд:

```
movAX, __var1[BP-4] ;загрузить в AX значение 1-й локальной переменной
movBX, __var2[BP-4] ;загрузить в BX значение 2-й локальной переменной.
```

Очистка стека от локальных переменных должна выполняться после восстановления сохраненных регистров, это можно сделать с помощью команд:

```
add  SP, 4
```

или

mov SP, BP .

Директивы описания сегментов и модели памяти

Директива model позволяет задать для программы несколько стандартных моделей сегментации. Вы можете также использовать ее для задания языка процедур программы. Она имеет следующий синтаксис [14]:
model [модификатор_модели] модель_памяти [имя_сегмента_кода]
[,[модификатор_языка] язык] [, модификатор_модели]
модель_памяти и модификатор_модели определяют модель сегментации памяти, используемую в программе.

В применяемых в ассемблере стандартных моделях можно использовать специальные сегменты: для кода, инициализированных данных, неинициализированных данных, инициализированных данных дальнего типа, неинициализированных данных дальнего типа, констант, стека.

Сегмент кода обычно содержит код модуля (но при необходимости может также содержать данные). В целях совместимости с языками высокого уровня инициализированные данные и константы интерпретируются поразному. Они содержат такие данные, как сообщения, когда важно начальное значение. Неинициализированные данные и стек содержат данные, начальные значения которых не существенны. Инициализированные данные дальнего типа (far) – это неинициализированные данные, которые не являются частью стандартного сегмента данных и которые доступны только при изменении значения сегментного регистра. Это же относится к неинициализированным данным дальнего типа.

Единственным обязательным параметром директивы model является модель памяти. Стандартные модели памяти описаны в табл. 6.1.

Таблица 6.1

Модель	Код	Данные	Описание
TINY	near	near	Весь код и все данные комбинируются в одну группу с именем DGROUP. Эта модель используется для программ, ассемблируемых в формат .COM. Некоторые языки эту модель не поддерживают
SMALL	near	near	Код представляет собой один сегмент. Все данные комбинируются в группу с именем DGROUP. Это наиболее общая модель, использующаяся для автономных программ на ассемблере

Окончание табл. 6.1

Модель	Код	Данные	Описание
MEDIUM	far	near	Для кода используется несколько сегментов, по одному на модуль. Данные находятся в группе с именем DGROUP
COMPACT	near	far	Код находится в одном сегменте. Все ближние данные находятся в группе с именем DGROUP. Для ссылки на данные используются дальние указатели
LARGE	far	far	Для кода используется несколько сегментов, по одному на модуль. Все ближние данные находятся в группе с именем DGROUP. Для ссылки на данные используются дальние указатели
HUGE	far	far	То же, что и модель LARGE (что касается Турбо ассемблера)
FLAT	near	near	То же, что и модель SMALL, но использует модель плоской памяти защищенного режима и предназначена для использования в Win32

Поле модификатор_модели позволяет изменить отдельные аспекты модели. При необходимости можно задавать несколько модификаторов модели. Доступные модификаторы модели приведены в табл. 6.2.

Таблица 6.2

Модификатор модели	Функция
NEARSTACK	Указывает, что сегмент стека должен включаться в DGROUP (если группа DGROUP присутствует), а SS должен указывать на DGROUP
FARSTACK	Указывает, что сегмент стека не должен включаться в DGROUP, а SS должен указывать на nothing (не определен)
USE16	Задает, что все сегменты в выбранной модели должны быть 16-разрядными (при выборе процессора 80386 или старше)
USE32	Задает, что все сегменты в выбранной модели должны быть 32-разрядными (при выборе процессора 80386 или старше)
DOS, OS_DOS	Задает, что прикладная программа ориентируется на DOS
OS2, OS_OS2	Задает, что прикладная программа ориентируется на OS2, Win32

Язык и модификатор_языка вместе определяют соглашения, используемые при вызове процедуры, а также используемый по умолчанию характер начала и завершения кода каждой процедуры. Они определяют так же, как будут объявляться общедоступные идентификаторы (которые использует компоновщик).

После выбора модели памяти вы можете использовать упрощенные сегментные директивы для того, чтобы начинать отдельные сегменты. Некоторые из них приведены в табл. 6.3.

Таблица 6.3

Директива	Описание
codeseg [имя]	Начинает или продолжает сегмент кода модуля. Для моделей с дальним типом кода вы можете задать имя, являющееся фактическим именем сегмента. Заметим, что таким образом вы можете генерировать для модуля более одного сегмента кода
dataseg	Начинает или продолжает ближний или инициализируемый по умолчанию сегмент данных
Const	Начинает или продолжает ближний сегмент констант модуля. Постоянные данные всегда являются ближними (NEAR) и эквивалентны инициализированным данным
stack [размер]	Начинает или продолжает сегмент стека модуля. Необязательный параметр «размер» определяет объем резервируемой для стека памяти (в словах). Если вы не задаете размер, Турбоассемблер резервирует по умолчанию 1 килобайт. Директивы стека обычно требуется использовать, если вы пишете на языке ассемблера автономную программу. Большинство языков высокого уровня сами создают для вас стек

Макрокоманда startupcode обеспечивает код инициализации, соответствующий текущей модели и операционной системе. Она отмечает также начало программы. Эта макрокоманда имеет следующий синтаксис:
startupcode .

Макрокоманда инициализирует регистры DS, SS, SP и помещает идентификатор @Startup в начало кода инициализации, который генерируется по директиве startupcode. Он представляет собой ближнюю метку, отмечающую начало программы.

Макрокоманда exitcode используется для генерации кода завершения, соответствующего текущей операционной системе. Ее можно использовать в модуле несколько раз (для каждой точки выхода). Эта макрокоманда имеет следующий синтаксис:
exitcode [возвращаемое_значение] .

Необязательное «возвращаемое_значение» задает число, которое должно возвращаться в операционную систему – «код возврата». Если вы не задаете возвращаемое значение, ассемблер предполагает, что это значение содержится в регистре AL.

Аргументы, локальные переменные и область видимости имен

Возможен следующий синтаксис описания процедуры [14]:

```
ИМЯ   proc
    [arg СПИСОК_АРГУМЕНТОВ]
    [local СПИСОК_АРГУМЕНТОВ]
...
[ИМЯ] endp
```

Приведем синтаксис определения передаваемых процедуре аргументов:

```
Arg АРГУМЕНТ [, АРГУМЕНТ ] ... [ =ИМЯ ]
```

При определении локальных переменных процедуры используется следующий синтаксис:

```
Local АРГУМЕНТ [, АРГУМЕНТ ] ... [ =ИМЯ ]
```

Отдельные аргументы имеют следующий синтаксис:

```
ИМЯ_АРГУМЕНТА [[ СЧЕТЧИК_1 ]] [: ТИП [: СЧЕТЧИК_2 ]]
```

Здесь ТИП – это тип данных аргумента – byte, word, dwordи т.п.

СЧЕТЧИК_2 задает, сколько элементов данного типа определяет аргумент.

Например, в определении аргумента

```
arg    TMP: dword: 4
```

определяется аргумент с именем TMP, состоящий из 4 двойных слов. По умолчанию СЧЕТЧИК_2 имеет значение 1 (кроме аргументов типа byte). Так как вы не можете занести в стек байтовое значение, для аргументов типа byte значение счетчика по умолчанию равно 2, что обеспечивает для них в стеке размер в слово.

Например:

```
arg    REALBYTE: byte: 1
```

СЧЕТЧИК_1 представляет собой число элементов массива. Если поле СЧЕТЧИК_1 не задано, то по умолчанию оно равно 1.

