МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение высшего профессионального образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Утверждаю:
Зав. каф АОИ
профессор
_____ Ю.П. Ехлаков
«____» _____2010 г.

Методические указания

по выполнению лабораторных работ и организации самостоятельной работы студентов по дисциплине:

«*Операционные среды, системы и оболочки*»

для студентов направления подготовки 080700 «Бизнес-информатика»

Разработчик:
доцент каф. АОИ
_____ Ю.Б. Гриценко
«____» _____ 2010 г.

Томск – 2010

Содержание

Часть 1. Семестр 2 (курс 1)

ЛАБОРАТОРНАЯ РАБОТА №1 «*Файлы пакетной обработки*»

1.1. Цель работы

Целью данной работы является:

- изучение назначения и основных возможностей командных файлов (Файлов пакетной обработки) операционных систем, построенных на платформе Windows NT;
- знакомство со специальными командами, используемыми в командных файлах;
- исследование стандартных потоков ввода-вывода и их перенаправление.

1.2. Язык командных файлов

Командный файл – это текстовый файл (в коде ASCII), состоящий из группы команд. Правила идентификации командных файлов совпадают с общими правилами идентификации файлов. Единственное исключение — командный файл всегда записывается на диск с расширением «.BAT» и/или «.CMD» (для операционных систем Windows на платформе NT).

Обратиться к командному файлу крайне просто. Набирается команда старта – имя файла, и нажимается клавиша Enter. После введения команды файл выбирается из рабочего каталога указанного или рабочего диска. Если в рабочем каталоге его нет, то поиск файла будет производиться в каталогах, описанных системной переменной %PATH%. При нахождении файла первая из его команд загружается в память, отображается на экране и выполняется. Этот процесс повторяется последовательно для всех команд файла (от первой до последней команды).

Выполнение командного файла можно прервать в любой момент, нажав на клавиши Ctrl-Break (Ctrl-C).

Организация командного файла. Существует несколько способов организации командных файлов. Файл можно создать с помощью любого текстового редактора или введением команд непосредственно с клавиатуры. В этом случае ввод оформляется файлом и записывается на диск.

В сеансе DOS клавиатура называется «CON» (CONsole) Для организации файла используется команда «COPY CON:». Наберите команду и имя создаваемого файла. Например, для создания файла «SAMPLE.BAT» введите:

```
C:\>COPY CON: SAMPLE.BAT
```

После этого введите составляющие файл команды. Набрав последнюю команду, одновременно нажмите клавиши Ctrl-Z (или функциональную клавишу F6) и клавишу Enter.

Стандартные потоки ввода-вывода и перенаправление потоков. Термин CONsole используется для обозначения стандартных потоков ввода-вывода. Когда говорят о вводе с консоли, подразумевается ввод с клавиатуры. Когда говорят о выводе на консоль, подразумевают вывод на экран монитора. Существуют специальные символы для перенаправления стандартных потоков ввода-вывода.

> `приемник` — перенаправить стандартный вывод в приемник (если файл-приемник существует, то он будет создан заново).

>> `приемник` — перенаправить стандартный вывод в приемник (если файл-приемник существует, то он будет сохранен, а информация будет записана в конец файла).

< `источник` — перенаправить стандартный ввод из источника.

`передатчик` **|** `приемник` — передает вывод одной команды на вход другой.

Замещаемые параметры. Внутри командного файла допускается использование замещаемых параметров. Параметр — это символьная переменная, расположенная в командной строке после имени команды. Он содержит дополнительную информацию, необходимую операционной системе при обработке команды. Параметром, например, может быть имя файла, к которому относится действие команды. Замещаемый параметр — это специальная переменная, которая в процессе выполнения команды подменяется обычным параметром (например, именем файла). В командном файле замещаемый параметр обозначается знаком процента `%` и цифрой от `0` до `9`. Таким образом, командный файл может включать до десяти замещаемых параметров. Символьные переменные, предназначенные для подмены замещающего параметра, вводятся в командной строке при обращении к команд-

ному файлу — набирается команда старта (имя файла) и список параметров в порядке, соответствующем последовательности замещаемых параметров внутри файла.

Параметры заменяются в порядке следования символьных переменных в командной строке. Первая переменная подменяет параметр `%1`, вторая – параметр `%2` и т.д. Вместо замещаемого параметра `%0` автоматически подставляется спецификация (имя) командного файла.

При введении замещаемых параметров командный файл становится более гибким. Поясним это на примере. Предположим, что на диске имеется несколько файлов, которые нужно копировать после каждой корректировки. В рассмотренном выше примере командный файл использовался для копирования конкретного файла. Этим же командным файлом можно воспользоваться и для копирования любого файла. В этом случае вместо имени копируемого файла подставляется замещаемый параметр. Имя копируемого файла будет вводиться в командной строке при обращении к командному файлу.

Назовем наш командный файл «COPYALL.BAT». Введем в нем:

```
COPY %1 A:
```

При обращении к файлу набирается его имя и через пробел – имя копируемого файла (в нашем примере «SHOPLIST.DOC»). Введите команду:

```
C:\>COPYALL.BAT SHOPLIST.DOC
```

На экран выводится следующая команда:

```
C:\>COPY SHOPLIST.DOC A:
 1 File(s) copied
```

DOS автоматически подставила имя файла на место замещаемого параметра `%1`. Усложним пример. Организуем командный файл «DIFNUM.BAT», автоматически копирующий любой указанный файл и присваивающий копии любое указанное имя:

```
COPY %1 A:%2
```

Для обращения к этому файлу наберите его имя, имя копируемого файла, в нашем примере «NEW.DOC», и имя копии «OLD.DOC»:

```
C:\>DIFNUM NEW.DOC OLD.DOC
```

На экране появляется следующая команда файла «DIFNUM.BAT»:

```
C:\>COPY NEW.DOC A:OLD.DOC
 1 File(s) copied
```

Первое имя в командной строке «NEW.DOC» поставлено вместо замещаемого параметра `%1`. Второе имя «OLD.DOC» – вместо замещаемого параметра `%2`.

Замещаемые параметры и замещаемые символы. Параметр в командной строке команды старта командного файла может включать замещаемые символы «?» и «*». Если замещаемый символ вводится для обозначения группы параметров, то команда выполняется по количеству параметров в группе (т.е. один раз для каждого параметра). Рассмотрим командный файл:

```
COPY %1 CON:
```

Этот файл копирует на экран (CON) файл, описанный замещаемым параметром `%1` (DISPLAY.BAT). Имя копируемого файла указывается в командной строке при обращении к командному файлу. Если указанный файл найден, его содержимое выводится на экран.

Этот файл копирует на экран (con) файл, описанный замещаемым параметром `%1`. Имя копируемого файла указывается в командной строке при обращении к командному файлу. Если указанный файл найден, его содержимое выводится на экран. Итак, командный файл «DISPLAY.BAT» записан на диск. Введем команду:

```
C:\>DISPLAY *.TXT
```

Все файлы рабочего диска с соответствующей спецификацией будут выведены на экран. Если имя копируемого файла включает обозначение процента, то при введении его в командную строку знак процента набирается два раза подряд. Например, имя «HIHO%.TXT» в командной строке должно быть представлено как «HIHO%%.TXT».

1.3. Некоторые команды DOS (Windows)

Для получения полного списка команд DOS, поддерживаемых вашей операционной системой Windows, построенной на платформе NT, необходимо ввести команду[1]:

HELP

Вот ее возможный результат:

```
Для получения сведений об определенной команде
наберите HELP <имя команды>
ASSOC  — Вывод либо изменение сопоставлений по
расширениям имен файлов.
AT     — Выполнение команд и запуск программ
по расписанию.
ATTRIB — Отображение и изменение атрибутов
файлов.
BREAK  — Включение/выключение режима обработки
комбинации клавиш CTRL+C.
CACLS  — Отображение/редактирование списков
управления доступом (ACL) к файлам.
CALL   — Вызов одного пакетного файла из дру-
гого.
CD     — Вывод имени либо смена текущей папки.
CHCP   — Вывод либо установка активной кодовой
страницы.
CHDIR  — Вывод имени либо смена текущей папки.
CHKDSK — Проверка диска и вывод статистики.
CHKNTFS— Отображение или изменение выполнения
проверки диска во время загрузки.
CLS    — Очистка экрана.
CMD    — Запуск еще одного интерпретатора ко-
мандных строк Windows.
COLOR  — Установка цвета текста и фона, ис-
пользуемых по умолчанию.
COMP   — Сравнение содержимого двух файлов или
двух наборов файлов.
```

[1] Синтаксис представлен для ОС Windows XP построенной на базе технологии NT, в ОС MS-DOS и Windows 9x количество аргументов и команд несколько меньше.

COMPACT— Отображение/изменение сжатия файлов в разделах NTFS.

CONVERT— Преобразование дисковых томов FAT в NTFS. Нельзя выполнить преобразование текущего активного диска.

COPY — Копирование одного или нескольких файлов в другое место.

DATE — Вывод либо установка текущей даты.

DEL — Удаление одного или нескольких файлов.

DIR — Вывод списка файлов и подпапок из указанной папки.

DISKCOMP— Сравнение содержимого двух гибких дисков.

DISKCOPY—Копирование содержимого одного гибкого диска на другой.

DOSKEY — Редактирование и повторный вызов командных строк; создание макросов.

ECHO — Вывод сообщений и переключение режима отображения команд на экране.

ENDLOCAL— Конец локальных изменений среды для пакетного файла.

ERASE — Удаление одного или нескольких файлов.

EXIT — Завершение работы программы CMD.EXE (интерпретатора командных строк).

FC — Сравнение двух файлов или двух наборов файлов и вывод различий между ними.

FIND — Поиск текстовой строки в одном или нескольких файлах.

FINDSTR— Поиск строк в файлах.

FOR — Запуск указанной команды для каждого из файлов в наборе.

FORMAT — Форматирование диска для работы с Windows.

FTYPE — Вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов.

GOTO — Передача управления в отмеченную строку пакетного файла.

GRAFTABL— Позволяет Windows отображать расширенный набор символов в графическом режиме.

HELP — Выводит справочную информацию о командах Windows.

IF — Оператор условного выполнения команд в пакетном файле.

LABEL — Создание, изменение и удаление меток тома для дисков.

MD — Создание папки.

MKDIR — Создание папки.

MODE — Конфигурирование системных устройств.

MORE — Последовательный вывод данных по частям размером в один экран.

MOVE — Перемещение одного или нескольких файлов из одной папки в другую.

PATH — Вывод либо установка пути поиска исполняемых файлов.

PAUSE — Приостановка выполнения пакетного файла и вывод сообщения.

POPD — Восстановление предыдущего значения текущей активной папки, сохраненного с помощью команды PUSHD.

PRINT — Вывод на печать содержимого текстовых файлов.

PROMPT — Изменение приглашения в командной строке Windows.

PUSHD — Сохранение значения текущей активной папки и переход к другой папке.

RD — Удаление папки.

RECOVER— Восстановление читаемой информации с плохого или поврежденного диска.

REM — Помещение комментариев в пакетные файлы и файл CONFIG.SYS.

REN — Переименование файлов и папок.

RENAME — Переименование файлов и папок.

REPLACE— Замещение файлов.

RMDIR — Удаление папки.

SET — Вывод, установка и удаление переменных среды Windows.

SETLOCAL— Начало локальных изменений среды для пакетного файла.

SHIFT — Изменение содержимого (сдвиг) подставляемых параметров для пакетного файла.

SORT — Сортировка ввода.

```
    START  — Запуск программы или команды в от-
дельном окне.
    SUBST  — Сопоставляет заданному пути имя дис-
ка.
    TIME   — Вывод и установка системного времени.
    TITLE  — Назначение заголовка окна для текуще-
го сеанса интерпретатора командных строк CMD.EXE.
    TREE   — Графическое отображение структуры па-
пок заданного диска или заданной папки.
    TYPE   — Вывод на экран содержимого текстовых
файлов.
    VER    — Вывод сведений о версии Windows.
    VERIFY — Установка режима проверки правильно-
сти записи файлов на диск.
    VOL    — Вывод метки и серийного номера тома
для диска.
    XCOPY  — Копирование файлов и дерева папок.
```

Чтобы получить информацию о какой-либо команде операционной системы можно также в командной строке набрать имя команды и через пробел указать знак /?. Например,

C:\>PAUSE /?

Далее приводится основной синтаксис некоторых команд, необходимых для выполнения лабораторной работы.

ECHO

ECHO [ON | OFF] — переключение режима отображения команд на экране.

ECHO [сообщение] — вывод сообщений.

Введите ECHO без параметра для определения текущего значения этой команды.

Введите ECHO. (с точкой) для получение пустой строки.

@ — знак экранирования. Отключает вывод на экран текущей строки.

GOTO — передача управления содержащей метку строке пакетного файла.

GOTO метка

метка — строка пакетного файла, оформленная как метка.

Метка должна находиться в отдельной строке и начинаться с двоеточия.

IF — оператор условного выполнения команд в пакетном файле.

IF [NOT] ERRORLEVEL число команда
IF [NOT] строка1==строка2 команда
IF [NOT] EXIST имя_файла команда

NOT — обращает истинность условия: истинное условие становится ложным, а ложное — истинным.

ERRORLEVEL число — условие является истинным, если код возврата последней выполненной программы не меньше указанного числа.

строка1==строка2 — это условие является истинным, если указанные строки совпадают.
IF (%1)==() — проверка на пустой параметр.

EXIST имя_файла — это условие является истинным, если файл с указанным именем существует.
команда — задает команду, выполняемую при истинности условия. За этой командой может следовать ключевое слово ELSE, служащее для указания команды, которая должна выполняться в том случае, если условие ложно.

Предложение ELSE должно располагаться в той же строке, что и команда, следующая за ключевым словом IF. Например:

```
IF EXIST имя_файла. (
del имя_файла.
```

) ELSE (
echo имя_файла. missing.
)

Следующий пример содержит ОШИБКУ, поскольку команда del должна заканчиваться переходом на новую строку:

IF EXIST имя_файла. del имя_файла. ELSE echo имя_файла. missing

Следующий пример также содержит ОШИБКУ, поскольку команда ELSE должна располагаться в той же строке, что и команда, следующая за IF:

IF EXIST имя_файла. del имя_файла.
ELSE echo имя_файла. missing

Вот правильный пример, где все команды расположены в одной строке:

IF EXIST имя_файла. (del имя_файла.) ELSE echo имя_файла. missing

PAUSE — приостановка выполнения пакетного файла и вывод сообщения:
Для продолжения нажмите любую клавишу . . .

DIR — вывод списка файлов и подкаталогов из указанного каталога.

DIR [диск:][путь][имя_файла] [/A[[:]атрибуты]] [/B] [/C] [/D] [/L] [/N] [/O[[:]порядок]] [/P] [/Q] [/S] [/T[[:]время]] [/W] [/X] [/4]

[диск:][путь][имя_файла]
Диск, каталог и/или файлы, которые следует включить в список.

/A Вывод файлов с указанными атрибутами.
атрибуты:
- D Каталоги
- R Доступные только для чтения

- H Скрытые файлы
- A Файлы для архивирования
- S Системные файлы
- Префикс «-» имеет значение НЕ

/B Вывод только имен файлов.

/C Применение разделителя групп разрядов для вывода размеров файлов (по умолчанию). Для отключения этого режима служит ключ /-C.

/D Вывод списка в несколько столбцов с сортировкой по столбцам.

/L Использование нижнего регистра для имен файлов.

/N Отображение имен файлов в крайнем правом столбце.

/O Сортировка списка отображаемых файлов.

порядок:

- N По имени (алфавитная)
- S По размеру (сперва меньшие)
- E По расширению (алфавитная)
- D По дате (сперва более старые)
- G Начать список с каталогов
- Префикс «-» обращает порядок

/P Пауза после заполнения каждого экрана.

/Q Вывод сведений о владельце файла.

/S Вывод списка файлов из указанного каталога и его подкаталогов.

/T Выбор поля времени для отображения и сортировки

время:

- C Создание
- A Последнее использование
- W Последнее изменение

/W Вывод списка в несколько столбцов.

/X Отображение коротких имен для файлов, чьи имена не соответствуют стандарту 8.3. Формат аналогичен выводу с ключом /N, но короткие имена файлов выводятся слева от длинных. Если короткого имени у файла нет, вместо него выводятся пробелы.

/4 Вывод номера года в четырехзначном формате

Стандартный набор ключей можно записать в переменную среды DIRCMD. Для отмены их действия введите в команде те же ключи с префиксом «-», например: /-W.

MD — создание каталога.

MKDIR [диск:]путь
MD [диск:]путь

CD — вывод имени либо смена текущего каталога.

CHDIR [/D] [диск:][путь]
CHDIR [..]
CD [/D] [диск:][путь]
CD [..]

.. обозначает переход в родительский каталог.

Команда CD диск: отображает имя текущего каталога указанного диска.

Команда CD без параметров отображает имена текущих диска и каталога.

Параметр /D используется для одновременной смены текущих диска и каталога.

RD — удаление каталога.

RMDIR [/S] [/Q] [диск:]путь
RD [/S] [/Q] [диск:]путь

/S Удаление дерева каталогов, т. е. не только указанного каталога, но и всех содержащихся в нем файлов и подкаталогов.

/Q Отключение запроса подтверждения при удалении дерева каталогов с помощью ключа /S.

COPY — копирование одного или нескольких файлов в другое место.

COPY [/D] [/V] [/N] [/Y | /-Y] [/Z] [/A | /B] источник [/A | /B]
[+ источник [/A | /B] [+ ...]] [результат [/A | /B]]

источник Имена одного или нескольких копируемых файлов.

/A Файл является текстовым файлом ASCII.

/B Файл является двоичным файлом.

/D Указывает на возможность создания зашифрованного файла результат Каталог и/или имя для конечных файлов.

/V Проверка правильности копирования файлов.

/N Использование, если возможно, коротких имен при копировании файлов, чьи имена не удовлетворяют стандарту 8.3.

/Y Подавление запроса подтверждения на перезапись существующего конечного файла.

/-Y Обязательный запрос подтверждения на перезапись существующего конечного файла.

/Z Копирование сетевых файлов с возобновлением.

Ключ /Y можно установить через переменную среды COPYCMD.

Ключ /-Y командной строки переопределяет такую установку.

По умолчанию требуется подтверждение, если только команда COPY не выполняется в пакетном файле.

Чтобы объединить файлы, укажите один конечный и несколько исходных файлов, используя подстановочные знаки или формат «файл1+файл2+файл3+...».

REN — переименование одного или нескольких файлов.

RENAME [диск:][путь]имя_файла1 имя_файла2.
REN [диск:][путь]имя_файла1 имя_файла2.

Для конечного файла нельзя указать другой диск или каталог.

DEL — удаление одного или нескольких файлов.

DEL [/P] [/F] [/S] [/Q] [/A[[:]атрибуты]] имена
ERASE [/P] [/F] [/S] [/Q] [/A[[:]атрибуты]] имена

имена — Имена одного или нескольких файлов. Для удаления сразу нескольких файлов используются подстановочные знаки. Если указан каталог, из него будут удалены все файлы.

/P Запрос на подтверждение перед удалением каждого файла.

/F Принудительное удаление файлов, доступных только для чтения.

/S Удаление указанных файлов из всех подкаталогов.

/Q Отключение запроса на подтверждение при удалении файлов.

/A Отбор файлов для удаления по атрибутам.

атрибуты:

- S Системные файлы
- R Доступные только для чтения
- H Скрытые файлы
- A Файлы для архивирования
- Префикс «-» имеет значение НЕ

TYPE — вывод содержимого одного или нескольких текстовых файлов.

TYPE [диск:][путь]имя_файла

FOR — выполнение указанной команды для каждого файла набора.

FOR %переменная IN (набор) DO команда [параметры]

%переменная – подставляемый параметр;

(набор) – набор, состоящий из одного или нескольких файлов. Допускается использование подстановочных знаков;

команда – команда, которую следует выполнить для каждого файла;

параметры – параметры и ключи для указанной команды.

В пакетных файлах для команды FOR используется запись %%переменная вместо %переменная. Имена переменных учитывают регистр букв (%i отличается от %I).

Добавление поддерживаемых вариантов команды FOR при включении расширенной обработки команд:

FOR /D %переменная IN (набор) DO команда [параметры]

Если набор содержит подстановочные знаки, команда выполняется для всех подходящих имен каталогов, а не имен файлов.

FOR /R [[диск:]путь] %переменная IN (набор) DO команда [параметры]

Выполнение команды для каталога [диск:]путь, а также для всех подкаталогов этого пути. Если после ключа /R не указано имя каталога, выполнение команды начинается с текущего каталога.

Если вместо набора указана только точка (.), команда выводит список всех подкаталогов.

FOR /L %переменная IN (начало,шаг,конец) DO команда [параметры]

Набор раскрывается в последовательность чисел с заданными началом, концом и шагом приращения. Так, набор (1,1,5) раскрывается в (1 2 3 4 5), а набор (5,-1,1) заменяется на (5 4 3 2 1)

FOR /F [«ключи»] %переменная IN (набор) DO команда [параметры]

FOR /F [«options»] %variable IN («literal string») DO command [command-parameters]

FOR /F [«options»] %variable IN ('command') DO command [command-parameters]

или, если использован параметр usebackq:

FOR /F [«options»] %variable IN (filenameset) DO command [command-parameters]

FOR /F [«options»] %variable IN ('literal string') DO command [command-parameters]

FOR /F [«options»] %variable IN (`command`) DO command [command-parameters]

Набор содержит имена одного или нескольких файлов, которые по очереди открываются, читаются и обрабатываются. Обработка состоит в чтении файла, разбивки его на отдельные строки текста и выделения из каждой строки заданного числа подстрок (в том числе нуля). Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла. По умолчанию ключ

/F выделяет из каждой строки файла первое слово, очищенное от окружающих его пробелов. Пустые строки в файле пропускаются. Необязательные параметры «ключи» служит для переопределения заданных по умолчанию правил обработки строк. Ключи представляют собой заключенную в кавычки строку, содержащую указанные параметры. Ключевые слова:

eol=c — определение символа комментариев в конце строки (допускается задание только одного символа);

skip=n — число пропускаемых при обработке строк в начале файла;

delims=xxx — определение набора разделителей для замены заданных по умолчанию пробела и знака табуляции;

tokens=x,y,m-n — определение номеров подстрок, выделяемых из каждой строки файла и передаваемых для выполнения в тело цикла. При использовании этого ключа создаются дополнительные переменные. Формат m-n представляет собой диапазон подстрок с номерами от m по n. Если последний символ в строке tokens= является звездочкой, создается дополнительная переменная, значением которой будет весь оставшийся текст в строке после обработки последней подстроки;

usebackq — применение новой семантики, при которой строки, заключенные в обратные кавычки, выполняются как команды, строки, заключенные в прямые одиночные кавычки, являются строкой литералов команды, а строки, заключенные в двойные кавычки, используются для выделения имен файлов в списках имен файлов.

Поясняющий пример:

FOR /F “eol=; tokens=2,3* delims=,” %%i in (myfile.txt) do @echo %%i %%j %%k

— эта команда обрабатывает файл myfile.txt, пропускает все строки, которые начинаются с символа точки с запятой, и передает вторую и третью подстроки из каждой строки в тело цикла, причем подстроки разделяются запятыми и/или пробелами. В теле цикла переменная %i используется для второй подстроки, %j – для третьей, а %k получает все оставшиеся подстроки после третьей.

Имена файлов, содержащие пробелы, необходимо заключать в двойные кавычки.

Для того чтобы использовать двойные кавычки, необходимо использовать параметр usebackq, иначе двойные кавычки будут восприняты как границы строки для обработки.

Переменная %i явно описана в инструкции for, а переменные %j и %k описываются неявно с помощью ключа tokens=. Ключ tokens= позволяет извлечь из одной строки файла до 26 подстрок, при этом, не допускается использование переменных больших чем буквы 'z' или 'Z'. Следует помнить, что имена переменных FOR являются глобальными, поэтому одновременно не может быть активно более 52 переменных.

Синтаксис команды FOR /F также позволяет обработать отдельную строку, с указанием параметра filenameset, заключенным в одиночные кавычки.

Строка будет обработана как единая строка из входного файла.

Наконец, команда FOR /F позволяет обработать строку вывода другой команды.

Для этого следует ввести строку вызова команды в апострофах вместо набора имен файлов в скобках. Строка передается для выполнения обработчику команд CMD.EXE, а вывод этой команды записывается в память и обрабатывается так, как будто строка вывода взята из файла. Например, следующая команда:

FOR /F “usebackq delims==“ %%i IN (`set`) DO @echo %%i

— выведет перечень имен всех переменных среды, определенных в настоящее время в системе.

SHIFT — изменение содержимого (сдвиг) подставляемых параметров для пакетного файла.

SHIFT [/n] — команда SHIFT при включении расширенной обработки команд поддерживает ключ /n, задающий начало сдвига параметров с номера n, где n может быть от 0 до 9.

Например, в следующей команде:

SHIFT /2

%3 заменяется на %2, %4 на %3 и т.д., а %0 и %1 остаются без изменений.

CALL — вызов одного пакетного файла из другого.

CALL [диск:][путь]имя_файла [параметры]

параметры – набор параметров командной строки, необходимых пакетному файлу.

CHOICE[2] — ожидает ответа пользователя.

CHOICE [/C[:]варианты] [/N] [/S] [/T[:]c,nn] [текст]

/C[:]варианты — варианты ответа пользователя.
По умолчанию строка включает два варианта: YN
/N Ни сами варианты, ни знак вопроса в строке приглашения не отображаются.
/S Учитывать регистр символов.
/T[:]c,nn Ответ «c» выбирается автоматически после nn секунд ожидания текст Строка приглашения
После выполнения команды переменная ERRORLEVEL приобретает значение, равное номеру выбранного варианта ответа.

FC — сравнение двух файлов или двух наборов файлов и вывод различий между ними.

FC [/A] [/C] [/L] [/LBn] [/N] [/OFF[LINE]] [/T] [/U] [/W]
[/nnnn][диск1:][путь1]имя_файла1 [диск2:][путь2]имя_файла2
FC /B [диск1:][путь1]имя_файла1 [диск2:][путь2]имя_файла2

/A Вывод только первой и последней строк для каждой группы различий.
/B Сравнение двоичных файлов.
/C Сравнение без учета регистра символов.
/L Сравнение файлов в формате ASCII.
/LBn Максимальное число несоответствий для заданного числа строк.
/N Вывод номеров строк при сравнении текстовых файлов ASCII.
/OFF[LINE] Не пропускать файлы с установленным атрибутом «Автономный».
/T Символы табуляции не заменяются эквивалентным числом пробелов.
/U Сравнение файлов в формате UNICODE.

[2] CHOICE — это внешняя команда.

/W Пропуск пробелов и символов табуляции при сравнении.

/nnnn Число последовательных совпадающих строк, которое должно встретиться после группы несовпадающих.

[диск1:][путь1]имя_файла1

Указывает первый файл или набор файлов для сравнения.

[диск2:][путь2]имя_файла2

Указывает второй файл или набор файлов для сравнения.

FIND — поиск текстовой строки в одном или нескольких файлах.

FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] «строка» [[диск:][путь]имя_файла[...]]

/V Вывод всех строк, НЕ содержащих заданную строку.

/C Вывод только общего числа строк, содержащих заданную строку.

/N Вывод номеров отображаемых строк.

/OFF[LINE] Не пропускать файлы с установленным атрибутом «Автономный».

/I Поиск без учета регистра символов.

«строка» Искомая строка.

[диск:][путь]имя_файла

Один или несколько файлов, в которых выполняется поиск.

Если путь не задан, поиск выполняется в тексте, введенном с клавиатуры либо переданном по конвейеру другой командой.

SORT — осуществляет сортировку файла.

SORT [/R] [/+n] [/M килобайтов] [/L язык] [/REC символов]

[[диск1:][путь1]имя_файла1] [/T [диск2:][путь2]]

[/O [диск3:][путь3]имя_файла3]

/+n Задает число символов, n, до начала каждого сравнения. /+3 показывает, что каждое сравнение будет начинаться с третьего символа каждой строки. Строки меньше чем n символов собираются перед всеми остальными строками.

По умолчанию, сравнение начинается с первого символа каждой строки.

/L[OCALE] язык Перекрывает установленные в системе по умолчанию язык и раскладку заданными. Пока существует возможность только одного выбора: ««C»« – наиболее быстрый способ упорядочивания последовательности.

Сортировка всегда идет без учета регистра.

/M[EMORY] килобайтов Задает количество основной памяти, используемой для сортировки, в килобайтах. Размер памяти должен быть не менее 160КБ.

/REC[ORD_MAXIMUM] символов Определяет максимальное число символов в записи (по умолчанию 4096, максимальное 65535).

/R[EVERSE] Обратный порядок сортировки; т.е. сортировка идет от Я до А, и затем от 9 до 0.

[диск1:][путь1]имя_файла1 Определяет имя сортируемого файла. Если оно опущено, то будет использоваться стандартный поток ввода. Явное задание сортируемого файла работает быстрее, чем перенаправление того же файла в качестве стандартного потока ввода.

/T[EMPORARY] [диск2:][путь2] Определяет путь к папке, содержащей рабочие файлы сортировки, в том случае, когда данные не помещаются в основной памяти. По умолчанию используется системная временная папка.

/O[UTPUT] [диск3:][путь3]имя_файла3 Определяет имя файла, в котором сохраняются отсортированные результаты. Если оно опущено данные записываются в стандартный поток вывода. Явное задание файла вывода работает быстрее, чем перенаправление стандартного потока вывода в этот же файл.

1.4. Варианты заданий на выполнение

При разработке учтите возможность неправильного запуска ваших программ (например, с недостаточным количеством аргументов) и предусмотрите вывод сообщения об ошибке и подсказки. При выполнении работы используйте данные методические указания и справочник THelp.

Вариант 1. Разработать командный файл создающий, копирующий или удаляющий файл, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 2. Разработать командный файл создающий, копирующий или удаляющий каталог, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 3. Разработать командный файл, добавляющий вводом с клавиатуры содержимое текстового файла (в начало или в конец в зависимости от ключей (замещаемого параметра) /b /e).

Вариант 4. Разработать командный файл, регистрирующий время своего запуска в файле протокола run.log и автоматически запускающий некоторую программу (например, антивирусную и т. п.) по пятницам или 13 числам.

Вариант 5. Разработать командный файл, копирующий произвольное число файлов заданных аргументами из текущего каталога в указываемый каталог.