Если список аргументов завершается символом равенства (=) и идентификатором, то ассемблер будет приравнивать этот идентификатор к общему размеру блока аргументов (в байтах). Если вы не используете автоматическое использование соглашений языков высокого уровня в ассемблере, то можете использовать данное значение в конце процедуры в качестве аргумента инструкции ret.

Аргументы и переменные определяются в процедуре как операнды в памяти относительно BP. Передаваемые аргументы, определенные с помощью директивы arg, имеют положительное смещение относительно BP. Локальные

переменные, определенные с помощью директивы local, имеют отрицательное смещение от BP.

Например:

```
FUNC1  proc  NEAR
      arg   A:word, B:word:4, C:byte =ArgSize
      local X:dword, Y:word:2 =LocSize
...
```

Здесь A определяется, как [BP+4], B – [BP+6], C – [BP+14], ArgSize – 20. X – [BP-2], Y – [BP-6], a LocSize – 8.

Если для аргументов и локальных переменных, а также меток операторов в подпрограмме, не заданы имена с предшествующим префиксом локального идентификатора, все аргументы, заданные в заголовке процедуры, определены ли они с помощью директивы arg или local и метки имеют глобальную область действия.

Идентификаторы с локальной областью действия разрешает директива locals (не путать с директивой local).

Например:

```
locals __
TST1  proc
     arg    __A: word, __B: word, __C: byte
     local  __X: word, __Y: dword
...
```

В этом примере директива locals __ определяет двойной символ подчеркивания как префикс локальных имен. Это означает, что все имена, начинающиеся с данной пары символов, будут считаться локальными в пределах подпрограммы.

Шаблон подпрограммы

Соединив все, что было сказано выше, можно предложить следующий шаблон подпрограммы [14]:

```
locals __
...
Func    proc  near
       arg    __p1: word, __p2: word, ... = __ArgSize
       local  __v1: byte, __v2: word, ... = __LocSize
;Действие: ...
;Входные параметры: ...
;Выходные параметры: ...
;Возвращает: ...
;Обращение: ...
;Замечания:
```

```
;  регистры не модифицирует
;  стек чистит от параметров

push  BP                ; BP – указатель стека
mov   BP, SP            ;
sub   SP, __LocSize     ;выделение памяти для локальных переменных
push  SI, DI            ;сохранение регистров
      ...
mov   SI, __p1          ;доступ к параметру
      ...
mov    __v2, SI         ;доступ к локальной переменной
      ...
pop    DI, SI           ;восстановление регистров
mov   SP, BP            ;чистка стека от лок. переменных
pop    BP               ;восстановление BP
ret   __ArgSize         ;возврат и чистка стека от параметров
Func endp
```

Пример использования подпрограммы

Разработать подпрограмму, которая удаляет, начиная с заданной позиции строки, указанное число символов. Разработать программу, которая вводит с клавиатуры строку, вводит позицию и длину удаляемой части строки и удаляет эту часть [14].

```
      locals __
      model small
      stack 100h
      dataseg
MESS1     db 0dh,0ah,0dh,0ah,"Введите строку:",0dh,0ah,"$"
MESS2     db 0dh,0ah,"Введите позицию: $"
MESS3     db 0dh,0ah,"Введите число удаляемых символов: $"
MESS4     db 0dh,0ah,0dh,0ah,"Строка после удаления:",0dh,0ah,"$"
S_BUFLEN db 80          ; Макс. длина строки
S_FACTLEN db?           ; Длина фактически введенной строки
S_INPBUF db 80 dup(?)   ; Введенная строка
N_BUFLEN db 3           ; Макс. длина числа при вводе
N_FACTLEN db ?          ; Фактическая длина
N_INPBUFdb 3 dup(?)     ; Строка представления числа
POSDEL dw ?           ; Позиция, начиная с которой удаляем
LENDEL dw ?           ; Сколько символов удалить
codeseg
startupcode
; Ввод строки
MLOOP:  lea     DX, MESS1
```

```
        mov AH, 09h
int     21h                     ;Приглашение к вводу строки
lea     DX, S_BUFLEN
        mov     AH, 0Ah
        int     21h             ;Ввод строки
        mov     BL, S_FACTLEN
        cmp BL, 0               ;Строка пустая?
          jne LLL0              ;Нет – продолжать
          jmp QUIT              ;Закончить работу
LLL0:   movBH, 0
;Дополнить длину до слова
          add BX, 2             ; и получить адрес позиции
          add BX, DX; сразу после конца строки
          mov byte ptr [BX],0       ;Записать признак конца строки
; Ввод позиции удаления
LLL1:   lea DX, MESS2     ;Приглашение
          mov AH, 09h; к вводу
int     21h; позиции удаления
lea     DX, N_BUFLEN
        mov     AH, 0Ah
        int     21h                     ;Ввод строки числа
          lea BX, N_INPBUF      ;Адрес строки представления числа
          mov CL, N_FACTLEN;Длина этой строки
          call VAL              ;Перевод в целое число
          jc LLL1               ;Ошибка? – повторить ввод
          cmp AL, 0             ;Ноль?
          je LLL1               ;Повторить ввод
          cmp AL, S_FACTLEN;Превышает длину строки?
          jg LLL1               ;Повторить ввод
          mov POSDEL, AX        ;Запомнить позицию удаления
; Ввод длины удаляемой части
LLL2:   lea DX, MESS3           ;Приглашение
          mov AH, 09h           ; к вводу
          int     21h           ; числа удаляемых
          lea     DX, N_BUFLEN
        mov     AH, 0Ah
        int     21h             ;Ввод строки числа удаляемых
          lea BX, N_INPBUF      ;Адрес строки представления числа
          mov CL, N_FACTLEN;Длина этой строки
          call  VAL             ;Перевод в целое число
          jc LLL2               ;Ошибка? – повторить ввод
          mov LENDEL, AX        ;Запомнить число удаляемых
          add AX, POSDEL ;Подсчитать, не выходит ли
          dec AX                ; удаляемая часть
          cmp AL, S_FACTLEN     ; за конец строки?
```

```
        jg LLL2                 ;Если да – повторить ввод
; Занесение параметров в стек и вызов подпрограммы удаления
        lea AX, S_INPBUF
        push AX                 ;1-й параметр – адрес строки
        push POSDEL             ;2-й параметр – позиция удаления
        push LENDEL             ;3-й параметр – число удаляемых
        call DELSUBS            ;Вызов подпрограммы
; Вывод результата
        lea     DX, MESS4
       mov     AH, 09h
       int     21h;Заголовок вывода
        lea BX, S_INPBUF
        mov CX, 80
LLL3:   cmp byte ptr [BX],0     ;Цикл поиска конца строки и выход
        je LLL4                 ; если найден конец строки
        inc BX                  ;Сдвиг по строке
        loop LLL3
LLL4:   mov byte ptr [BX],'$';Заменить признак конца строки
        lea     DX, S_INPBUF
       mov     AH, 09h
       int     21h              ; Вывод результата
        jmp MLOOP               ; На повторение работы
QUIT:   exitcode 0
;Действие:
;  функция вычисляет целое число по его строковому представлению.
;  Результат не может быть больше 255.
;  Для неверно введенных чисел устанавливает флаг переноса
; Параметры:
;  BX – адрес строки представления числа
;  CX – длина этой строки
; Возвращает:
;  CF – установлен, если в строке не цифры, AX – не определен
;       сброшен, строка нормальная, AX – число
;  AX – преобразованное число, если сброшен
VAL proc near
        push DX             ;Сохранить все изменяемые регистры,
                        ; кроме AX, в котором результат
        mov CH, 0           ;Расширяем длину до слова
        mov AX, 0           ;Начальное значение результата
        mov DL, 10          ;Основание системы счисления
__1:    imul DL             ;Умножить на основание
        jc      __2         ;Переполнение байта?
        mov DH, [BX]    ;Очередная цифра
        sub DH, '0'         ;Получить значение цифры
        jl      __2         ;Это была не цифра!
```

```
        cmp DH, 9
        jg      __2             ;Это опять же была не цифра!
        add AL, DH;+ значение цифры к результату
        jc      __2             ;Переполнение байта?
        inc BX                  ;Сдвиг по строке
        loop    __1         ;Цикл по строке
        jmp     __3         ;Нормальное число
__2:    stc                     ;Было переполнение – устанавливаем CF
__3:    pop DX                  ;Восстановить все, что сохраняли
        ret
VAL endp
; Подпрограмма удаления подстроки
DELSUBS proc    near
       arg  __Ldel: word, __Pdel: word, __StrAdr: word = __ArgSize
;Paramsstruc;Структура стека после сохранения BP
;  SaveBP dw ?        ;  Сохраненное значение BP
;  SaveIP dw ?        ;  Адрес возврата
;  LDel dw ?          ;  3-й параметр – число удаляемых
;  PDel dw?           ;  2-й параметр – позиция удаления
;  StrAdr dw ?        ;  1-й параметр – адрес строки
;Params  ends
       push    BP      ;Сохранить BP
       mov BP, SP      ;Теперь BP адресует стек ПОСЛЕ сохранения BP,
                       ; но ДО сохранения остальных регистров
         push ESAXSIDICX        ;Сохранить все изменяемые регистры
         mov AX,DS          ; ES будет указывать на
         mov ES,AX          ;  сегмент данных
         mov DI,__StrAdr  ; Установить в DI адрес,
         add DI,__PDel      ; куда надо
         dec DI             ;  пересылать символы
         movSI,DI           ;A в SI – адрес,
         add SI,__LDel      ; откуда их пересылать
         cld                ;Продвигаться от начала строки к концу
__REPEAT:
       movsb
       cmp     byte ptr [SI-1], 0
       jne     __REPEAT
         pop CXDISIAXES         ;Восстановить все, что сохраняли
         pop BP
         ret     __ArgSize          ;Убрать из стека 3 параметра-слова
DELSUBS endp
end
```

Задания для самостоятельного выполнения

В приведенных ниже вариантах заданий используется стандартное представление строк ASCII с кодом 0 в качестве ограничителя конца строки. Способ передачи параметров выбирается программистом произвольно. Рекомендуется зациклить программу по вводу, а признаком окончания работы считать ввод пустой строки.

1. Разработать подпрограмму, которая определяет, содержится ли одна заданная строка в другой заданной строке, и если да, то, начиная с какой позиции. Разработать программу, которая вводит с клавиатуры две строки и сообщает, содержится ли одна из них в другой и сколько раз.
2. Разработать подпрограмму, которая подсчитывает, сколько раз заданный символ встречается в строке. Разработать программу, которая вводит с клавиатуры строку, вводит число N и выдает список символов, которые встречаются в строке не менее чем N раз.
3. Разработать две подпрограммы, одна из которых соединяет две строки в одну, а другая обрезает строку до заданной длины (или дополняет пробелами, если длина строки меньше заданной). Разработать программу, которая вводит с клавиатуры число N, затем вводит несколько строк (конец ввода – пустая строка) и формирует новую строку, состоящую из первых N символов каждой введенной строки.
4. Разработать две подпрограммы, одна из которых сравнивает две строки по лексикографическому порядку, а другая обменивает значения двух строк. Разработать программу, которая вводит с клавиатуры несколько строк (конец ввода – пустая строка) и сортирует их в лексикографическом порядке.
5. Разработать подпрограмму, которая разбивает заданную строку на две части: первое слово строки (до первого пробела) и остаток строки (пробелы после первого слова отбрасываются). Разработать программу, которая вводит с клавиатуры строку и выводит каждое слово с новой строки.
6. Разработать подпрограмму, которая переставляет символы заданной строки в обратном порядке. Разработать программу, которая вводит с клавиатуры строку и переставляет в обратном порядке символы в каждом слове (слова разделяются пробелами).
7. Разработать подпрограмму, которая вставляет подстроку в строку, начиная с заданной позиции. Разработать программу, которая вводит с клавиатуры исходную строку, вводит подстроку и позицию вставки, вставляет подстроку в строку.
8. Разработать две подпрограммы, одна из которых преобразует любую заданную букву в заглавную (в том числе для русских букв), а другая -

преобразует букву в строчную. Разработать программу, которая вводит с клавиатуры строку и заменяет первые буквы всех слов на заглавные, а остальные буквы – на строчные.

9. Разработать две подпрограммы. Первая разбивает заданную строку на две части: первое слово строки (до первого пробела) и остаток строки (пробелы после первого слова отбрасываются). Вторая преобразует строку в целое число или 0, если в строке записано нечисловое значение. Разработать подпрограмму, которая вводит строку, состоящую из целых чисел (положительных или отрицательных) и вычисляет сумму этих чисел.
10. Разработать две подпрограммы. Первая ищет слово в заданной строке, начиная с заданной позиции, и возвращает номер позиции начала найденного слова и его длину (если слова в строке нет – возвращает нули). Вторая преобразует символы заданной строки, начиная с заданной позиции и заданной длины, в целое число или 0, если в строке записано нечисловое значение. Разработать подпрограмму, которая вводит строку, состоящую из целых чисел (положительных или отрицательных) и вычисляет максимальное из этих чисел.
11. Разработать подпрограмму, которая преобразовывает заданное целое двоичное число в символьную строку. Написать программу, которая сортирует массив слов по возрастанию, формирует и выводит на экран строку с числовыми значениями элементов отсортированного массива.
12. Разработать подпрограмму, которая удаляет все вхождения заданного символа в заданной строке. Написать программу, которая вводит две строки и удаляет все символы первой строки из второй.

Контрольные вопросы

1. Что такое «ближние» и «дальние» подпрограммы?
2. Как определяется, «ближний» или «дальний» вариант команды call использован в программе?
3. Какие еще способы передачи параметров можно предложить, кроме двух, описанных в данной работе?
4. Может ли массив быть параметром процедуры?
5. Нельзя ли адресовать параметры в стеке через регистр SP, не используя BP?
6. Что и как нужно изменить в программе из примера, если используется версия ассемблера, не поддерживающая понятие структуры?
7. Изменить описание подпрограммы из примера с использованием упрощенных директив описания подпрограмм.
8. Что означает операнд команды ret?
9. Какой последовательностью команд можно заменить команду ret 6?

7. Работа с математическим сопроцессором

Программная модель сопроцессора

Программная модель арифметического сопроцессора состоит из 3 групп регистров.