Вариант 6. Разработать командный файл, который помещает список файлов текущего каталога в текстовый файл и в зависимости от ключа сортирует по какому-либо полю. Реализовать два варианта: с использованием только команды DIR, с использованием команд DIR и SORT.

Вариант 7. Разработать командный файл, который в интерактивном режиме мог бы дописывать в файл текст, удалять строки из файла, и распечатывать на экране содержимое файла.

Вариант 8. Разработать командный файл, который дописывал бы имя файла, полученного входным параметром в сам файл N количество раз. N – также задается параметром.

Вариант 9. Разработать командный файл, который бы запускал бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает. В работе используйте для сравнения дат команду FC.

Вариант 10. Разработать командный файл, который бы запускал бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл

уже запускали сегодня, то ваш файл ничего не делает. Сравнение дат реализуйте через переменные, а не через файлы.

Вариант 11. Разработать командный файл, который получал в качестве параметра какое-либо имя, и проверял, определена ли токая переменная среды или нет, и выводил соответствующее сообщение.

Вариант 12. Разработать командный файл, который получал в качестве параметра какой-либо символ и в зависимости от второго параметра вырезал или сохранял в заданном файле все строки начинающиеся на этот символ.

Вариант 13. В некотором файле храниться список пользователей ПК и имя их домашних каталогов. Необходимо разработать программу, которая просматривает данный файл и в интерактивном режиме задает вопрос – копировать текущему пользователю (в его домашний каталог) какой-либо заданный файл (в качестве параметра) или нет. Если «Да» то программа копирует файл.

Вариант 14. Разработать командный файл, который бы выводил в зависимости от ключа на экран имя файла с самой последней или с самой ранней датой последнего использования.

Вариант 15. Разработать командный файл, который бы получал в качестве аргумента имя текстового файла и выводил на экран информацию о том, сколько символов, слов и строк в текстовом файле.

Вариант 16. Разработать командный файл (аналог команды tail в Unix). Командный файл печатает конец файла. По умолчанию – 10 последних строк. Явно можно задать номер строки, от которой печатать до конца.

Вариант 17. Разработать командный файл, который бы склеивал текстовые файлы, заданные в качестве аргументов, и сортировал бы строки результирующего файла в зависимости от ключа по убыванию или по возрастанию.

Вариант 18. Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы созданные, удаленные).

Вариант 19. Разработать командный файл, который формировал бы ежемесячный отчет об изменениях в рабочем каталоге (файлы измененные).

Вариант 20. Выполняющий в зависимости от ключа один из 3–х вариантов работы:

- с ключом /n дописывает в начало указанных текстовых файлов строку с именем текущего файла;
- с ключом /b создает резервные копии указанных файлов;
- с ключом /d удаляет указанные файлы после предупреждения.

ЛАБОРАТОРНАЯ РАБОТА №2 «Программирование на языке SHELL в ОС Unix»

2.1. Цель работы

Изучение языка Shell, использование переменных среды, переменных Shell и предопределенных переменных.

2.2. Программирование в языке Shell

Версии Shell

Shell — интерпретатор команд, подаваемых с терминала или из командного файла. Это обычная программа (т.е. не входит в ядро операционной системы UNIX). Ее можно заменить на другую или иметь несколько.

Две наиболее известные версии:

- Shell (версии 7 UNIX) или Bourne Shell (от фамилии автора S.R.Bourne из фирмы Bell Labs);
- C-Shell (версии Berkley UNIX).

Они похожи, но есть и отличия: C-Shell мощнее в диалоговом режиме, а обычный Shell имеет более элегантные управляющие структуры.

Shell — язык программирования, так как имеет:

- переменные;
- управляющие структуры (типа if);
- подпрограммы (в том числе командные файлы);
- передачу параметров;
- обработку прерываний.

Файл начала сеанса (login-файл)

Независимо от версии Shell при входе в систему UNIX ищет файл начала сеанса с предопределенным именем, чтобы выполнить его как командный файл;

- для UNIX версии 7 это: .profile;
- для C-Shell это: .login и/или .cshrc.

В этот файл обычно помещают команды:

- установки характеристик терминала;
- оповещения типа who, date;
- установки каталогов поиска команд (обычно: /bin, /usr/bin);
- смена подсказки с $ на другой символ и т.д.

Процедура языка Shell

Это командный файл. Два способа его вызова на выполнение:

1. $ sh dothat (где dothat — некоторый командный файл);
2. $ chmod 755 dothat (сделать его выполнимым, т.е. -rwxr-xr-x)
   $ dothat.

Следует знать порядок поиска каталогов команд (по умолчанию):

- текущий;
- системный /bin;
- системный /usr/bin.

Следовательно, если имя вашего командного файла дублирует имя команды в системных каталогах, последняя станет недоступной (если только не набирать ее полного имени).

Переменные Shell

В языке Shell версии 7 определение переменной содержит имя и значение: var = value.

Доступ к переменной — по имени со знаком $ спереди:

```
fruit = apple (определение);
echo $fruit (доступ);
apple (результат echo).
```

Таким образом, переменная — это строка. Возможна конкатенация строк:

```
$ fruit = apple
$ fruit = pine$fruit
$ echo $fruit
pineapple
$ fruite = apple
$ wine = ${fruite}jack
$ echo $wine
applejack
$
```

Другие способы установки значения переменной — ввод из файла или вывод из команды, а также присваивание значений переменной — параметру цикла for из списка значений, заданного явно или по умолчанию.

Переменная может быть:

- Частью полного имени файла: $d/filename, где $d — переменная (например, d = /usr/bin).

- Частью команды:

```
$ S = "sort + 2n + 1 - 2" (наличие пробелов требует кавычек "")
$ $S tennis/lpr
$ $S basketball/lpr
$ $S pingpong/lpr
$
```

Однако внутри значения для команды не могут быть символы |, >, <, & (обозначающие канал, перенаправления и фоновый режим).

Предопределенные переменные Shell

Некоторые из них можно только читать. Наиболее употребительные:

HOME — "домашний" каталог пользователя; служит аргументом по умолчанию для cd;

PATH — множество каталогов, в которых UNIX ищет команды;

PS1 — первичная подсказка (строка) системы (для v.7 - $).

Изменение PS1 (подсказки) обычно делается в login-файле, например:

PS1 = ?

или PS1 = "? " (с пробелом, что удобнее).

Изменение PATH:

```
$ echo $PATH                        - посмотреть;
:/bin:/usr/bin                      - значение PATH;
$ cd                                - "домой";
$ mkdir bin                         - новый каталог;
$ echo $HOME                        - посмотреть;
/users/maryann                      - текущий каталог;
$ PATH = :$HOME/bin:$PATH           - изменение PATH;
$ echo $PATH                        - посмотреть;
:/users/maryann/bin:/bin:/usr/bin   - новое значение PATH.
```

Установка переменной Shell выводом из команды

Пример 1:

```
$ now = `date` (где `` - обратные кавычки)
$ echo $now
Sun Feb 14 12:00:01 PST 1985
$
```

Пример 2: (получение значения переменной из файла):

```
$ menu = `cat food`
```

$ echo $menu

apples cheddar chardonnay (символы возврата каретки заменяются на пробелы).

Переменные Shell — аргументы процедур

Это особый тип переменных, именуемых цифрами.

Пример:

$ dothis grapes apples pears (процедура).

Тогда позиционные параметры (аргументы) этой команды доступны по именам:

$1 = `grapes`
$2 = `apples`
$3 = `pears`

и т.д. до $9. Однако есть команда shift, которая сдвигает имена на остальные аргументы, если их больше 9 (окно шириной 9).

Другой способ получить все аргументы (даже если их больше 9): $*, что эквивалентно $1$2 ...

Количество аргументов присваивается другой переменной: $#(диез).

Наконец, имя процедуры - это $0; переменная $0 не учитывается при подсчете $#.

Структурные операторы Shell

Оператор цикла **for**

Пусть имеется командный файл makelist (процедура)

```
$ cat makelist
sort +1 -2 people | tr -d -9 | pr -h Distribution | lpr.
```

Если вместо одного файла people имеется несколько, например: adminpeople, hardpeople, softpeople,..., то необходимо повторить выполнение процедуры с различными файлами. Это возможно с помощью for — оператора. Синтаксис:

```
for <переменная> in <список значений>
do <список команд>
done
```

Ключевые слова for, do, done пишутся с начала строки.

Пример (изменим процедуру makelist):

```
for file in adminpeople, hardpeople, softpeople
do
        Sort +1 -2 $file | tr ... | lpr
done.
```

Можно использовать метасимволы Shell в списке значений.

Пример:

```
for file in *people (для всех имен, кончающихся на people)
do
        ...
done.
```

Если in опущено, то по умолчанию в качестве списка значений берется список аргументов процедуры, в которой содержится цикл, а если цикл не в процедуре, то — список параметров командной строки (то есть в качестве процедуры выступает команда).

Пример:

```
for file
  do
        ...
  done
```

Для вызова makelist adminpeople hardpeople softpeople будет сделано то же самое.

Условный оператор **if**

Используем имена переменных, представляющие значения параметров процедуры:

```
sort +1 -2 $1 | tr ... | lpr
```

Пример неверного вызова:

makelist (без параметров), где $1 неопределен. Исправить ошибку можно, проверяя количество аргументов – значение переменной $# посредством if - оператора.

Пример: (измененной процедуры makelist):

```
if test $# -eq 0
then
        echo "You must give a filename"
        exit 1
else
        sort +1 -2 $1 | tr ... | lpr
fi
```

Здесь test и exit - команды проверки и выхода. Таким образом, синтаксис оператора if:

if <если эта команда выполняется успешно, то>;

then <выполнить все следующие команды до else или, если его нет, до fi>;

[else <иначе выполнить следующие команды до fi>]

Ключевые слова if, then, else и fi пишутся с начала строки.

Успешное выполнение процедуры означает, что она возвращает значение true = 0 (zero) (неуспех - возвращаемое значение не равно 0).

Оператор exit 1 задает возвращаемое значение 1 для неудачного выполнения makelist и завершает процедуру.

Возможны вложенные if. Для else if есть сокращение elif, которое одновременно сокращает fi.

Команда **test**

Не является частью Shell, но применяется внутри Shell-процедур.

Имеется три типа проверок:

- оценка числовых значений;
- оценка типа файла;
- оценка строк.

Для каждого типа свои примитивы (операции op).

1. Для чисел синтаксис такой: N op M, где N, M - числа или числовые переменные;

op принимает значения: -eq, -ne, gt, -lt, -ge, -le.

2. Для файла синтаксис такой: op filename, где op принимает значения:

- -s (файл существует и не пуст);
- -f (файл, а не каталог);
- -d (файл-директория (каталог);
- -w (файл для записи);
- -r (файл для чтения).

Для строк синтаксис такой: S op R, где S, R - строки или строковые переменные или op1 S, где op принимает значения:

- = (эквивалентность);
- != (не эквивалентность);

op1 принимает значения:

- -z (строка нулевой длины);
- -n (не нулевая длина строки).

Наконец, несколько проверок разных типов могут быть объединены логическими операциями -a (AND) и -o (OR).

Примеры:

```
$ if test -w $2 -a -r S1
> then cat $1 >> $2
> else echo "cannot append"
> fi
$
```

В некоторых вариантах ОС UNIX вместо команды test используются квадратные скобки, т.е. if [...] вместо if test ... .

Оператор цикла **while**

Синтаксис:

```
while <команда>
do
<команды>
done
```

Если "команда" выполняется успешно, то выполнить "команды", завершаемые ключевым словом done.

Пример:

```
if test $# -eq 0
then
        echo "Usage: $0 file ..." > &2
        exit
fi
while test $# -gt 0
do
if test -s $1
then
        echo "no file $1" > &2
else
        sort + 1 - 2 $1 | tr -d ... (процедуры)
fi
        shift (* перенумеровать аргументы *)
done
```

Процедуры выполняются над всеми аргументами.

Оператор цикла **until**

Инвертирует условие повторения по сравнению с while

Синтаксис:

```
until <команда>
do
<команды>
done
```

Пока "команда" не выполнится успешно, выполнять команды, завершаемые словом done.

Пример:

```
if test S# -eq 0
then
        echo "Usage $0 file..." > &2
        exit
fi
until test S# -eq 0
do
if test -s $1
then
        echo "no file $1" > &2
else
        sort +1 -2 $1 | tr -d ... (процедура)
fi
shift (сдвиг аргументов)
done
```

Исполняется аналогично предыдущему.

Оператор выбора **case**

Синтаксис:

```
case <string> in
string1) <если string = string1, то выполнить все следующие команды до ;; > ;;
string2) <если string = string2, то выполнить все следующие команды до ;; > ;;
string3) ... и т.д. ...
esac
```

Пример:

Пусть процедура имеет опцию -t, которая может быть подана как первый параметр:

```
.....
```

```
together = no
case $1 in
-t)     together = yes
        shift ;;
-?)     echo "$0: no option $1"
        exit ;;
esac
if test $together = yes
then
        sort ...
fi
```

где ? - метасимвол (если -?, т.е. "другая" опция, отличная от -t, то ошибка). Можно употреблять все метасимволы языка Shell, включая ?, *, [-]. Легко добавить (в примере) другие опции, просто расширяя case.

Использование временных файлов в каталоге /tmp

Это специальный каталог, в котором все файлы доступны на запись всем пользователям.

Если некоторая процедура, создающая временный файл, используется несколькими пользователями, то необходимо обеспечить уникальность имен создаваемых файлов. Стандартный прием – имя временного файла $0$$, где $0 - имя процедуры, а $$ - стандартная переменная, равная уникальному идентификационному номеру процесса, выполняющего текущую команду.

Хотя администратор периодически удаляет временные файлы в /tmp, хорошей практикой является их явное удаление после использования.

Комментарии в процедурах

Они начинаются с двоеточия : , которое считается нуль-командой, а текст комментария - ее аргументом. Чтобы Shell не интерпретировал метасимволы ($, * и т.д.), рекомендуется заключать текст комментария в одиночные кавычки.

В некоторых вариантах ОС UNIX примечание начинается со знака #.

Пример процедуры

```
:'Эта процедура работает с файлами, содержащими имена'
: 'и номера телефонов,'
```

```
:'сортирует их вместе или порознь и печатает результат на'
:'экране или на принтере'
:'Ключи процедуры:'
:'-t (together) - слить и сортировать все файлы вместе'
:'-p (printer) - печатать файлы на принтере'
if test $# - eq 0
then
        echo "Usage: $ 0 file ... " > & 2
        exit
fi
together = no
print = no
while test $# -gt 0
do case $1 in
-t)     together = yes
        shift ;;
-p)     print = yes
        shift ;;
-?)     echo "$0: no option $1"
        exit ;;
*) if test $together = yes
then
        sort -u +1 -2 $1 | tr ... > /tmp/$0$$
if $print = no
then
        cat /tmp/$0$$
else
        lpr -c /tmp/$0$$
fi
        rm /tmp/$0$$
        exit
else if test -s $1
then    echo "no file $1" > &2
else    sort +1 -2 $1 | tr...> /tmp/$0$$
if $print = no
then    cat /tmp/$0$$
else    lpr -c /tmp/$0$$
fi
        rm /tmp/$0$$
fi
shift
```

```
  fi;;
  esac
  done.
```

Процедура проверяет число параметров $# и, если оно равно нулю, завершается. В противном случае она обрабатывает параметры (оператор case). В качестве параметра может выступать либо ключ (символ, предваряемый минусом), либо имя файла (строка, представленная метасимволом *). Если ключ отличен от допустимого (метасимвол ? отличен от t и p), процедура завершается. Иначе в зависимости от наличия ключей t и p выполняются действия, заявленные в комментарии в начале процедуры.