1. Регистры R0..R7 – предназначены для хранения вещественных операндов, которые используются в вычислениях. Каждый регистр содержит 80 бит (0–63 – мантисса, 64–78 – порядок, 79 – знак числа). Эти регистры составляют стек сопроцессора и оптимизированы на реализацию вычислений с использованием обратной польской записи.
2. Три служебных регистра SWR, CWR и TWR длиной 16 бит каждый:
   - SWR – регистр состояния сопроцессора. Содержит информацию о текущем состоянии сопроцессора, указывает, какой из регистров R0 – R7 является вершиной стека сопроцессора, какие исключения возникли после выполнения последней команды и каковы особенности ее выполнения. Этот регистр является аналогом регистра флагов центрального процессора;
   - CWR – управляющий регистр сопроцессора. С помощью его полей можно регулировать точность выполнения вычислений, управлять округлением, маскировать исключения;
   - TWR – регистр слова тегов. Используется для контроля за состоянием каждого из регистров R0 – R7. Для этого каждому из регистров стека сопроцессора в регистре TWR отведено по 2 бита: 0, 1 – R0; 2, 3 – R1 и т.д.
3. Два регистра указателей DPR и IPR. Используются при обработке исключительных ситуаций:
   - DPR – регистр указателя данных. Используется для хранения адреса операнда команды, вызвавшей исключение;
   - IPR – регистр указателя команды. Используется для хранения адреса вызвавшей исключение команды.

Стек регистров сопроцессора организован по принципу кольца. Вершина стека является плавающей и перемещается после записи операнда в вершину (например, если текущей вершиной является регистр R0, то после записи значения в стек текущей вершиной станет регистр R1). Команды сопроцессора не оперируют физическими номерами регистров R0 – R7. Они используют логические номера регистров ST(0), ST(1)...ST(7) относительно текущей вершины стека.

Регистр состояния сопроцессора SWR содержит 16 бит, назначение которых описано в табл. 7.1.

Таблица 7.1

Бит	Обозначение	Назначение
0	IE	Недействительная операция
1	DE	Денормализованный операнд
2	ZE	Ошибка деления на нуль
3	OE	Ошибка переполнения – выход порядка за максимально допустимый диапазон значений
4	UE	Ошибка антипереполнения (результат слишком маленький)
5	PE	Ошибка точности – округление числа при выходе за пределы разрядной сетки
6	SF	Ошибка работы стека сопроцессора. 1 – возникла одна из исключительных ситуаций PE, UE или IE, выполнена попытка записи в заполненный стек или чтения из пустого стека. После анализа этого бита его нужно обнулить вместе с битами PE, UE, IE
7	ES	Суммарная ошибка работы сопроцессора. 1 – возникла любая из шести исключительных ситуаций (биты 0 – 5)
8	C0	Код условия
9	C1	Код условия
10	C2	Код условия
11-13	TOP	Номер физического регистра R0..R7, который является текущей вершиной стека
14	C3	Код условия
15	B	Бит занятости. 1 – сопроцессор выполняет команду или происходит прерывание от основного процессора. 0 – сопроцессор свободен

Биты C0 – C3 являются аналогом регистра флагов центрального процессора. В них команды сопроцессора записывают коды условий, интерпретация которых зависит от конкретной команды.

Регистр управления сопроцессором CWR содержит 16 бит, назначение которых описано в табл. 7.2.

Регистр тегов TWR представляет совокупность двухразрядных полей, соответствующих определенному физическому регистру стека сопроцессора. Значение поля характеризует состояние соответствующего физического регистра:

- 00 – регистр занят допустимым ненулевым значением;
- 01 – регистр содержит ноль;
- 10 – регистр содержит одно из специальных значений, кроме нуля;

- 11 – регистр пуст и в него можно записать число.

Таблица 7.2

Бит	Обозначение	Назначение
0	I	Маски исключений. Предназначены для маскирования исключительных ситуаций, возникновение которых фиксируется битами 0–5 регистра SWR. 1 – соответствующее исключение обрабатывается самим сопроцессором. 0 – при возникновении исключения возбуждается прерывание 10h, обработчик которого должен быть написан программистом
1	D	
2	Z	
3	O	
4	U	
5	P	
6	Зарезервировано	
7	IEM	Маска разрешения прерываний. 1 – даже при возникновении незамаскированного исключения (бит 0 -5 равен 0) прерывание не возбуждается
8-9	PC	Поле управления точностью: 00 –мантисса занимает 24 бита, 10 – мантисса занимает 53 бита, 11 – мантисса занимает 64 бита
10-11	RC	Поле управления округлением: 00 – округление по обычным правилам, 01 – округление в меньшую сторону, 10 – округление в большую сторону, 11 – отбрасывание дробной части результата (используется в операциях целочисленной арифметики)
13-15	Зарезервировано	

Форматы данных сопроцессора

Сопроцессор работает с числами в следующих форматах:

- двоичные целые числа длиной 16, 32 и 64 бита;
- упакованные целые десятичные числа длиной до 9 байт (могут содержать до 18 десятичных цифр);
- вещественные числа в коротком (32 бита), длинном (64 бита) и расширенном (80 бит) форматах.

Вещественные числа являются основным форматом, с которым работает сопроцессор. Они представляются в виде мантиссы и порядка, связанных формулой

$$A = (\pm M) \cdot 2^{\pm(p)}.$$

Здесь A – вещественное число, M – мантисса числа, p – порядок числа.

Мантисса должна быть представлена в нормализованном виде, т.е. удовлетворять соотношению: $1 \le M < 2$. Другими словами, в мантиссе всегда должна быть единичная целая часть. Требование нормализованности позволяет избежать случаев неоднозначного представления одного и того же числа вещественными числами с различным значением порядка.

В сопроцессоре на аппаратном уровне принято соглашение, что порядок p всегда задается неотрицательным значением q, называемым характеристикой и связанным с p формулой

q = p + фиксированное смещение.

Форматы вещественных чисел описаны в табл. 7.3.

Таблица 7.3

Формат числа	Короткий	Длинный	Расширенный
Длина (бит)	32	64	80
Диапазон значений	$10^{-38}...10^{38}$	$10^{-308}...10^{308}$	$10^{-4392}…10^{4392}$
Мантисса M	23 бита 0–22 биты	53 бита 0–51 биты	64 бита 0–62 биты
Характеристика q	8 бит 23–30 биты	11 бит 52–61 биты	15 бит 63–78 биты
Диапазон характеристик	0…255	0…2047	0…32767
Значение фиксированного смещения	+127	+1023	+16383
Диапазон порядков	–126.... +127	-1022… +1023	-16382… +16383
Бит знака числа	31	62	79
Директива описания данных в программе	DD	DQ	DT

Основным форматом для сопроцессора является расширенный формат вещественных чисел. Если используются другие вещественные или целочисленные форматы, сопроцессор выполняет их преобразование в расширенный формат.

Примеры описания вещественных чисел в программе на языке ассемблера:

```
; описание числа 51.25 в коротком формате
DD   51.25
DD   51.25E0
DD   0.5125E2
; описание числа 3 в длинном формате
DQ   3.0
```

```
DQ  0.3E1
; описание числа 0.005 в расширенном формате
DT  0.005
DT  5.0E-3
```

При работе с дампом вещественного числа нужно иметь ввиду следующие моменты:

- в коротком (32 бита) и длинном (64 бита) представлении единичный бит целой части мантиссы не хранится, а добавляется к записи числа на аппаратном уровне. Это позволяет немного увеличить разрядность мантиссы;
- в длинном (80 бит) представлении вещественного числа единичный бит целой части мантиссы присутствует в явном виде.

Кроме обычных чисел сопроцессор умеет работать с некоторыми специальными форматами, которые могут получиться в результате выполнения некоторых математических операций. Они называются специальными численными значениями.

Существуют следующие специальные численные значения:

- денормализованные вещественные числа – числа, меньше минимального нормализованного числа. Это числа очень близкие к нулю. Биты порядка заполнены нулями;
- нуль. Возможен положительный нуль (все биты числа равны нулю) и отрицательный (все биты равны нулю, а знаковый – единице);
- бесконечность. Бывает положительная и отрицательная. В положительной нулевыми являются биты знака и мантиссы, а порядок заполнен единицами. В отрицательной бит знака равен единице;
- не-числа. Существуют сигнальные и тихие не-числа. В сигнальных первый бит мантиссы равен 0 (или 10 для расширенного формата), все биты порядка установлены в 1. Сопроцессор не может формировать сигнальное не-число, а программист может загрузить его в стек сопроцессора преднамеренно для вызова исключительной ситуации. В тихом не-числе первый бит мантиссы равен 1 (11 для расширенного формата), все биты порядка установлены в 1. Сопроцессор может генерировать тихие не-числа в операциях, если один из операндов является тихим не-числом;
- неопределенность. Знаковый бит равен 1, первый бит мантиссы – 1 (11 для расширенного формата), остальные биты мантиссы сброшены в 0, порядок заполнен единицами;
- неподдерживаемое число. К таким числам относятся все остальные ситуации.

Команды сопроцессора

Для команд сопроцессора используются следующие мнемонические соглашения:

- все команды начинаются с буквы F;
- вторая буква определяет тип операнда, с которым работает команда. I – целое двоичное число, B – целое число в упакованном формате, в остальных случаях – вещественное число;
- последняя буква P – после выполнения команды операнд выталкивается из стека сопроцессора;
- последняя или предпоследняя буква R – реверс операндов при выполнении команды.

Команды передачи данных приведены в табл. 7.4.

Таблица 7.4

Команда	Действие
Передача вещественных чисел	
FLD источник	Загрузка числа из памяти в вершину стека сопроцессора
FST приемник	Сохранение числа из вершины стека сопроцессора в память
FSTP приемник	Сохранение числа из вершины стека сопроцессора в память с последующим выталкиванием его из вершины стека
Передача целых чисел	
FILD источник	Загрузка числа из памяти в вершину стека сопроцессора
FIST приемник	Сохранение числа из вершины стека сопроцессора в память
FISTP приемник	Сохранение числа из вершины стека сопроцессора в память с последующим выталкиванием его из вершины стека
Передача упакованных чисел	
FBLD источник	Загрузка числа из памяти в вершину стека сопроцессора
FBST приемник	Сохранение числа из вершины стека сопроцессора в память
FBSTP приемник	Сохранение числа из вершины стека сопроцессора в память с последующим выталкиванием его из вершины стека
Другие способы передачи	
FXCH ST(I)	Обмен вершины стека ST(0) с другим регистром стека ST(I)

Команды загрузки констант в вершину стека сопроцессора перечислены в табл. 7.5.

Таблица 7.5

Команда	Действие
FLDZ	Загрузить 0
FLD1	Загрузить 1
FLDPI	Загрузить значение π
FLDL2T	Загрузить двоичный логарифм десяти
FLDL2E	Загрузить двоичный логарифм числа e
FLDLG2	Загрузить десятичный логарифм двойки
FLDLN2	Загрузить натуральный логарифм двойки

Группа команд сравнения выполняет сравнение содержимого вершины стека ST(0) с источником и устанавливают соответствующие флаги сопроцессора или основного процессора (начиная с PentiumPro). Источник может находиться или в регистре стека ST(i), не являющемся вершиной, или в памяти. Если источник в команде не указан, то сравнение выполняется с регистром ST(1). При сравнении целых чисел источник может находиться только в памяти.

Команды сравнения перечислены в табл. 7.6.

Таблица 7.6

Команда	Действие
FCOM источник	Сравнение вещественных чисел
FCOMP источник	Сравнение вещественных чисел и выталкивание операнда из вершины стека
FCOMPP	Сравнение ST(0) и ST(1) и выталкивание их из стека
FUCOM источник	Сравнение вещественных данных без учета порядков
FUCOMP источник	Сравнение вещественных данных без учета порядков и выталкивание операнда из вершины стека
FUCOMPP	Сравнение ST(0) и ST(1) без учета порядков и выталкивание их из стека
FICOM источник	Сравнение целочисленных данных
FICOMP источник	Сравнение целочисленных данных и выталкивание операнда из вершины стека
FCOMI ST(0), ST(i) (Pentium Pro)	Сравнение вещественных чисел с установкой флагов основного процессора

Окончание табл. 7.6

Команда	Действие
FCOMIP ST(0), ST(i) (Pentium Pro)	Сравнение вещественных чисел с установкой флагов основного процессора и выталкивание операнда из вершины стека
FUCOMIST(0), ST(i) (Pentium Pro)	Сравнение вещественных чисел без учета порядков с установкой флагов основного процессора
FUCOMIPST(0), ST(i) (Pentium Pro)	Сравнение вещественных чисел без учета порядков с установкой флагов основного процессора и выталкивание операнда из вершины стека
FTST	Сравнение содержимого ST(0) с нулем
FXAM	Анализ содержимого ST(0) и определение типа находящегося в нем числа

После выполнения команд сравнения по результатам сравнения устанавливаются флаги в регистре SWR в соответствии с табл. 7.7.

Таблица 7.7

Условие	Флаг C3	Флаг C2	Флаг C0
ST(0) > источник	0	0	0
ST(0) < источник	0	0	1
ST(0) = источник	1	0	0
Операнды несравнимы	1	1	1

Для реализации условных переходов по значениям этих флагов их нужно поместить в регистр флагов основного процессора. Это делается последовательным выполнением двух команд:

```
FSTSW     ; загрузить регистр SWR в AX.
SAHF      ; загрузить AH в младший байт регистра флагов.
```

После выполнения этих команд значение C0 находится в CF, C2 – PF, C3– ZF.

Команды сравнения для процессора PentiumPro позволяют сразу выполнить такую загрузку флагов в процессе выполнения самой команды сравнения.

Результатом действия команды FXAM является установка флагов C3, C2, C0. Возможные комбинации значений приведены в табл. 7.8.

Таблица 7.8

Тип числа в ST(0)	Флаг C3	Флаг C2	Флаг C0
Не поддерживаемое	0	0	0
Не-число	0	0	1
Нормальное конечное число	0	1	0
Бесконечность	0	1	1
Нуль	1	0	0
Регистр пуст	1	0	1
Денормализованное число	1	1	0

Группа команд арифметических операций предназначена для реализации основных арифметических действий над вещественными и целыми числами. В командах обработки вещественных чисел в качестве источника может выступать либо регистр из стека сопроцессора, либо операнд в памяти, имеющий короткий или длинный формат. В командах обработки целых чисел источник может быть только операндом в памяти.

Описание базовых арифметических команд приведено в табл. 7.9.

Таблица 7.9

Команда	Действие
FADD	Сложение ST(0) и ST(1). Результат помещается в ST(0)
FADD источник	Сложение ST(0) и источника. Результат помещается в ST(0)
FADD ST(i), ST	Сложение ST(i) и ST(0). Результат помещается в ST(i)
FADDP ST(i), ST	Сложение ST(i) и ST(0) с выталкиванием значения из ST(0). Результат помещается в ST(i – 1)
FSUB	Вычитание значения ST(1) из ST(0). Результат помещается в ST(0)
FSUB источник	Вычитание значения источника из ST(0). Результат помещается в ST(0)
FSUB ST(i), ST	Вычитание значения ST(0) из ST(i). Результат помещается в ST(i)
FSUBP ST(i), ST	Вычитание значения ST(0) из ST(i) с выталкиванием значения из ST(0). Результат помещается в ST(i – 1)
FSUBR ST(i), ST	Вычитание значения ST(i) из ST(0). Результат помещается в ST(i)
FSUBRP ST(i), ST	Вычитание значения ST(i) из ST(0) с выталкиванием значения из ST(0). Результат помещается в ST(i – 1)
FMUL	Умножает значение ST(0) на ST(1). Результат помещается в ST(0)