Обработка прерываний в процедурах

Если при выполнении процедуры получен сигнал прерывания (от клавиши BREAK или DEL, например), то все созданные временные файлы останутся неудаленными (пока это не сделает администратор) ввиду немедленного прекращения процесса.

Лучшим решением является обработка прерываний внутри процедуры оператором trap:

Синтаксис:

```
  trap 'command arguments' signals...
```

Кавычки формируют первый аргумент из нескольких команд, разделенных точкой с запятой. Они будут выполнены, если возникнет прерывание, указанное аргументами signals (целые):

2 - когда вы прерываете процесс;

1 - если вы "зависли" (отключены от системы)

и др.

Пример (развитие предыдущего):

```
  case $1 in
  .....
  *) trap 'rm /tmp/*; exit' 2 1 (удаление временных файлов)
  if test -s $1
  ..............
  rm /tmp/*
```

Лучше было бы:

```
  trap 'rm /tmp/* > /dev/null; exit' 2 1
```

так как прерывание может случиться до того, как файл /tmp/$0$$ создан и аварийное сообщение об этом случае перенаправляется на null-устройство.

Выполнение арифметических операций: expr

Команда expr вычисляет значение выражения, поданного в качестве аргумента, и посылает результат на стандартный вывод. Наиболее интересным применением является выполнение операций над переменными языка Shell.

Пример суммирования 3 чисел:

```
$ cat sum3
expr $1 + $2 + $3
$ chmod 755 sum3
$ sum3 13 49 2
64
$
```

Пример непосредственного использования команды:

```
$ expr 13 + 49 + 2 + 64 + 1
129
$
```

В expr можно применять следующие арифметические операторы: +, -, *, /, % (остаток). Все операнды и операции должны быть разделены пробелами.

Заметим, что знак умножения следует заключать в кавычки (одинарные или двойные), например: '*', так как символ * имеет в Shell специальный смысл.

Более сложный пример expr в процедуре (фрагмент):

```
num = 'wc -l < $1'
tot = 100
count = $num
avint = 'expr $tot / $num'
avdec = 'expr $tot % $num'
while test $count -gt 0
do ...
```

Здесь wc -l осуществляет подсчет числа строк в файле, а далее это число используется в выражениях.

Отладка процедур Shell

Имеются три средства, позволяющие вести отладку процедур.

- Размещение в теле процедуры команд echo для выдачи сообщений, являющихся трассой выполнения процедуры.
- Опция -v (verbose = многословный) в команде Shell приводит к печати команды на экране перед ее выполнением.

- Опция -x (execute) в команде Shell приводит к печати команды на экране по мере ее выполнения с заменой всех переменных их значениями; это наиболее мощное средство.

Утилита AWK

Awk - утилита, подобная grep. Однако, кроме поиска по образцу, она позволяет проверять отношения между полями строк (записей) и выполнять некоторые действия над строками (генерировать отчеты). Название не является акронимом, оно образовано первыми буквами фамилий авторов (A.V.Aho, P.Y.Weinberger, B.W.Kernighan).

Задание поиска-действия следует синтаксису:
/<образец>/{<действие>}

И образец, и действие могут отсутствовать. Найденные по образцу строки при отсутствии заданного действия выводятся в стандартный вывод (на экран).

Образец задается регулярным выражением, как и в grep. Если образец отсутствует, обрабатываются все строки.

Рассмотрим примеры действий, которые можно выполнить командой awk.

Перестановка полей строки выполняется с помощью ссылки на поле $n, где n - номер поля.

Например:

```
$ cat people
Mary Clark 101
Henry Morgan 112
Bill Williams 100
$ awk '{print $2 "," $1 "^I" $3}' people
Clark, Mary 101
Morgan, Henry 112
Williams, Bill 100
```

где ^(control - I) - знак табуляции для подвода каретки к очередной позиции табуляции (для выравнивания третьего поля).

Действия для awk могут быть заданы в файле.

Например:

```
$ cat swap
{print $2 "," $1 "^I" $3}
$ awk -f swap people
```

Awk имеет встроенные образцы и переменные. Образцы BEGIN и END означают начало и конец файла соответственно. Переменная NR (Number of Records) означает число записей (строк) в файле, NF - число полей в записи. Можно использовать переменные, объявленные пользователем. Пример, подсчитывающий среднее значение третьего поля файла tennis (программа действий для awk - в файле average):

```
$ cat > average
{total = total + $3}
END {print "Average value is", total/NR}
^D
$ awk -f average tennis
Average value is 8.9
$
```

Образец поиска в awk может содержать условные выражения. Пример, в котором в файле tennis пишутся все записи, значение третьего поля в которых не меньше 10:

```
$ awk '$3 >= 10 {print $0}'tennis
Steve Daniel 11
Hank Parker 18
Jack Austen 14
$
```

Знак $0 (доллар-ноль) есть ссылка на всю запись (строку). В общем случае выражение для условия подчиняется синтаксису, близкому к синтаксису выражений в языке C. Кроме того, в команде awk допустимо указывать отрезок образцов. Пример выборки всех записей, сделанных с 1976 до 1978 г.:

```
$ sort -n -o chard.s chard
$ awk '/1976/, /1978/ {if($2 < 8.00 print $0}' chard.s
1976 7.50 Chateau
1977 7.75 Chateau
1978 5.99 Charles
```

Как видно из примера, в программах действий для awk можно использовать управляющие структуры с синтаксисом, близким к языку C.

Пример цикла для печати полей всех записей файла в обратном порядке:

$ awk {for (i = NF; i > 0; --i) print $i} f1, где NF - число полей в записи.

Встроенные функции AWK

length(arg) - Функция длины arg. Если arg не указан, то выдает длину текущей строки.

exp(),log(),sqrt() - Математические функции экспонента, логарифм и квадратный корень.

int() - Функция целой части числа.

substr(s,m,n) - Возвращает подстроку строки s, начиная с позиции m, всего n символов.

index(s,t) - Возвращает начальную позицию подстроки t в строке s. (Или 0, если t в s не содержится.)

sprintf(fmt,exp1,exp2,...) - Осуществляет форматированную печать (вывод) в строку, идентично PRINTF.

split(s,array,sep) - Помещает поля строки s в массив array и возвращает число заполненных элементов массива. Если указан sep, то при анализе строки он понимается как разделитель.

Операции отношения awk

X = = Y – X равно Y?
X ! = Y – X не равно Y?
X > Y – X больше чем Y?
X = > Y X больше чем или равно Y?
X < Y – X меньше чем Y?
X < = Y – X меньше чем или равно Y?
X ~ Re – X совпадает с регулярным выражением Re?
X ! ~ Re – X не совпадает с регулярным выражением Re?

Старшинство операций в awk

Группа	Операции
1	= += -= *= /= %=
2	||
3	&&
4	> >= < <= == != ~ !~
5	Строка конкатенации «x» «y» становится «xy»
6	+ -
7	* / %
8	++ --

Стандартные переменные

ARGC – число аргументов в командной строке;
ARGV – массив с аргументами командной строки;
FILENAME – строка текущего файла ввода;
FNR – номер текущей записи в текущем файле;
FS – разделитель полей ввода;
NF – число полей в текущей записи;
NR – номер текущей записи;
OFMT – формат вывода чисел (по умолчанию % 6g);
OFS – разделитель полей ввода (по умолчанию пробел);
ORS – Разделитель выводимых записей (по умолчанию новая строка);
RS – Разделитель полей ввода (по умолчанию новая строка).

2.3. Список команд Shell

date — вывод даты;
who — вывод пользователей;
who am i— вывод собственного имени;
exit — выход из системы (для передачи кода завершения);
mail — почта;
write — передача сообщения другому пользователю;
man — информация о команде;
news — новости;
ed — текстовый редактор (a/... ./w имя/ctrl-d)
ls — перечень имен файлов в каталоге;
ls -t — перечень файлов во временном порядке;
ls -l — перечень файлов в полном виде;
ls -li — перечень файлов в расширенном виде;
cat — распечатка файла (cat>имя — создание файла);
pr — распечатка по 66 строк;
mv — перенос файла;
cp — копирование файла;
rm — удаление файла;
ln — назначение связи;
rmdir — удалить каталог;
mkdir — создать каталог;
pwd — определение своего рабочего каталога;
cd — смена каталога;
wc — подсчет числа строк, слов и символов;
tail +n — вывод файла начиная со строки с номером n;

cmp — поиск различий между файлами (до первого различия);
diff — поиск всех различий;
echo — вывод строки (` ` — результата, ' ' — команды);
echo $? — выдача кода завершения команды (0, 1, 2);
wait — ждать завершения всех процессов;
kill — убить процесс (kill -9 #_процесса);
ps — список процессов;
nohup — выполнение команды после отключения (nohup кмд&);
nice — запуск с пониженным приоритетом (nice кмд&);
at — запуск в определенное время (at команды ctrl-d);
export — сообщение интерпретатору о использовании переменных;
sh — переход в порожденный shell;
du — определение занятого пространства;
df — свободное пространство диска;
chmod — смена права доступа;
mesg — (n — запрет, y — разрешение) сообщения;
sleep — пауза;
set — показать все ранее определенные переменные;
set `` — установить значение переменной;
time — информация о времени выполнения команды;
uname — информация о системе (uname -a — полная);
read — присваивает переменной значение последующей строки;
touch — заменяет время модификации файла на настоящее;
for — цикл (for i in список/ do команды/ done);
case — выбор (case слово in/шаблон) команды ;;/esac);
if — условие (if команда / then команды, если условие верно / else команды, если условие ложно/fi;
while — цикл (while команда/do тело цикла, выполняется пока команда возвращает истина/done;
until — цикл (аналог while, но ждет ложь);
trap — последовательность действий выполняемая при прерывании (trap 'rm -f $old; exit 1' 1 2 15), где

0 — выход из интерпретатора
1 — отбой
2 — прерывание (DEL)
3 — останов (ctrl-\; вызывает распечатку содержимого памяти программы)
9 — уничтожение
15 — окончание выполнения.

Встроенные переменные интерпретатора

$# — число аргументов;
$* — все аргументы, передаваемые интерпретатору ($@);
$- — флаги передаваемые интерпретатору;
$? — возвращение значения последней выполненной команды;
$$ — номер процесса интерпретатора;
$! — номер процесса последней команды, запущенной с &;

Правила сопоставления шаблонов в интерпретаторе

* — задание любой строки, в том числе и пустой.
? — любой одиночный символ;
"..." — задает в точности ...; ""(") защищает от спецсимволов;
\c — задает с буквально;
a|b — только для выражения выбора, a или b.

Значения переменных

$var — значение var;
${var-thing} — значение var, если оно определено, в противном случае thing;
${var=thing} — значение var, если var не определено, то присваивается значение thing;
${var?строка} — если var определено — $var, в противном случае выводится строка и инт. прекращает работу;
${var+thing} — thing, если $var определено, в противном случае ничего.

Метасимволы

| — конвейер (связь выходного потока одной программы с выходным потоком другой);
& — асинхронный запуск;
; — последовательное выполнение;
> — помещение выходного потока;
>> — добавление выходного потока;
* — любая строка;
? — любой символ;
[ссс] — задает любой символ из [ссс] в имени файла;
`...` — инициирует выполнение команды;

( ) — инициирует выполнение команды в порожденном shell;
{ } — инициирует выполнение команды в текущем shell;
$1 — заменяется аргументом командного файла;
$var — значение переменной var в программе на языке shell;
${var} — значение var;
\ — перевод строки;
'...' — непосредственное использование;
"..." — непосредственное использование, после того, как $`...` и \ будут интерпретированы;
— остальная строка — комментарий;
p1&&p2 — выполнить p1, в случае успеха p2;
p1||p2 — выполнить p1, в случае неудачи p2;
2>file — переключить поток диагностики на файл;
2>&1 — поместить стандартный поток диагностики в выходной поток;
1>&2 — добавление выходного потока к стандартному потоку диагностики.

2.4. Варианты заданий на выполнение

Вариант 1. Разработать программу, отправляющую почту (содержимое файла) группе пользователей, выбираемых из общего списка (хранящегося в другом файле) в интерактивном режиме. Например, вы отвечаете "Y" для тех, кому надо посылать, "N" — не надо, "Q" — конец выбора.

Вариант 2. Разработать программу, выводящую через определенный интервал времени информацию о пользователях в системе: кто вошел, кто вышел.

Вариант 3. Разработать программу, выполняющую в зависимости от ключа один из 3–х вариантов работы:
- с ключом /n дописывает в начало указанных текстовых файлов строку с именем текущего файла;
- с ключом /b создает резервные копии указанных файлов;
- с ключом /d удаляет указанные файлы после предупреждения.

Вариант 4. Разработать программу создающую, копирующую или удаляющую файл, указанный в командной строке, в зависимости от выбранного ключа (замещаемого параметра) /n , /c , /d.

Вариант 5. Разработать программу, добавляющую вводом с клавиатуры содержимое текстового файла (в начало или в конец в зависимости от ключей (замещаемого параметра) /b /e).

Вариант 6. Разработать программу, регистрирующую время своего запуска в файле протокола run.log и автоматически запускающую некоторую программу (например, антивирусную и т. п.) по пятницам или 13 числам.

Вариант 7. Разработать программу, копирующую произвольное число файлов заданных аргументами из текущего каталога в указываемый каталог.

Вариант 8. Разработать программу, которая в интерактивном режиме могла бы дописывать в файл текст, удалять строки из файла, и распечатывать на экране содержимое файла.

Вариант 9. Разработать программу, которая бы запускала бы какой-либо файл один раз в сутки. То есть, если файл запускается первый раз в сутки, то он запускает какой-либо файл. Если ваш файл уже запускали сегодня, то ваш файл ничего не делает.

Вариант 10. Разработать программу, которая получала бы в качестве параметра какой-либо символ и в зависимости от второго параметра вырезала или сохраняла в заданном файле все строки начинающиеся на этот символ.

Вариант 11. В некотором файле храниться список пользователей ПК и имя их домашних каталогов. Необходимо разработать программу, которая просматривает данный файл и в интерактивном режиме задает вопрос – копировать текущему пользователю (в его домашний каталог) какой-либо заданный файл (в качестве параметра) или нет. Если «Да» то программа копирует файл.

Вариант 12. Разработать программу, которая бы выводил в зависимости от ключа на экран имя файла с самой последней или с самой ранней датой последнего использования.

Вариант 13. Разработать программу (аналог команды wc), которая бы получала бы в качестве аргумента имя текстового файла и вы-

водила на экран информацию о том, сколько символов, слов и строк в текстовом файле.

Вариант 14. Разработать программу (аналог команды tail), которая печатает конец файла. По умолчанию – 10 последних строк. Явно можно задать номер строки, от которой печатать до конца.

Вариант 15. Разработать программу, которая склеивала бы текстовые файлы, заданные в качестве аргументов, и сортировала бы строки результирующего файла в зависимости от ключа по убыванию или по возрастанию.

Вариант 16. Разработать программу, которая формировала бы ежемесячный отчет об изменениях в рабочем каталоге (файлы созданные, удаленные).