Окончание табл. 7.9

Команда	Действие
FMULST(i)	Умножает значение ST(0) на ST(i). Результат помещается в ST(0)
FMULST(i) , ST	Умножает значение ST(0) на ST(i). Результат помещается в ST(i)
FMULPST(i) , ST	Умножает значение ST(0) на ST(i) с выталкиванием значения из ST(0). Результат помещается в ST(i – 1)
FDIV	Делит значение ST(0) на ST(1). Результат помещается в ST(0)
FDIVST(i)	Делит значение ST(0) на ST(i). Результат помещается в ST(0)
FDIVST(i), ST	Делит значение ST(0) на ST(i). Результат помещается в ST(i)
FDIVPST(i), ST	Делит значение ST(0) на ST(i) с выталкиванием значения из ST(0). Результат помещается в ST(i – 1)
FDIVRST(i), ST	Делит значение ST(i) на ST(0). Результат помещается в ST(i)
FDIVRPST(i), ST	Делит значение ST(i) на ST(0) с выталкиванием значения из ST(0). Результат помещается в ST(i – 1)

Дополнительные арифметические команды предназначены для вычисления различных функций. Их перечень приведен в табл. 7.10.

Таблица 7.10

Команда	Действие
FSQRT	Вычисляет квадратный корень из значения ST(0). Результат помещается в ST(0)
FABS	Вычисляет модуль значения ST(0). Результат помещается в ST(0)
FCHS	Изменение знака ST(0) на противоположный
FXTRACT	Выделение мантиссы и порядка из значения ST(0). Мантисса помещается в ST(0) и представляется вещественным числом с нулевым порядком. Порядок заносится в ST(1) и представляется вещественным числом со знаком, имеющим значение истинного порядка числа (без учета фиксированного смещения)
FSCALE	Команда, обратная FXTRACT. ST(0) – мантисса числа, ST(1) – порядок числа. $ST(0) = ST(0) * 2^{ST(1)}$

Окончание табл. 7.10

Команда	Действие
FRNDINT	Округляет до целого значение в ST(0) в соответствии с порядком округления, заданным регистром CWR
FPREM	Нахождение частичного остатка от деления путем последовательного вычитания не более 64 раз содержимого ST(1) из ST(0)
FSIN	Вычисление sin(x), x расположен в ST(0). Результат помещается в ST(0). $-2^{63}<=x<=2^{63}$
FCOS	Вычисление cos(x), x расположен в ST(0). Результат помещается в ST(0). $-2^{63}<=x<=2^{63}$
FSINCOS	Вычисление sin(x) и cos(x). ST(1)=sin(ST(0)). ST(0)=cos(ST(0)). $-2^{63}<=x<=2^{63}$
FTAN	Вычисление частичного тангенса tg(a)=x/y. ST(0)=x, ST(1)=y. $-2^{63}<=a<=2^{63}$
FPATAN	Вычисление арктангенса a=arctg(x/y). ST(0)=x, ST(1)=y. ST(0)=x, ST(1)=y. Результат заносится в ST(0). $0 <\|y\|<\|x\|<\infty$
F2XM1	Вычисление 2^x-1. $-1 < x < 1$
FYL2X	Вычисление $y*Log_2(x)$. x – ST(0), y – ST(1). $0< x <\infty$, $-\infty< y < +\infty$.
FYL2XP1	Вычисление $y*Log_2(x+1)$. x – ST(0), y – ST(1). $0<= \|x\| <1-2^{0.5}/2$, $-\infty< y < +\infty$.

Группа команд управления сопроцессором предназначена для выгрузки/загрузки управляющих регистров, анализа и установки значений флагов сопроцессора. Эти команды в качестве операнда всегда имеют участок памяти определенной длины. Команды управления приведены в табл. 7.11.

Таблица 7.11

Команда	Действие
FINIT	Инициализация сопроцессора. Эквивалентна последовательности команд WAITFNINIT. CWR=037h, SWR=0, TWR=0FFh, DPR=0, IPR=0
FNINIT	Инициализация сопроцессора без ожидания обработки исключительных ситуаций
FLDCW источник	Загрузка CWR из слова в памяти
FSTCW приемник	Запись CWR в слово памяти. Эквивалентна WAITFNSTCW

Окончание табл. 7.11

Команда	Действие
FNSTCW приемник	Запись CWR в слово памяти без ожидания обработки исключительных ситуаций
FSTSW приемник	Запись SWR в слово памяти. Эквивалентна WAITFNSTSW
FNSTSW приемник	Запись SWR в слово памяти без ожидания обработки исключительных ситуаций
FSAVE приемник	Сохранение полного состояния сопроцессора в участок памяти 94 или 108 байт. Эквивалентна WAITFNSAVE
FNSAVE приемник	Сохранение полного состояния сопроцессора без ожидания обработки исключительных ситуаций
FXSAVE приемник (Pentium II)	Быстрое сохранение полного состояния сопроцессора в участке памяти размером 512 байт.
FRSTOR источник	Восстановление полного состояния сопроцессора, сохраненного командой FSAVE
FXSTOR источник (Pentium II)	Восстановление полного состояния сопроцессора, сохраненного командой FXSAVE
FSTENV приемник	Сохранение пяти вспомогательных регистров в приемнике (CWR, SWR, TWR, DPR, IPR). Эквивалентна WAIT FNSTENV
FNSTENVприемник	Сохранение пяти вспомогательных регистров в приемнике (CWR, SWR, TWR, DPR, IPR) без ожидания обработки исключительных ситуаций
FLDENV источник	Загрузка из памяти (14 или 28 байт) пяти вспомогательных регистров сопроцессора (CWR, SWR, TWR, DPR, IPR)
FWAIT WAIT	Ожидание готовности сопроцессора
FCLEX	Обнуление флагов исключений в регистре SWR (PE, UE, OE, ZE, DE, IE, ES, SE, B). Эквивалентна WAIT FNCLEX
FNCLEX	Обнуление флагов исключений в регистре SWR (PE, UE, OE, ZE, DE, IE, ES, SE, B) без ожидания обработки исключительных ситуаций
FINCSTP	Поле TOP в CSW увеличивается на 1
FDECSTP	Поле TOP в CSW уменьшается на 1
FFREE ST(i)	Освобождение регистра данных ST(i)

Обработка исключительных ситуаций

Исключение – внутреннее прерывание процессора, возникающее в ходе вычислительного процесса. В сопроцессоре может возникать 6 типов исключений. При возникновении какого-либо исключения устанавливается соответствующий бит в регистре состояния SWR.

Обработка исключения может выполняться двумя способами:

- обработка самим сопроцессором (маскированная обработка);
- программная обработка посредством реализации обработки возникающих прерываний (не замаскированная обработка).

Способ обработки зависит от значений соответствующих битов регистра управления сопроцессором CWR. Для каждого вида исключения в нем есть соответствующий бит. Если этот бит установлен в 1 (исключение маскировано), исключение обрабатывается самим сопроцессором. В противном случае (исключение не замаскировано) обработка должна быть выполнена программным путем.

Порядок маскированной обработки по видам исключений приведен в табл. 7.12.