Вариант 17. Разработать программу, разбирающую содержимое письма (файл или входной поток), выделяющую заголовок письма с адресом отправителя (поля From: или From) и отправляющую содержимое письма без заголовка обратно отправителю.

Вариант 18. Разработать программу, которая изменяет текстовый файл так, что четные и нечетные строки меняются местами.

Вариант 19. Разработать программу, которая бы в зависимости от параметров, строила бы выборку по какому бы условию (числовые значения) из табличного файла.

Вариант 20. Разработать программу, которая инвертирует текстовый файл или его строки.

ЛАБОРАТОРНАЯ РАБОТА №3 «Исследование структур основной памяти – conventional memory»

3.1. Цель работы

Изучить структуру системных таблиц реального режима Windows и организацию цепочек блоков памяти.

3.2. Организация хранения байтов в памяти

При просмотре памяти имейте в виду, что двухбайтовые слова хранятся в виде {младший байт}{старший байт} – т.е. порядке обратном естественному представлению многоразрядного числа.

То же самое относится к порядку расположения слов в двойном слове – сначала младшее слово, потом старшее. Всегда действует общий принцип – младшее лежит в ячейке памяти с младшим адресом. Таким образом, полный 4-х байтный указатель (например, на таблицу таблиц) 1234:5678H будет в дампе памяти выглядеть как:

```
78 56 34 12
 \/    \/
  |     |___ старшее слово с переставленными байтами
  |
  |_________ младшее слово с переставленными байтами
```

3.3. Информация о структурах памяти

Это список указателей, каждый из которых представляет собой двойное слово (4 байта). Старшее слово – это сегментный адрес, младшее – смещение в сегменте. Например, для указателя, у которого сегментный адрес=1234H, а смещение 5678H, абсолютный физический адрес ячейки памяти образуется, как сумма смещения и сегментного адреса * 16 (т.е. сегментный адрес сдвинут влево на 1 шестнадцатиричный разряд):

```
 1234 H 0110 H 0112 H
+ 5678H + 0026H + 0006H
-------- --------- ---------
=179B8H =01126H =01126H
```

Таким образом 0110:0026 – это тоже, что и 0112:0006 !

Структура таблицы таблиц

Данная структура является НЕДОКУМЕНТИРОВАННОЙ и используется для изучения низкоуровневой информации о структурах памяти.

Смещение	Длина	Содержимое
-2	2	сегм. адр. 1 MCB
0	4	указ. на 1 DPB
+ 4	4	указ. на список таблиц открытых файлов
+ 8	4	указ. на первый драйвер DOS (CLOCK$)
…	…	…

Структура блока управления памятью (MCB)

MCB – Это НЕДОКУМЕНТИРОВАННЫЙ управляющий блок, который используется при распределении, модификации и освобождении блоков системной памяти.

Смещение	Длина	Содержимое
+0	1	'M'(4dH) – за этим блоком есть еще блоки 'Z'(5aH) – данный блок является последним
+1	2	Владелец параграф владельца (для FreeMem); 0 = владеет собой
+3	2	Размер число параграфов в этом блоке распределения
+5	0Bh	Зарезервировано
+10h	?	Блок памяти начинается здесь и имеет длину (Размер*10H) байт

Замечания:

- блоки памяти всегда выровнены на границу параграфа («сегмент блока»);
- блоки M-типа: следующий блок находится по (сегмент блока + Размер):0000;
- блоки Z-типа: (сегмент блока + Размер):0000 = конец памяти (a000H=640K).

В любом MCB указан его владелец – сегментный адрес PSP программы владельца данного блока памяти. А в PSP есть ссылка на

окружение данной программы, в котором можно найти имя программы – путь ее запуска.

Сленить, что грамма (и числе) и ее сами распоблоках паэтому, в памяти саграммы в хозяина указственный себя.

дует помсама проPSP в том окружение лагаются в мяти. ПоМСВ блока мой прокачестве зан собадрес самого

Когда реальном начинает DS:0000 и зывают на этой про-

программа в режиме выполнение, ES:0000 указначало PSP граммы. Информация PSP позволяет выделить имена файлов и опции из строки команд, узнать объем доступной памяти, определить окружение и т.д.

Использование окружения. Окружение не превышает 32K байт и начинается на границе параграфа. Смещение 2cH в PSP текущей программы содержит номер параграфа окружения.

Вы можете найти нужное 'имя' серией сравнений строк ASCIIZ (Строка ASCIIZ, используемая во многих функциях DOS и в языке C, представляет собой последовательность символов ASCII, заканчивающуюся байтом 00H), пока не дойдете до пустой строки (нулевой длины), что указывает конец окружения. Обычно 'имя' в каждой строке окружения задано прописными буквами, но это необязательно.

Одна типичная операция с окружением используется программами типа оболочки, которые запускают вторичную копию COMMAND.COM. Такие программы обычно ищут 'имя' «COMSPEC» и используют соответствующее 'значение', как полный путь интерпретатора команд DOS ── программы, запускаемой через функцию DOS 4bH .

Некоторые программы требуют, чтобы оператор поместил информацию для приложения в окружение посредством команды SET.

Приложение может использовать такую информацию при каждом выполнении. Например, текстовый процессор может отыскивать в окружении 'имя' «DICTIONARY» и использовать соответствующее 'значение' как имя файла со словарными данными.

Более подробную информацию о структурах памяти можно получить из справочника TECH Help!

3.4. Задание на выполнение

1. Подготовиться к работе, используя материалы лекций, данное пособие, справочник TEACH-HELP.

2. Познакомиться с работой одной из программ, позволяющих просмотреть содержимое ОЗУ в виде шестнадцатиричного дампа – например, PEEK.COM (во время работы доступен HELP – F1, карта памяти – F8 и информация о блоке памяти – F6).

3. Найти в памяти таблицу таблиц (для получения ее адреса – запусти lol.com), познакомиться с ее содержимым и посмотреть указатель на 1 MCB (упр. блок памяти).

4. Проследить в памяти цепочку блоков, определяя их принадлежность и сравнивая с информацией из карты памяти (F8).

5. Написать отчет о найденной цепочке блоков памяти с их адресами и размерами (в файле MEMBLOCK.TXT)

ЛАБОРАТОРНАЯ РАБОТА №4 «Изучение файловой системы FAT»

4.1. Цель работы

Знакомство со структурой системной области диска и принципами организации файловой системы FAT.

4.2. Логическая структура дисков

Диски (точнее, дискеты и жесткие диски) – это внешние запоминающие устройства, которому операционная система обеспечивает стандартную поддержку. При этом операционная система задает логическую структуру носителя, которым может быть дискета или раздел жесткого диска.

Эта структура создается командой FORMAT и превращает носитель в последовательность логических секторов, нумеруемых с нуля. Размер каждого сектора – 512 байтов.

Любой диск, отформатированный в FAT, состоит из:

- загрузочного сектора (логический сектор 0);
- нескольких (обычно двух) копий таблицы размещения файлов (FAT);
- корневого каталога;
- области данных, где размещаются подкаталоги и файлы.

Загрузочный сектор. Загрузочный сектор – это логический сектор диска с номером 0. Он содержит программу начальной загрузки ОС с данного диска (если диск системный) или сообщение об отсутствии системы на диске, если ее там нет. Начальная область загрузочного сектора содержит, кроме того, блок информации о диске.

Смещение	Длина	Содержимое	Комментарий
+0	3	JMP ****	Переход на начало загрузчика
+3	8	OEM	Название фирмы и версия ОС
+0Bh	2	SectSize	Количество байтов в секторе
+0Dh	1	ClustSize	Количество секторов в кластере
+0Eh	2	ResSecs	Количество резервных секторов
+10h	1	FATCnt	Количество FAT
+11h	2	RootSize	Макс. число элементов корневого каталога

+13h	2	TotSecs	Общее число секторов на носителе
+15h	1	Media	Описатель носителя
+16h	2	FATSize	Количество секторов в одной FAT
+18h	2	TrkSecs	Количество секторов на дорожке
+1Ah	2	HeadCnt	Количество головок (поверхностей диска)
+1Bh	2	HidnSec	Количество скрытых секторов (может быть использовано при разбиении диска на разделы)
1Eh		Размер заголовка	

Примечания:

1. Поле OEM может содержать любые 8 байтов. Оно не используется ОС.
2. Часть загрузочного сектора со смещениями с 0Bh по 15h называется блоком параметров BIOS (BPB). BPB – это таблица параметров диска, которая используется драйверами дисков.
3. Резервные сектора располагаются между загрузочным сектором и первой FAT.
4. Описатель носителя совпадает с первым байтом FAT.
5. Для чтения загрузочного сектора можно использовать Int 25h с DX=0 или функции BIOS, точнее:
   - для дискет: читать через Int 13h головку 0, дорожку 0, сектор 1;
   - для жестких дисков: читать таблицу разделов для получения головки, дорожки и сектора начала раздела диска.
6. Информация, хранящаяся в загрузочном секторе, позволяет вычислить номер логического сектора по номеру кластера, его содержащего. Следующие формулы описывают единственный документированный метод такого расчета.

а) Вычислить размер корневого каталога в секторах, учитывая, что один элемент каталога состоит из 32 байтов:

```
RootSectors = (RootSize * 32) / SectSize;
```

б) Вычислить количество секторов, отведенных под FAT:

```
FATSectors = (FATSize * FATCnt);
```

в) Вычислить номер первого сектора области данных:

DataStart = ResSecs + FATSectors + RootSectors;

г) Вычислить логический номер первого сектора в данном кластере:

Sector = DataStart + ((Cluster-2) * ClustSize).

Системные файлы. В дополнение команда FORMAT формирует таблицу размещения файлов (FAT) и директорий диска. Обе эти структуры тесно связаны с организацией доступа к файлам. На каждом диске имеется две копии FAT. Эта таблица имеет исключительное значение при обслуживании файлов, поэтому в случае потери первой копии FAT, система получает доступ ко второй.

Если в командной строке FORMAT указан параметр /s, то на форматируемый диск записываются копии системных файлов. В DOS имеется три системных файла — IO.SYS, MSDOS.SYS и COMMAND.COM.

Системные файлы хранятся на диске, с которого загружается операционная система. Файлы записываются в строго определенном порядке и имеют строго определенное месторасположение.

Структура директория. Директорий – это таблица-описание содержимого диска. Каждому файлу в таблице соответствует одна запись. Запись занимает 32 байта, разбитых на 8 участков или полей. В каждое поле записывается информация, используемая системой при обслуживании файла. В таблице приводится краткое описание содержимого каждого поля.

Название поля	***Короткий адрес***	***Описание***
Имя файла	0-7	Имя файла. Если имя состоит менее, чем из 8 символов, то оно дополняется пробелами. Если в первом байте поля находится одно из следующих значений, это означает: - 00H — Данная запись не заполнена и ранее не заполнялась данными. Если система обнаруживает такую запись, то она целиком пропускается, что сокращает

		время поиска требуемого файла. - E5H Запись относится к уничтоженному файлу. При уничтожении файл физически остается на диске, а в соответствующей ему записи директория в поле «имя файла» изменяется значение первого байта. Программы восстановления «стертых» файлов основаны именно на этой особенности. - 2EH Запись относится к директорию или поддиректорию. Если второй байт этого поля имеет то же значение, то в 26 и 27 байтах данной записи записан номер первого кластера родительского директория. Родительский директорий является корневым, если эти байты равны 00H. (Строение кластеров рассматриваются ниже).
Расширение	8-10	Расширение имени файла. Если расширение занимает менее трех байтов, то оно дополняется пробелами.
Атрибуты	11	Атрибуты файла. Атрибуты определяются комбинацией битов 11-го байта записи. Система считает, что файлу присвоен данный атрибут, если соответствующий бит равен 1. Если он равен 0, принимается, что данный атрибут не установлен. Бит Атрибут файла (если значение бита равно 1) 0 «Только чтение». Файл можно только читать. При любой попытке записи в файл, генерируется сообщение об ошибке. 1 «Скрытый файл». При введении команды DIR имя файла с этим атрибутом не выводится на экран. 2 «Системный файл». Системные файлы используются при загрузке. 3 «Метка тома». Переменные в поле «имя файла» и «расширение» составляют метку данного диска. Все остальные поля

		этой записи не просматриваются. Запись должна входить в корневой директорий и быть на диске единственной. 4 «Поддиректорий». Запись относится к поддиректорию. 5 «Архив». Означает, что данный файл был обработан, а не откопирован командой BACKUP. При использовании BACKUP для копирования файла, архивный бит очищается.
Для нужд системы	12-21	Резервируются для системного пользования.
Метка времени	22-23	«Маркер времени». Содержит время создания файла или последней его корректировки. Нумерация битов начинается с первого бита 22-го байта. Байт 22 включает биты 0-7, байт 23 — биты 8-15. В битах 11-15 записывается значение часа (0-23), в битах 5-10 — значение минут (0-59), в битах 0-4 — значение секунд (0-59). Числа хранятся в двоичном представлении.
Метка даты	24-25	«Маркер даты». Содержит дату создания файла или последней его корректировки. Нумерация битов начинается с первого бита 24-го байта. Байт 24 включает биты 0-7, байт 25 — 8-15. Биты 9-15 используются для хранения значения года (1980=0), биты 5-8 — для хранения значения месяца (1-12), биты 0-4 — значения дня месяца (1-31). Числа хранятся в двоичном представлении.
Начальный кластер	26-27	«Начальный кластер». По значению этого поля определяется начальный адрес файла на диске.
Размер файла	28-31	«Размер файла». В первом слове (байты 28 и 29) хранится последняя часть значения, во втором (байты 30 и 31) — первая. В обоих случаях крайний значащий байт — первый.

Таблица размещения файлов. Директорий — это таблица-описание содержимого диска. Таблица размещения файлов представляет собой карту с адресами файлов на диске. Каждому файлу в таблице соответствует группа записей, описывающих его физическое расположение на диске. Операционная система разбивает файл на кластеры. Размер кластера на жестком диске зависит от количества логических дисков, на которые он был разбит при форматировании.

Кластерам присваиваются номера в соответствии с их физическим расположением на диске. Первый кластер расположен непосредственно за директорием, второй — непосредственно за первым кластером и т.д.

На односторонних дискетах номера кластеров возрастают в порядке возрастания номеров сегментов. Кластеры последовательно заполняют сектора на дорожке диска. За кластером, занимающим последний сегмент на дорожке, следует кластер, занимающий первый сегмент следующей дорожки. При этом номер кластера увеличивается на 1. На двухсторонней дискете нумерация производится следующим образом. На 0-ой стороне номера кластера возрастает в порядке возрастания номеров сектора. Затем кластеры начинают заполнять дорожку с тем же номером на стороне 1, и опять нумерация кластеров возрастает с ростом номеров сегментов. Заполнив дорожку на стороне 1, кластеры продолжают заполнять следующую дорожку с первого сегмента на стороне 0. Нумерация кластеров возрастает непрерывно.

При вычислении номера кластера по таблице размещения файлов первым шагом системы является получение номера первого кластера файла. Этот номер находится в 26-27 байтах соответствующей записи директория. Затем вычисляется номер второго кластера. Для этого значение первого номера умножается на 1,5 и целая часть полученного произведения рассматривается как адрес слова в таблице размещения файлов. Это слово представляет собой необработанный номер кластера (рассчитываемый номер в неявном виде).