Таблица 7.12

Исключение	Действие
Недействительная операция	Возникает при работе со стеком или арифметическими операциями. Причины: • загрузка операнда в непустой регистр стека; • попытка извлечь операнд из пустого регистра; • использование операнда с недопустимым значением. В SWR устанавливается флаг IF. Маскируется битом IM в CWR. Если в SWR флаг SF=1, возникло при работе со стеком, SF=0 – неверный операнд. Маскированная обработка – в регистр стека записывается тихое не-число
Деление на ноль	Возникает в командах деления. В SWR устанавливается флаг ZE. Маскируется битом ZM в CWR. Маскированная реакция – формирование результата в виде знаковой бесконечности
Денормализованный операнд	Возникает при попытке выполнить операцию с денормализованным операндом. В SWR устанавливается флаг DE. Маскируется битом DM в CWR. Маскированная реакция – только установка флага DE

Окончание табл. 7.12

Исключение	Действие
Переполнение	Возникает если порядок числа слишком велик для формата приемника. В SWR устанавливается флаг OE. Маскируется битом OM в CWR. Маскированная реакция – формирование максимального представимого значения или знаковой бесконечности
Антипереполнение	Возникает если порядок числа слишком мал для формата приемника. В SWR устанавливается флаг UE. Маскируется битом UM в CWR. Маскированная реакция – формирование минимального представимого значения.
Неточный результат	Возникает когда результат невозможно точно представить в формате приемника и его нужно округлять. В SWR устанавливается флаг PE. Маскируется битом PM в CWR. Характер выполненного округления показывает бит C1: • C1=0 – результат усечен; • C1=1 – результат округлен в большую сторону. Маскированная реакция – округление числа

При немаскированной обработке появление ошибки в сопроцессоре вызывает прерывания 10h и 75h. Программист должен написать обработчик одного из этих прерываний, выполняющий коррекцию возникшей ошибки.

Пример программы с использованием сопроцессора

Программа, которая вводит вещественное число в формате числа с фиксированной точкой, преобразует его во внутренний формат сопроцессора и выполняет обратное преобразование из внутреннего формата сопроцессора в строку символов.

```
model small
.486
stack 100h
.data
ent     db    0Ah, 0Dh, 'Enter: $'
rsl     db    0Ah, 0Dh, 'Result: $'
opre    db    0Ah, 0Dh, 'Operation: $'
opr     db    ?

inf     db    'Inf$'
nan     db    'NaN$'
```

```
n1      db    40
len1    db    ?
n1d     db    40 dup(?)

n2      db    40
len2    db    ?
n2d     db    40 dup(?)

result1 dq    ?
result2 dq    ?
result  dq    ?
resstr  db    40 dup(?)

ext     db    0Ah, 0Dh, 'Exit – Esc, continue – any other key...$'
.code
        startupcode
        finit
        lea    dx, ent + 2
        jmp    _wrtk
_begin:
        lea    dx, ent
_wrtk:  mov    ah, 09h
        int    21h

        lea    dx, n1
        mov    ah, 0ah
        int    21h

        lea    dx, opre
        mov    ah, 09h
        int    21h

        mov    ah, 01h
        int    21h

        mov    opr, al

        lea    dx, ent
        mov    ah, 09h
        int    21h

        lea    dx, n2
        mov    ah, 0ah
        int    21h
```

```
        push   offset result1
        push   offset n1d
        call   StrToDouble
        add    sp, 2

        push   offset result2
        push   offset n2d
        call   StrToDouble
        add    sp, 2

        lea    dx, rsl
        mov    ah, 09h
        int    21h

        cmp    opr, '+'
        je     _add
        cmp    opr, '-'
        je     _sub
        cmp    opr, '*'
        je     _mul
        cmp    opr, '/'
        je     _div
        jmp    _end

_add:
        fld    result1
        fld    result2
        faddp  st(1), st(0)              ; fadd и вытолкнуть после выполнения st(0)
        fstp   result
        jmp    _write
_sub:
        fld    result1
        fld    result2
        fsubp  st(1), st(0)
        fstp   result

        jmp    _write
_mul:
        fld    result1
        fld    result2
        fmulp  st(1), st(0)
        fstp   result

        jmp    _write
```

```
_div:
        fld    result1
        fld    result2
        fdivp  st(1), st(0)
        fstsw  ax
        test   ax, 100b
        je     _nr
        lea    dx, inf
        jmp    _wrtn
_nr:    test   ax, 1
        je     _nk
        lea    dx, nan
        jmp    _wrtn
_nk:    fstp   result
_write:
        push   offset resstr
        push   offset result
        call   DoubleToStr
        add    sp, 2

        lea    dx, resstr
_wrtn:  mov    ah, 09h
        int    21h

_end:
        lea    dx, ext
        mov    ah, 09h
        int    21h

        mov    ah, 08h
        int    21h
        cmp    al, 27
        jnz    _begin

        exitcode 0
DoubleToStr proc  near
        push   bp
mov     bp, sp
        sub    sp, 4              ; выделяем 4 байта в стеке
push   ax bx dx cx di
pushf

        fnstcw [bp-4]             ; сохраним значение регистра управления

        fnstcw [bp-2]
```

```
        and    word ptr [bp - 2], 1111001111111111b; биты 11–10 управление
округлением, 11 – к нулю
        or     word ptr [bp - 2], 0000110000000000b

        fldcw  [bp - 2]          ; Запись нового значения регистра управления

        mov    bx, [bp + 4]

        fld    qword ptr[bx]     ; заталкиваем в стек сопроцессора число

ftst
        fstsw  ax
        and    ah, 1
        cmp    ah, 1
        jne    @@NBE
        mov    bx, [bp + 6]
        mov    byte ptr[bx], '-'
        inc    word ptr[bp + 6]

@@NBE:  fabs
        fst    st(1)
        fst    st(2)
        frndint
        fsub   st(2), st(0)

        mov    word ptr[bp - 2], 10
        fild   word ptr[bp - 2]

        fxch   st(1)
        xor    cx, cx

@@BG:   fprem
        fist   word ptr [bp - 2]
        push   word ptr [bp - 2]

        fxch   st(2)
        fdiv   st(0), st(1)
        frndint
        fst    st(2)

        inc    cx

        ftst ; сравнить st(0) с 0
        fstsw  ax                ; SR -> AX
        sahf ; AH вфлаги
```

```
      jnz    @@BG    ; если 14 бит SR == 0 (6 бит AH) (если zf == 0 прыжок)

      mov    ax, cx
      mov    bx, [bp + 6]

@@BFG:  pop    dx
      add    dx, '0'
      mov    byte ptr[bx], dl
      inc    bx
      loop   @@BFG

      fxch   st(3)
      fst    st(2)

      ftst
      fstsw  ax
      sahf
      jz     @@CNE

      mov    byte ptr[bx], '.'

      mov    cx, 16

@@BFR:  fmul   st(0), st(1)
      fst    st(2)
      frndint
      fsub   st(2), st(0)
      fist   word ptr [bp - 2]
      fxch   st(2)
      mov    ax, [bp - 2]
      add    ax, '0'
      inc    bx
      mov    byte ptr[bx], al

      loop   @@BFR

@@NIL:  cmp    byte ptr[bx], '0'
      jne    @@CNR
      dec    bx
      jmp    @@NIL
@@CNR:  inc    bx
@@CNE:  mov    byte ptr[bx], '$'

      fstp   st(0)
      fstp   st(0)
```

```
        fstp   st(0)
fstp   st(0)

        fldcw  [bp - 4]          ; восстановим настройки сопроцессора

popf
        pop    di cx dx bx ax
        add    sp, 4
        pop    bp
        ret
DoubleToStr  endp


StrToDouble proc  near
        push   bp
        mov    bp, sp
sub    sp, 2                  ; выделяем 2 байта в стеке
push   ax bx dx cx di
pushf
        mov    word ptr[bp - 2], 10      ; помещаем в выделенные 2 байта 10
        fild   word ptr[bp - 2]          ; заталкиваем в стек сопроцессора 10
        fldz                             ; заталкиваем в стек сопроцессора 0
        mov    di, 0
        mov    bx, [bp + 4]              ; помещаем в bx адрес из стека
cmp    byte ptr[bx], '-'
        jne    @@BPN
        inc    bx
        mov    di, 1
@@BPN:  movsx  ax, byte ptr [bx]
        cmp    ax, '.'
        je     @@PNT1
        cmp    ax, 0dh
        jne    @@CNT
        fxch   st(1)
        fstp   st(0)
        jmp    @@REN
@@CNT:  sub    ax, '0'
mov    word ptr[bp - 2], ax
        fmul   st(0), st(1)              ; умножаем число на вершине стека на 10
        fiadd  word ptr[bp - 2]    ; добавляем к числу на вершине стека то что было в ax
inc    bx
        jmp    @@BPN
@@PNT1:
        xor    cx, cx
@@BEG:  inc    bx
```

```
        movsx  ax, byte ptr [bx]
        cmp    ax, 0dh
        je     @@END
        loop   @@BEG
@@END:  dec    bx
        fxch   st(1)
        fldz
@@APN:  movsx  ax, [bx]
        cmp    ax, '.'
        je     @@PNT2
        sub    ax, '0'
        mov    word ptr[bp - 2], ax
        fiadd  word ptr[bp - 2]
        fdiv   st(0), st(1)
        dec    bx
        jmp    @@APN
@@PNT2:
;
        fxch   st(1)                ; меняем число 10 и остаток местами
        fstp   st(0)                ; выталкиваем 10
        faddp  st(1)        ; складываем целую и дробную части
@@REN:
        cmp    di, 1
        jne    @@CYK
        fchs
@@CYK:  mov    bx, [bp + 6]         ; помещаем в bx адрес из стека
        fstp   qword ptr [bx]       ; помещаем по адресу из стека число
popf
        pop    di cx dx bx ax
        add    sp, 2
        pop    bp
        ret
StrToDouble  endp

end
```