Затем полученное слово обрабатывается следующим образом. Если номер первого кластера четный, то номер следующего кластера — это число из трех младших шестнадцатиричных цифр слова. Если он был нечетным, то номер следующего кластера состоит из трех старших цифр слова.

Чтобы найти номер следующего кластера, номер второго кластера опять умножается на 1,5. Затем по адресу, равному целой части произведения, из таблицы выбирается слово и обрабатывается аналогичным образом: если номер второго кластера четный, отбрасывается

старшая цифра, если нечетный – младшая. Полученное шестнадцатиричное число из трех цифр принимается за номер следующего кластера.

Этот процесс повторяется до тех пор, пока полученное в результате число не попадет в интервал значений FF8-FFF. Номера с этими значениями обозначают последний кластер файла.

Процедура вычисления номера кластера на дисках с большой памятью аналогична рассмотренной выше процедуре. На большом диске помещается более 4 096 кластеров, поэтому запись таблицы размещения не может состоять из 1,5 байтов. Под нее отводится 2 байта. Номер первого кластера считывается из директория и затем увеличивается вдвое. Произведение рассматривается как адрес следующего кластера внутри таблицы. То есть слово (2 байта) с полученным результатом представляет собой номер следующего кластера. Первым в таблице записан младший байт, вторым — старший.

В следующее таблице приведен список номеров кластера, интерпретирующихся операционной системой особым образом.

Значение	***Интерпретация***
(0)000	Свободный кластер
(F)FF0-(F)FF6	Кластер, используемый для системного пользования
(F)FF7	Запорченный кластер
(F)FF8-(F)FFF	Последний кластер файла
(X)XXX	Кластер входит в цепочку

4.3. Задание на выполнение

При исследовании диска работать ТОЛЬКО с ДИСКЕТОЙ

1. Подготовиться к работе, используя справочные материалы данного руководства, рекомендованную литературу и справочник TEACH-HELP.

2. Познакомиться с основным меню DE.EXE

3. Исследовать средства работы с диском на уровне DOS (кластер, логический сектор, BOOT, FAT, ROOT DIR).

4. Исследовать структуру загрузочного сектора системной и обычной дискеты.

5. Исследовать структуру таблицы распределения файлов.

6. Исследовать структуру корневого справочника.

7. Исследовать изменения в системной области диска при создании и удалении файла и способы восстановления удалённых файлов.

8. Сформулировать принцип восстановления удаленных файлов в DOS, условия восстановления и рекомендации пользователю по работе в системе увеличивающие шансы успешного восстановления.
(Речь идет не об использовании стандартной утилиты — например, UNDELETE, а о АЛГОРИТМЕ восстановления!)

Часть 2. Семестр 3 (курс 2)

ЛАБОРАТОРНАЯ РАБОТА №1 «Разработка и компиляция С-программы в среде UNIX»

1.1. Цель работы

Изучение работы с компилятором языка С и создание программ в среде UNIX.

1.2. Средства разработки программ

Система UNIX обеспечивает богатый набор средств для разработки программ, включающий компиляторы, линкер, символьный отладчик, средства ведения программных проектов и разработки языковых процессоров, архивные средства и другие.

Вызов компиляторов

В системе UNIX имеются компиляторы с языков С, ФОРТРАН и ПАСКАЛЬ и другие. Команды вызова компилятора имеют вид cc, f77 или fc, pc и т.п.

Параметрами этих команд являются файлы с текстами программ на исходных языках, имена которых должны оканчиваться на .c, .f, .p и т.п. соответственно.

Примеры:

```
$ cc program.c
$ fc test.f
$ pc example.p
```

Результатом работы компилятора является файл исполняемого кода, имеющий по умолчанию имя a.out. Если вы хотите другое имя, его можно указать явно ключом -o <имя> при вызове компилятора.

Пример:

```
$ fc -o test test.f
$ ls
test
test.f
$
```

Линкер

На практике программы создаются из множества раздельно транслируемых модулей, каждый из которых занимает отдельный

файл. Результатом компиляции каждого модуля является файл объектного (перемещяемого) кода, имя которого получается заменой .c (или .f, .p и т.д.) на .o. Затем все объектные файлы объединяются в единую программу, помещаемую в файл исполняемого кода, посредством линкера. Линкер может вызываться как независимой командой ld, так и автоматически при выполнении команд вызова компилятора cc, fc, pc и т.д. В последнем случае эти команды могут иметь несколько параметров-файлов, имена которых могут оканчиваться не только на .c, .f, .p, ..., но и на .o. Файлы исходного текста компилируются, а затем все файлы объектного кода, как полученные в результате компиляции, так и параметры команды вызова компилятора, передаются линкеру. Результатом, по-прежнему, является файл с именем по умолчанию a.out, если вы не указали явно другое имя.

При этом, как правило, объектные файлы уничтожаются. Чтобы сохранить их, можно подавить автоматический вызов линкера ключом -c (только компиляция) в команде вызова компилятора.

Пример:

```
$ fc -c test.f check prove.f
$ ld /lib/frt0.o *.o -lF77
$ ls
a.out
check.f
check.o
prove.f
prove.o
test.f
test.o
$
```

Здесь добавлены файл /lib/frt0.o стартового модуля для программы на ФОРТРАНе (для C /lib/crt0.o) и библиотека - lF77 подпрограмм для ФОРТРАНа (для C - lc); могут быть добавлены и другие библиотеки. Обозначение -lx является сокращением для /lib/libx.a для любого x. Следует заметить, что библиотеки указываются последними (не являются ключами команды ld). При автоматическом вызове линкера стартовый модуль и ряд библиотек вызываются по умолчанию. Чтобы их увидеть, следует применить ключ -v в командах вызова компилятора.

Библиотеки

Как мы показали выше, на вход линкера могут подаваться не только файлы объектного кода, но и библиотечные файлы, которые

оказываются очень удобным средством хранения объектных модулей, если их становится очень много.

Имя библиотечного файла обычно оканчивается на .a. Имеется команда ar (архив) для создания, пополнения и просмотра библиотечных файлов.

Пример создания библиотеки из трех объектных файлов:

```
$ ar rcv exam.a test.o check.o prove.o
a - test.o
a - check.o
a - prove.o
$
```

Здесь ключи команды ar означают:

r - заменить (replace) модули в библиотеке;
c - создать (create) библиотеку;
v - печатать включаемые модули (verbose).

Можно распечатать содержимое библиотеки командой ar с ключом t (table of content):

```
$ ar t exam.a
test.o
check.o
prove.o
$
```

и ссылаться на библиотеку в командах вызова компиляторов или линкера, например:

```
$ ld -o test /lib/frt0.o exam.a -lF77
```

Следует помнить, что порядок размещения модулей в библиотеке существенен.

Например, если подпрограмма test вызывает подпрограмму check, то test.o должен предшествовать check.o в библиотеке.

Для выявления и печати таких зависимостей предназначена команда lorder.

Символьный отладчик

Как правило, имеется единый символьный отладчик для программ на нескольких языках программирования, компиляторы которых вырабатывают объектный код и таблицы символов в едином формате. Символьный отладчик обычно имеет альтернативные имена для разных языков, например, cdb, fdb и pdb для языков C, ФОРТРАН-77 и

PASCAL соответственно. Вызов отладчика производится одноименной командой, в которой указывается файл кода отлаживаемой программы, по умолчанию a.out. Для генерации и сохранения отладочной информации (таблиц символов и т.п.) компиляция программы должна производиться с ключом -g.

Пример:

```
$ fc -g test.f check.f prove.f
$ fdb
>
```

Знак > — подсказка отладчика, приглашающая вводить команды отладчика. Команды отладчика позволяют:

- посмотреть текущие значения переменных выполняемого оператора, строки исходного текста, процедуры, файла в указанном формате;
- устанавливать и сбрасывать точки прерывания для пошагового выполнения отлаживаемой программы между точками прерывания и/или с постоянным шагом;
- задавать утверждения, проверяемые перед выполнением каждого оператора для останова перед теми операторами, для которых утверждение истинно;
- копировать все сигналы, связанные с отлаживаемой программой;
- записывать и повторно использовать команды сеанса отладки (командный файл на входе отладчика).

Более подробную информацию о символьном отладчике можно найти в руководстве по команде cdb.

Средства оценки эффективности исполнения программы

Имеются средства, позволяющие выяснить, сколько времени и места требуется для выполнения программы и ее частей. Эти средства полезны для сравнения качества реализации программы на разных языках программирования или в разных версиях системы UNIX.

Команда time позволяет выяснить три значения потраченного на решение указанной программы времени в секундах: реального, равного времени, замеренному по секундомеру, зависящего от количества пользователей в данный момент; пользовательского, равного времени счета программы по командам пользователя, и системного, равного времени системы при обслуживании этой программы. Команда time в Shell и в C-Shell имеет разные формы выдачи.

Пример (в Shell):

```
$ time test
```

```
real 30.0
user 27.6
sys 0.5
$
```

Команда size показывает размер памяти в байтах трех различных частей программы: выполняемого кода (называемого текстом), области инициируемых данных и области неинициируемых данных.

Пример:

```
$ size test
1356 + 260 + 613 = 2226 b = 004265 b
```

Общий размер программы test равен 4265 байтов. Если имя оцениваемого файла отсутствует, подразумевается a.out.

Для программы на языке С есть более тонкое средство, позволяющее получить профиль программы, показывающий для каждой подпрограммы долю времени ее выполнения в процентах (% time), суммарное время ее выполнения в секундах (cumsecs), количество ее вызовов (# call) и время, потраченное на один вызов в миллисекундах (ms/call). Для получения профиля следует перетранслировать программу с ключом -p (профиль), а после запуска программы, во время которого статистическая информация будет собрана в файле mon.out, выполнить команду prof для обработки указанного файла.

Пример:

```
$ cc -p program.c
$ mv a.out program
$ program
$ ls
mon.out
program
program.c
$ prof program
name % time cumsecs # call ms/call
conv 58.6 11.38 2000 5.69
strcat 30.1 9.50 100 95.0
main 1.1 2.1 1 2.1
. . . . . .
atoi 0.0 0.1 1 0.0
```

1.3. Создание процессов

На самом высоком уровне абстракции система состоит из множества процессов. Каждый процесс ответственен за обеспечение служебных функций определенного характера.

Любой поток может осуществить запуск процесса. Однако необходимо учитывать ограничения, вытекающие из основных принципов защиты.

Обычно разработчиков программного обеспечения не заботит тот факт, что командный интерпретатор создает процессы — это просто подразумевается. Однако в большой мультипроцессорной системе вы можете пожелать, чтобы одна главная программа выполняла запуск всех других процессов вашего приложения.

Рассмотрим функцию fork(). *fork()* — порождает процесс, являющийся его точной копией. Новый процесс выполняется в том же адресном пространстве и наследует все данные порождающего процесса.

Между тем, родительский и дочерний процесс имеют различные идентификаторы процессов, так как в системе не может быть двух процессов с одинаковыми идентификаторами. Есть и еще одно отличие, это значение, возвращаемое функцией fork(). В дочернем процессе функция возвращает ноль, а в родительском процессе идентификатор дочернего процесса.

Пример, использования функции fork():

```
printf("PID родителя равен %d\n", getpid());
if (child_pid = fork()) {
printf("Это родитель, PID сына %d\n", child_pid);
} else {
printf("Это сын, PID %d\n", getpid());
}
```

1.4. Задание на выполнение

1. Изучить процедуру компиляции.

2. Повторить стандартный ввод – вывод, разбор аргументов и переменных среды. Исследовать работу функций fopen(), fclose(), fprintf(), функция fork() - запуск программы в фоновом режиме, в системе unix.

3. Разработать программу, запускающуюся в фоне (сервер). При запуске она проверяет, запущен ли сервер. Если он запущен,

то программа удаляет ранее запущенный сервер и завершает свою работу. Если сервер не запущен, то программа запускает сервер в работу (вешает фоновый процесс). Идентификатор запущенного сервера можно хранить в файле.

ЛАБОРАТОРНАЯ РАБОТА №2 «Разработка программы моделирующей различные механизмы планирования заданий»

2.1. Цель работы

Изучение механизмов планирования заданий

2.2. Диспетчеризация потоков

В настоящее время обязательным условием, предъявляемым к операционным системам, претендующих на применение многозадачности, является реализация в них ***механизмов многозадачности***.

Многозадачность подразумевает параллельное выполнение нескольких действий, однако практическая реализация параллельной работы упирается в проблему совместного использования ресурсов вычислительной системы. И главным ресурсом, распределение которого между несколькими задачами называется ***диспетчеризацией*** (scheduling), является процессор. Поэтому в однопроцессорной системе по-настоящему параллельное выполнение нескольких задач невозможно. Существует достаточно большое количество различных методов диспетчеризации.

Под понятием задачи в терминах ОС и программных комплексов могут пониматься две разные вещи: ***процессы и потоки***. Процесс является более масштабным представлением задачи, поскольку обозначает независимый модуль программы или весь исполняемый файл целиком с его адресным пространством, состоянием регистров процессора, счетчиком команд, кодом процедур и функций. Поток же является составной частью процесса и обозначает последовательность исполняемого кода. Каждый процесс содержит как минимум один поток, при этом максимальное количество потоков в пределах одного процесса в большинстве ОС ограничено только объемом оперативной памяти вычислительного комплекса. Потоки, принадлежащие одному процессу, разделяют его адресное пространство, поэтому они могут легко обмениваться данными, а время переключения между такими потоками (то есть время, за которое процессор переходит от выполнения команд одного потока к выполнению команд другого) оказывается значительно меньшим, чем время переключение между процессами. В связи с этим в задачах реального времени параллельно выполняемые задачи стараются максимально компоновать в виде потоков, исполняющихся в пределах одного процесса.

Каждый поток имеет важное свойство, на основании которого ОС принимает решение о том, когда предоставить ему время процессора. Это свойство называется ***приоритетом потока*** и выражается целочисленным значением. Количество приоритетов (или уровней приоритетов) определяется функциональными возможностями ОС, при этом самое низкое значение (0) закрепляется за потоком, который предназначен для корректной работы системы, когда ей "ничего не надо делать".

Поток может находиться в одном из следующих состояний:

– Активный поток — это тот поток, который в данный момент выполняется системой.
– Поток в состоянии готовности — поток, который может выполняться и ждет своей очереди.
– Блокированный поток — поток, который не может выполняться по некоторым причинам (например, ожидание события или освобождения нужного ресурса).

Методы диспетчеризации, т.е. предоставления разным потокам доступа к процессору, в общем случае могут быть разделены на две группы. К первой относятся случаи, когда все потоки, которые разделяют процессор, имеют одинаковый приоритет, т.е. их важность с точки зрения системы одинакова:

– ***FIFO (First In First Out).*** Первой выполняется задача, первой вошедшая в очередь, при этом она выполняется до тех пор, пока не закончит свою работу или не будет заблокирована в ожидании освобождения некоторого ресурса или события. После этого управление передается следующей в очереди задаче.
– ***Карусельная многозадачность*** (round robin). При этом методе диспетчеризации в системе задается специализированная константа, определяющая продолжительность непрерывного выполнения потока, так называемый квант времени выполнения (time slice). Таким образом, выполнение потока может быть прервано либо окончанием его работы, либо блокированием в ожидании ресурса или события, либо завершением кванта времени (того самого time slice). После этого управление передается следующему в очередности потоку. По окончании времени последнего потока управление передается первому потоку, находящемуся в состоянии готовности. Таким образом, выполнение каждого потока разбито на последовательность временных циклов.