Задания для самостоятельного выполнения

1. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3+5x^2-4x-20$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать короткий формат вещественных чисел.
2. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3-7x^2-9x+49$ меняет знак на противоположный, найти ее

корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать двойной формат вещественных чисел.

3. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3+7x^2-16x-112$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать расширенный формат вещественных чисел.
4. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3+8x^2+x-42$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать короткий формат вещественных чисел.
5. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3-8x^2+x+42$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать двойной формат вещественных чисел.
6. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3+x^2-12$ хменяет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать расширенный формат вещественных чисел.
7. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3-16x$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать короткий формат вещественных чисел.
8. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3-6x^2-16x+96$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать двойной формат вещественных чисел.
9. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3-x^2-25x+25$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e. В противном случае вывести сообщение. Использовать расширенный формат вещественных чисел.
10. Ввести вещественные значения a, b (a<b) и e. Если на отрезке [a; b] функция $f(x)=x^3-7x^2+4x+12$ меняет знак на противоположный, найти ее корень на этом отрезке методом деления отрезка пополам с точностью e.

В противном случае вывести сообщение. Использовать короткий формат вещественных чисел.

Контрольные вопросы

1. Из каких групп регистров состоит программная модель сопроцессора?
2. Что такое стек регистров сопроцессора?
3. Что содержит регистр SWR?
4. Для чего предназначен регистр CWR?
5. Какие форматы вещественных чисел обрабатывает сопроцессор и чем они отличаются?
6. Какие специальные форматы использует сопроцессор?
7. Какие команды передачи данных имеются в сопроцессоре?
8. Какие команды загрузки констант имеются в сопроцессоре?
9. Какие команды сравнения имеются в сопроцессоре?
10. Какие команды арифметических операций имеются в сопроцессоре?
11. Какие дополнительные арифметические команды имеются в сопроцессоре?
12. Какие команды управления имеются в сопроцессоре?
13. Как сопроцессор обрабатывает исключительные ситуации?

Заключение

В пособии рассмотрены вопросы программирования процессоров Intel на языке ассемблера в реальном режиме, одним из применений которого является использование во встраиваемых системах. Материал пособия предназначен для студентов технических вузов в рамках курса «Машинно-ориентированное программирование» и может быть использован в качестве для наполнения цикла лабораторных работ.

Пособие разработано в рамках выполнения базовой части государственного задания в сфере научной деятельности (проект № 3442 "Информационно-алгоритмическое обеспечение систем цифрового управления, автономной высокоточной навигации и технического зрения для перспективных летательных аппаратов: разработка теоретических основ проектирования, алгоритмов, способов эффективной и надежной программной реализации, использование высокопроизводительной вычислительной инфраструктуры для экспериментального моделирования").

Библиографический список

1. Голубь Н.Г. Искусство программирования на Ассемблере: лекции и упражнения. – СПб.: ДиаСофтЮП, 2002.
2. Магда Ю.С. Ассемблер для процессоров Intel Pentium. – СПб.: Питер, 2006.
3. Калашников О.А. Ассемблер? Это просто! Учимся программировать. – БХВ-Петербург, 2011.
4. Аблязов Р.З. Программирование на ассемблере для платформы x86-64. – М.: ДМК Пресс, 2011.
5. Зубков С.В. Ассемблер для Dos, Windows и Unix. – М.: ДМК Пресс, 2004.
6. Пирогов В.Ю. Assembler: учебный курс. – М.: Изд-во «Нолидж», 2001.
7. Крупник А.Б. Изучаем Ассемблер. – СПб.: Питер, 2005.
8. Пильщиков В.Н. Программирование на языке ассемблера IBM PC. – М.: Диалог-МИФИ, 1999.
9. Юров В.И. Assembler: учебник для вузов. – СПб.: Питер, 2003.
10. Рудольф Марек. Ассемблер на примерах: базовый курс. – СПб.: Наука и Техника, 2005.
11. Рудаков П.И., Финогенов К.Г. Язык ассемблера: уроки программирования. – М.: Диалог-МИФИ, 2001.
12. Кип Ирвин. Язык ассемблера для процессоров Intel. – М.: Издательский дом «Вильямс», 2005.
13. Авдеев В.А. Периферийные устройства. Интерфейсы, схемотехника, программирование. – М.: ДМК Пресс, 2009.
14. Дроздов С.Н., Калачев Д.П. Методическая разработка к лабораторным работам «Программирование на языке ассемблера ПЭВМ IBM PC». – Таганрог: Изд-во ТРТУ, 1997.

Содержание

Учебное издание

Скороход Сергей Васильевич
Селянкин Владимир Васильевич
Дроздов Сергей Николаевич
Калачев Дмитрий Петрович
Хусаинов Наиль Шавкятович

Основы программирования микропроцессоров Intel для встраиваемых систем

Учебное пособие

Ответственный за выпуск Скороход С.В.
Редакторы: Проценко И. А., Селезнева Н. И.
Корректоры: Проценко И. А., Селезнева Н. И.

Подписано в печать 28.12.2016
Заказ № 206 Тираж 50 экз.
Формат 60x84 $^{1}/_{16}$ Печ.л. – 5,1 Уч. изд. л. – 5,0

Издательство Южного федерального университета
344091, г. Ростов-на-Дону, пр. Стачки, 200/1.
Тел. (863) 2478051.
Отпечатано в Отделе полиграфической,
корпоративной и сувенирной продукции
ИПК КИБИ МЕДИА ЦЕНТРА ЮФУ.
ГСП 17А, Таганрог, 28, Энгельса, 1.
Тел. (8634) 371717, 371615.