Появление второй группы методов диспетчеризации связано с необходимостью распределения времени процессора между потоками,

имеющими разную важность, т.е. разный приоритет. В таких случаях для потоков с равным приоритетом используется один из указанных выше методов диспетчеризации, а передача управления между потоками с разным приоритетом осуществляется одним из следующих методов:

– В наиболее простом случае если в состояние готовности переходят два потока с разными приоритетами, то процессорное время передается тому, у которого более высокий приоритет. Данный метод называется ***приоритетной многозадачностью***, но его использование в таком виде связано с рядом сложностей. При наличии в системе одной группы потоков с одним приоритетом и другой группы с другим, более низким приоритетом, при карусельной диспетчеризации каждой группы в системе с приоритетной многозадачностью потоки низкоприоритетной группы могут вообще не получить доступа к процессору.

– Одним из решений проблем приоритетной многозадачности стала так называемая ***адаптивная многозадачность***, широко применяющаяся в интерфейсных системах. Суть метода заключается в том, что приоритет потока, не выполняющегося какой-то период времени, повышается на единицу. Восстановление исходного приоритета происходит после выполнения потока в течение одного кванта времени или при блокировке потока. Таким образом, при карусельной многозадачности, очередь (или "карусель") более приоритетных потоков не может полностью заблокировать выполнение очереди менее приоритетных потоков.

– В задачах реального времени предъявляются специфические требования к методам диспетчеризации, поскольку передача управления потоку должна определяться критическим сроком его обслуживания (т.н. deadline-driven scheduling). В наибольшей степени этому требованию соответствует ***вытесняющая приоритетная многозадачность***. Суть этого метода заключается в том, что как только поток с более высоким, чем у активного потока, приоритетом переходит в состояние готовности, активный поток вытесняется (т.е. из активного состояния принудительно переходит в состояние готовности) и управление передается более приоритетному потоку.

На практике широко применяются как комбинации описанных методов, так и различные их модификации.

2.3. Задание на выполнение

Реализуйте практическое задание на любом языке высокого уровня, с использованием платформ Windows или Unix.

Создайте программу, реализующую механизмы планирования выполнения заданий: FIFO, Round Robin, приоритетная многозадачность, адаптивная многозадачность.

Для этого создайте в программе класс, имеющий свойства Number=0, Priority=N и метод AddNumber(), в котором происходит инкрементирование (увеличение на единицу) свойства Number.

По нажатию на кнопку окна программы в ней генерируются несколько экземпляров этого класса (например, 5), и запускается цикл или периодическое событие (перед стартом работы события, выбирается тип используемого механизма планирования), которое в зависимости от выбранного вида планирования заданий вызывает метод AddNumber() какого-либо экземпляра объекта.

На экране в виде таблицы печатается номер экземпляра, его приоритет Priority[3] и свойство Number.

Программа не должна «зависать» и может быть закрыта в любой момент времени.

Подготовьте отчет о работе программы.

[3] Для адаптивного планирования — текущий и начальный приоритеты.

ЛАБОРАТОРНАЯ РАБОТА №3 «Изучение основ программирования на языке Ассемблер»

3.1. Цель работы

Целью данной работы является изучение структуры программы на ассемблере с использованием различных директив сегментации; изучение функций ввода/вывода, арифметических и логических команд микропроцессора i8086; изучение написания процедур и макросов с использованием модульной структуры программы.

3.2. Структура программы на ассемблере

Программа на ассемблере представляет собой совокупность блоков памяти, называемых сегментами памяти. Программа может состоять из одного или нескольких таких блоков-сегментов. Каждый сегмент содержит совокупность предложений языка, каждое из которых занимает отдельную строку кода программы.

Предложения ассемблера бывают четырех типов:

- *команды или инструкции*, представляющие собой символические аналоги машинных команд. В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды системы команд микропроцессора;
- *макрокоманды* – оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями;
- *директивы*, являющиеся указанием транслятору ассемблера на выполнение некоторых действий. У директив нет аналогов в машинном представлении;
- *строки комментариев*, содержащие любые символы, в том числе и буквы русского алфавита. Комментарии игнорируются транслятором.

3.2.1. Синтаксис ассемблера

Предложения, составляющие программу, могут представлять собой синтаксическую конструкцию, соответствующую команде, макрокоманде, директиве или комментарию. Для того чтобы транслятор ассемблера мог распознать их, они должны формироваться по определенным синтаксическим правилам.

Формат предложений ассемблера[4]:

```
Оператор директивы [;текст комментария]
Оператор команды [;текст комментария]
Оператор макрокоманды [; текст комментария]
```

Формат директив:

```
[Имя] директива [операнд1…операндN][;комментарий]
```

Формат команд и макрокоманд

```
[Имя метки:] КОП [операнд1…операндN][;комментарий]
```

Здесь:

- *имя метки* – идентификатор, значением которого является адрес первого байта того предложения исходного текста программы, которое он обозначает;
- *имя* – идентификатор, отличающий данную директиву от других одноименных директив;
- *код операции (КОП) и директива* – это мнемонические обозначения соответствующей машинной команды, макрокоманды или директивы транслятора;
- *операнды* – части команды, макрокоманды или директивы ассемблера, обозначающие объекты, над которыми производятся действия. Операнды ассемблера описываются выражениями с числовыми и текстовыми константами, метками и идентификаторами переменных с использованием знаков операций и некоторых зарезервированных слов.

Допустимыми символами при написании текста программ являются:

- все латинские буквы: **A–Z**, **a–z**. При этом заглавные и строчные буквы считаются эквивалентными;
- цифры от **0** до **9**;
- знаки **?**, **@**, **$**, **_**, **&**;
- разделители **, . [ ] ( )** < > **{ }** + / * **%** ! ' " **?** \ = # ^.

Предложения ассемблера формируются из *лексем*, представляющих собой синтаксически неразделимые последовательности допустимых символов языка, имеющие смысл для транслятора.

Лексемами являются:

- *идентификаторы* – последовательности допустимых символов, использующиеся для обозначения таких объектов программы, как

[4] Квадратные скобки означают не обязательные параметры

коды операций, имена переменных и названия меток. Правило записи идентификаторов заключается в следующем: идентификатор может состоять из одного или нескольких символов. В качестве символов можно использовать буквы латинского алфавита, цифры и некоторые специальные знаки – _, ?, $, @. Идентификатор не может начинаться символом цифры. Длина идентификатора может быть до 255 символов, хотя транслятор воспринимает лишь первые 32, а остальные – игнорирует. Регулировать длину возможных идентификаторов можно с использованием опции командной строки **mv**. Кроме этого, существует возможность указать транслятору на то, чтобы он различал прописные и строчные буквы либо игнорировал их различие (что и делается по умолчанию). Для этого применяются опции командной строки **/mu, /ml, /mx**;

- *цепочки символов* – последовательности символов, заключенные в одинарные или двойные кавычки;
- *целые числа* в одной из следующих систем счисления: *двоичной, десятичной, шестнадцатеричной*. Отождествление чисел при записи их в программах на ассемблере производится по определенным правилам:
  - **Десятичные числа** не требуют для своего отождествления указания каких-либо дополнительных символов, например 25 или 139.
  - Для отождествления в исходном тексте программы **двоичных чисел** необходимо после записи нулей и единиц, входящих в их состав, поставить латинское “**b**”, например 10010101**b.**
  - **Шестнадцатеричные числа** имеют больше условностей при своей записи:
    - *Во-первых*, они состоят из цифр **0...9**, строчных и прописных букв латинского алфавита **a**, **b**, **c**, **d**, **e**, **f** или **A**, **B**, **C**, **D**, **E**, **F.**
    - *Во-вторых*, у транслятора могут возникнуть трудности с распознаванием шестнадцатеричных чисел из-за того, что они могут состоять как из одних цифр 0...9 (например, 190845), так и начинаться с буквы латинского алфавита (например, **ef15**). Для того чтобы "объяснить" транслятору, что данная лексема не является десятичным числом или идентификатором, программист должен специальным образом выделять шестнадцатеричное число. Для этого, на конце последовательности шестнадцатеричных цифр, составляющих шестнадцатеричное число, записывают латинскую букву “**h**”. Это обязательное условие. Если шест-

надцатеричное число начинается с буквы, то перед ним записывается ведущий ноль: **0**ef15**h.**

3.2.2. Директивы сегментации

В ходе предыдущего обсуждения мы выяснили все основные правила записи команд и операндов в программе на ассемблере. Открытым остался вопрос о том, как правильно оформить последовательность команд, чтобы транслятор мог их обработать, а микропроцессор – выполнить.

При рассмотрении архитектуры микропроцессора мы узнали, что он имеет шесть сегментных регистров, посредством которых может одновременно работать:

- с одним сегментом кода;
- с одним сегментом стека;
- с одним сегментом данных;
- с тремя дополнительными сегментами данных.

Еще раз вспомним, что физически сегмент представляет собой область памяти, занятую командами и (или) данными, адреса которых вычисляются относительно значения в соответствующем сегментном регистре.

Синтаксическое описание сегмента на ассемблере представляет собой следующую конструкцию:

```
Имя сегмента SEGMENT [тип выравнивания] [тип комбинирования] [класс сегмента] [тип размера сегмента]
    …
    содержимое сегмента
    …
Имя сегмента ENDS
```

Важно отметить, что функциональное назначение сегмента несколько шире, чем простое разбиение программы на блоки кода, данных и стека. Сегментация является частью общего механизма, связанного с концепцией модульного программирования. Она предполагает унификацию оформления объектных модулей, создаваемых компилятором, в том числе с разных языков программирования. Это позволяет объединять программы, написанные на разных языках. Именно для реализации различных вариантов такого объединения и предназначены операнды в директиве SEGMENT. Рассмотрим их подробнее:

- *Атрибут выравнивания сегмента* (тип выравнивания) сообщает компоновщику о том, что нужно обеспечить размещение начала

сегмента на заданной границе. Это важно, поскольку при правильном выравнивании доступ к данным в процессорах i80x86 выполняется быстрее. Допустимые значения этого атрибута следующие:

- BYTE – выравнивание не выполняется. Сегмент может начинаться с любого адреса памяти;
- WORD – сегмент начинается по адресу, кратному двум, то есть последний (младший) значащий бит физического адреса равен 0 (выравнивание на границу слова);
- DWORD – сегмент начинается по адресу, кратному четырем, то есть два последних (младших) значащих бита, равны 0 (выравнивание на границу двойного слова);
- PARA – сегмент начинается по адресу, кратному 16, то есть последняя шестнадцатеричная цифра адреса должна быть 0h (выравнивание на границу параграфа);
- PAGE – сегмент начинается по адресу, кратному 256, то есть две последние шестнадцатеричные цифры должны быть 00h (выравнивание на границу 256-байтной страницы);
- MEMPAGE – сегмент начинается по адресу, кратному 4 Кбайт, то есть три последние шестнадцатеричные цифры должны быть 000h (адрес следующей 4-Кбайтной страницы памяти).

По умолчанию тип выравнивания имеет значение PARA.

- *Атрибут комбинирования сегментов* (комбинаторный тип) сообщает компоновщику, как нужно комбинировать сегменты различных модулей, имеющие одно и то же имя. Значениями атрибута комбинирования сегмента могут быть:
  - PRIVATE – сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля;
  - PUBLIC – заставляет компоновщик соединить все сегменты с одинаковыми именами. Новый объединенный сегмент будет целым и непрерывным. Все адреса (смещения) объектов, а это могут быть, в зависимости от типа сегмента, команды и данные, будут вычисляться относительно начала этого нового сегмента;
  - COMMON – располагает все сегменты с одним и тем же именем по одному адресу. Все сегменты с данным именем будут перекрываться и совместно использовать память. Размер полученного в результате сегмента будет равен размеру самого большого сегмента;

- AT xxxx – располагает сегмент по абсолютному адресу параграфа (параграф – объем памяти, кратный 16, поэтому последняя шестнадцатеричная цифра адреса параграфа равна 0);
- STACK – определение сегмента стека. Заставляет компоновщик соединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра ss.

По умолчанию атрибут комбинирования принимает значение PRIVATE.

- *Атрибут класса сегмента* (тип класса) – это заключенная в кавычки строка, помогающая компоновщику определить соответствующий порядок следования сегментов при собирании программы из сегментов нескольких модулей. Компоновщик объединяет в памяти все сегменты с одним и тем же именем класса (имя класса, в общем случае, может быть любым, но лучше, если оно будет отражать функциональное назначение сегмента);
- *Атрибут размера сегмента*. Для процессоров i80386 и выше сегменты могут быть 16 или 32-разрядными. Это влияет, прежде всего, на размер сегмента и порядок формирования физического адреса внутри него. Атрибут может принимать следующие значения:
  - USE16 – это означает, что сегмент допускает 16-разрядную адресацию. При формировании физического адреса может использоваться только 16-разрядное смещение. Соответственно, такой сегмент может содержать до 64 Кбайт кода или данных;
  - USE32 – сегмент будет 32-разрядным. При формировании физического адреса может использоваться 32-разрядное смещение. Поэтому такой сегмент может содержать до 4 Гбайт кода или данных.

Все сегменты сами по себе равноправны, так как директивы SEGMENT и ENDS не содержат информации о функциональном назначении сегментов. Для того чтобы использовать их как сегменты кода, данных или стека, необходимо предварительно сообщить транслятору об этом, для чего используют специальную директиву ASSUME. Эта директива сообщает транслятору о том, какой сегмент, к какому сегментному регистру привязан. В свою очередь, это позволит транслятору корректно связывать символические имена, определенные в сегментах.

Далее приведем пример программы с использованием стандартных директив сегментации:

```
data segment para public 'data' ;сегмент данных
      message db 'Hello World,$' ; описание строки
```

```
data ends

stk segment stack
      db 256 dup ('?') ; размер сегмента стека
stk ends

code segment para public 'code' ; начало сегмента
 ; кода
main proc ;начало процедуры main
      assume cs:code,ds:data,ss:stk
 mov  ax,data ; адрес сегмента данных
 ; в регистр ax
      mov   ds,ax ; ax в ds

      ...

      mov   ah,9
      mov   dx,offset message
      int   21h ; ah=9 функция 21h прерывания
 ; выводит строку на экран, адрес
 ; которой храниться в регистре dx,
 ; строка должна обязательно
 ; заканчиваться символом $
      ...

      mov   ax,4c00h ; пересылка 4c00h в регистр ax
      int   21h ; вызов прерывания с номером 21h
main  endp ; конец процедуры main
code  ends ; конец сегмента кода
end   main ; конец программы с точкой входа main
```

Для простых программ, содержащих по одному сегменту для кода, данных и стека, хотелось бы упростить ее описание. Для этого в трансляторы MASM и TASM ввели возможность использования *упрощенных директив сегментации*. Но здесь возникла проблема, связанная с тем, что необходимо было как-то компенсировать невозможность напрямую управлять размещением и комбинированием сегментов. Для этого, совместно с упрощенными директивами сегментации, стали использовать директиву указания модели памяти **MODEL**, которая частично стала управлять размещением сегментов и выполнять функции директивы **ASSUME** (поэтому при использовании упрощенных директив сегментации директиву ASSUME можно не использовать). Эта

директива связывает сегменты, которые в случае использования упрощенных директив сегментации имеют предопределенные имена с сегментными регистрами (хотя явно инициализировать *ds* все равно придется).

Теперь, перепишем вышеприведенную программу с использованием упрощенных директив сегментации.

```
masm ;режим работы TASM: ideal или masm
model small ; модель памяти

.data ; сегмент данных
 message db 'Hello World,$' ; описание строки

.stack ;сегмент стека
 db 256 dup ('?') ; сегмент стека

.code ;сегмент кода
main proc ; начало процедуры main
 mov ax,@data ; заносим адрес сегмента данных в
 ; регистр ax
 mov ds,ax ;ax в ds

      ...

 mov  ah,9
 mov  dx,offset message
 int  21h ; ah=9 функция 21h прерывания
 ; выводит строку на экран, адрес
 ; которой храниться в регистре dx
      ...

 mov ax,4c00h ; пересылка 4c00h в регистр ax
 int 21h ; вызов прерывания с номером 21h
main endp ; конец процедуры main
end main ; конец программы с точкой входа main
```

Обязательным параметром директивы MODEL является *модель памяти*. Этот параметр определяет модель сегментации памяти для программного модуля. Предполагается, что программный модуль может иметь только определенные типы сегментов, которые определяются упомянутыми нами ранее *упрощенными директивами описания сегментов*. Эти директивы приведены в таблице 1.

Таблица 1. Упрощенные директивы определения сегмента

Формат директивы (режим MASM)	**Формат директивы (режим IDEAL)**	**Назначение**
.CODE [имя]	CODESEG[имя]	Начало или продолжение сегмента кода
.DATA	DATASEG	Начало или продолжение сегмента инициализированных данных. Также используется для определения данных типа near
.CONST	CONST	Начало или продолжение сегмента постоянных данных (констант) модуля
.FARDATA [имя]	FARDATA [имя]	Начало или продолжение сегмента инициализированных данных типа far

Наличие в некоторых директивах параметра **[имя]** говорит о том, что возможно определение нескольких сегментов этого типа. С другой стороны, наличие нескольких видов сегментов данных обусловлено требованием обеспечения совместимости с некоторыми компиляторами языков высокого уровня, которые создают разные сегменты данных для инициализированных и неинициализированных данных, а также констант.

При использовании директивы **MODEL** транслятор делает доступными несколько идентификаторов, к которым можно обращаться во время работы программы, с тем, чтобы получить информацию о тех или иных характеристиках данной модели памяти (таблица 3). Перечислим эти идентификаторы и их значения (табл. 2).

Таблица 2. Модели памяти

Модель	**Тип кода**	**Тип данных**	**Назначение модели**
TINY	near	near	Код и данные объединены в одну группу с именем DGROUP. Используется для создания программ формата .com.
SMALL	near	near	Код занимает один сегмент, данные объединены в одну группу с именем

			DGROUP. Эту модель обычно используют для большинства программ на ассемблере
MEDIUM	far	near	Код занимает несколько сегментов, по одному на каждый объединяемый программный модуль. Все ссылки на передачу управления – типа far. Данные объединены в одной группе; все ссылки на них – типа near
COMPACT	near	far	Код в одном сегменте; ссылка на данные – типа far
LARGE	far	far	Код в нескольких сегментах, по одному на каждый объединяемый программный модуль

Параметр модификатор директивы MODEL позволяет уточнить некоторые особенности использования выбранной модели памяти (табл. 3).

Таблица 3. Модификаторы модели памяти

Значение модификатора	**Назначение**
use16	Сегменты выбранной модели используются как 16-битные (если соответствующей директивой указан процессор i80386 или i80486)
use32	Сегменты выбранной модели используются как 32-битные (если соответствующей директивой указан процессор i80386 или i80486)
dos	Программа будет работать в MS-DOS

Необязательные параметры – язык и модификатор языка, определяют некоторые особенности вызова процедур. Необходимость в использовании этих параметров появляется при написании и связывании программ на различных языках программирования.

Описанные нами стандартные и упрощенные директивы сегментации не исключают друг друга. Стандартные директивы используются, когда программист желает получить полный контроль над размещением сегментов в памяти и их комбинированием с сегментами других модулей.

Упрощенные директивы целесообразно использовать для простых программ и программ, предназначенных для связывания с программными модулями, написанными на языках высокого уровня. Это позволяет компоновщику эффективно связывать модули разных языков за счет стандартизации связей и управления.

3.2.3. Создание COM-программ

Все вышеприведенные директивы сегментации и примеры программ предназначены для создания программ в EXE-формате[5]. Компоновщик LINK автоматически генерирует особый формат для EXE-файлов, в котором присутствует специальный начальный блок (заголовок) размером не менее 512 байт.

Для выполнения можно также создавать COM-файлы. Примером часто используемого COM-файла является COMMAND.COM.

Размер программы. EXE-программа может иметь любой размер, в то время как COM-файл ограничен размером одного сегмента и не превышает 64K. COM-файл всегда меньше, чем соответствующий EXE-файл; одна из причин этого - отсутствие в COM-файле 512-байтового начального блока EXE-файла.

Сегмент стека. В EXE-программе определяется сегмент стека, в то время как COM-программа генерирует стек автоматически. Таким образом, при создании ассемблерной программы, которая будет преобразована в COM-файл, стек должен быть опущен.

Сегмент данных. В EXE программе обычно определяется сегмент данных, а регистр DS инициализируется адресом этого сегмента. В COM-программе все данные должны быть определены в сегменте кода. Ниже будет показан простой способ решения этого вопроса.

Инициализация. EXE-программа записывает нулевое слово в стек и инициализирует регистр DS. Так как COM-программа не имеет ни стека, ни сегмента данных, то эти шаги отсутствуют.

Когда COM-программа начинает работать, все сегментные регистры содержат адрес префикса программного сегмента (PSP), – 256-байтового (шест. 100) блока, который резервируется операционной системой DOS непосредственно перед COM или EXE программой в памяти. Так как адресация начинается с шест. смещения 100 от начала PSP, то в программе после оператора SEGMENT кодируется директива ORG 100H.

[5] За исключением модели памяти TINY при использовании упрощенных директив сегментации.

Обработка. Для программ в EXE и COM форматах выполняется ассемблирование для получения OBJ-файла, и компоновка для получения EXE-файла. Если программа создается для выполнения как EXE-файл, то ее уже можно выполнить. Если же программа создается для выполнения как COM-файл, то компоновщиком будет выдано сообщение:

Warning: No STACK Segment
(Предупреждение: Сегмент стека не определен)

Ниже приведем пример COM-программы:

```
CSEG  Segment 'Code'
assume CS:CSEG,DS:CSEG,ES:CSEG,SS:CSEG
org   100h
start:
      ...

      mov   ah,9
      mov   dx,offset message
      int   21h ; ah=9 функция 21h прерывания
 ; выводит строку на экран, адрес
 ; которой храниться в регистре dx
      ...

 int 20h ; выход из COM-программы
 message db 'Hello World,$' ; описание строки
ends
end start
```

3.3. Компиляция программ на ассемблере

Для написания программ на языке ассемблере вы можете воспользоваться любым текстовым редактором, поддерживающим кодировку ASCII-символов, например «Блокнот»/«Notepad» из ОС Windos или встроенным текстовым редактором FAR/DN/NC и др.

Для создания исполняемых файлов из программ написанных на языке ассемблере вам необходимо использовать компилятор TASM.EXE и линковщик TLINK.EXE.

TASM.EXE компилирует программные модули ассемблера в объектные модули OBJ. А TLINK.EXE из нескольких модулей делает один исполняемый файл EXE или COM. Более подробный синтаксис использования TASM.EXE и TLINK.EXE можно получить запустив эти программы без параметров.

3.4. Функции прерываний ввода/вывода

В операционной системе существует большая группа функций 21h прерывания (прерывания DOS). Небольшую часть этих функций составляют функции ввода вывода информации.

Для вызова какого-либо прерывания необходимо:

- в регистр AH занести номер функции прерывания;
- в зависимости от типа прерывания в какие-либо регистры занести дополнительные параметры;
- использовать команду int с указанием номера прерывания.

С 9h функцией прерывания 21h вы познакомились на предыдущей лабораторной работе, пример ее вызова приведен на странице 12.

В данной лабораторной работе вам понадобится помимо функции вывода строки использовать функции ввода и вывода символа.

Для вызова функции ввода символа используют 1h-функцию 21h прерывания. После нажатия символа на клавиатуре в регистре AL сохраняется код ASCII нажатого символа.

Для вывода символа на экран можно воспользоваться функцией 2h прерывания 21h. В регистр AL помещается ASCII-код символа и вызывается прерывание 21h.

Дополнительную информацию по различным функциям прерываний операционной системы можно взять в электронном справочнике thelp.

3.5. Арифметические команды

Сложение двоичных чисел без знака. Микропроцессор выполняет сложение операндов по правилам сложения двоичных чисел. Проблем не возникает до тех пор, пока значение результата не превышает размерности поля операнда. Например, при сложении операндов размером в байт результат не должен превышать число 255. Если это происходит, то результат оказывается неверным. К примеру, выполним сложение: 254 + 5 = 259 в двоичном виде. 11111110 + 0000101 = 1 00000011. Результат вышел за пределы восьми бит и правильное его значение укладывается в 9 бит, а в 8-битовом поле операнда осталось значение 3, что, конечно, неверно. В микропроцессоре этот исход сложения прогнозируется и предусмотрены специальные средства для фиксирования подобных ситуаций и их обработки. Так, для фиксирования ситуации выхода за разрядную сетку результата, как в данном случае, предназначен флаг переноса cf. Он располагается в бите 0 регистра флагов eflags/flags. Именно установкой этого флага фиксирует-

ся факт переноса единицы из старшего разряда операнда. Естественно, что программист должен предусматривать возможность такого исхода операции сложения и средства для корректировки. Это предполагает включение участков кода после операции сложения, в которых анализируется флаг cf. Анализ этого флага можно провести различными способами. Самый простой и доступный – использовать команду условного перехода jcc. Эта команда в качестве операнда имеет имя метки в текущем сегменте кода. Переход на эту метку осуществляется в случае, если в результате работы предыдущей команды флаг cf установился в 1. В системе команд микропроцессора имеются три команды двоичного сложения:

- inc операнд – операция инкремента, то есть увеличения значения операнда на 1;
- add операнд_1,операнд_2 – команда сложения с принципом действия: операнд_1 = операнд_1 + операнд_2
- adc операнд_1,операнд_2 – команда сложения с учетом флага переноса cf: операнд_1 = операнд_1 + операнд_2 + значение_cf

Обратите внимание на последнюю команду – это команда сложения, учитывающая перенос единицы из старшего разряда. Механизм появления такой единицы мы уже рассмотрели. Таким образом, команда adc является средством микропроцессора для сложения длинных двоичных чисел, размерность которых превосходит поддерживаемые микропроцессором длины стандартных полей.

Рассмотрим пример вычисления суммы чисел:

```
<3> ;prg1
<3> masm
<3> model small
<4> stack 256
<5> .data
<6> a db 254
<7> .code ;сегмент кода
<8> main:
<9> mov ax,@data
<10> mov ds,ax
<11> ...
<12> xor ax,ax
<13> add al,17
<14> add al,a
<15> jnc m1;если нет переноса, то перейти
 ;на m1
<16> adc ah,0;в ax сумма с учетом переноса
<17> m1: ...
```

```
<18> exit:
<19> mov ax,4c00h ;стандартный выход
<20> int 21h
<21> end main ;конец программы
```

В строках 13–14 создана ситуация, когда результат сложения выходит за границы операнда. Эта возможность учитывается строкой 15, где команда jnc (хотя можно было обойтись и без нее) проверяет состояние флага cf. Если он установлен в 1, то это признак того, что результат операции получился больше по размеру, чем размер операнда, и для его корректировки необходимо выполнить некоторые действия. В данном случае мы просто полагаем, что границы операнда расширяются до размера AX, для чего учитываем перенос в старший разряд командой ADC (строка 15).

Сложение двоичных чисел со знаком. Микропроцессор не подозревает о различии между числами со знаком и без знака. Вместо этого у него есть средства фиксирования возникновения характерных ситуаций, складывающихся в процессе вычислений. Некоторые из них мы рассмотрели при обсуждении сложения чисел без знака:

- флаг переноса cf, установка которого в 1 говорит о том, что произошел выход за пределы разрядности операндов;
- команду adc, которая учитывает возможность такого выхода (перенос из младшего разряда).

Другое средство – это регистрация состояния старшего (знакового) разряда операнда, которое осуществляется с помощью флага переполнения of в регистре eflags (бит 11).

Дополнительно к флагу of при переносе из старшего разряда устанавливается в 1 и флаг переноса cf. Так как микропроцессор не знает о существовании чисел со знаком и без знака, то вся ответственность за правильность действий с получившимися числами ложится на программиста. Проанализировать флаги cf и of можно командами условного перехода jc\jnc и jo\jno соответственно.

Что же касается команд сложения чисел со знаком, то они те же, что и для чисел без знака.

Вычитание двоичных чисел без знака. Как и при анализе операции сложения, порассуждаем над сутью процессов, происходящих при выполнении операции вычитания. Если уменьшаемое больше вычитаемого, то проблем нет, – разность положительна, результат верен. Если уменьшаемое меньше вычитаемого, возникает проблема: результат

меньше 0, а это уже число со знаком. В этом случае результат необходимо завернуть. Что это означает? При обычном вычитании (в столбик) делают заем 1 из старшего разряда. Микропроцессор поступает аналогично, то есть занимает 1 из разряда, следующего за старшим, в разрядной сетке операнда.

Таким образом, после команды вычитания чисел без знака нужно анализировать состояние флага cf. Если он установлен в 1, то это говорит о том, что произошел заем из старшего разряда и результат получился в дополнительном коде.

Аналогично командам сложения, группа команд вычитания состоит из минимально возможного набора. Эти команды выполняют вычитание по алгоритмам, которые мы сейчас рассматриваем, а учет особых ситуаций должен производиться самим программистом. К командам вычитания относятся:

- dec операнд – операция декремента, то есть уменьшения значения операнда на 1;
- sub операнд_1,операнд_2 – команда вычитания; ее принцип действия:
  операнд_1 = операнд_1 – операнд_2
- sbb операнд_1,операнд_2 – команда вычитания с учетом заема (флага cf): операнд_1 = операнд_1 – операнд_2 – значение_cf

Как видите, среди команд вычитания есть команда sbb, учитывающая флаг переноса cf. Эта команда подобна adc, но теперь уже флаг cf выполняет роль индикатора заема 1 из старшего разряда при вычитании чисел.

Рассмотрим пример программной обработки ситуации:

```
<1> ;prg2
<2> masm
<3> model small
<4> stack 256
<5> .data
<6> .code ;сегмент кода
<7> main: ;точка входа в программу
<8> ...
<9> xor ax,ax
<10> mov al,5
<11> sub al,10
<12> jnc m1 ;нет переноса?
<13> neg al ;в al модуль результата
<14> m1: ...
<15> exit:
```

```
<16> mov ax,4c00h ;стандартный выход
<17> int 21h
<18> end main ;конец программы
```

В этом примере в строке 11 выполняется вычитание. С указанными для этой команды вычитания исходными данными результат получается в дополнительном коде (отрицательный). Для того чтобы преобразовать результат к нормальному виду (получить его модуль), применяется команда neg, с помощью которой получается дополнение операнда. В нашем случае мы получили дополнение дополнения или модуль отрицательного результата. А тот факт, что это на самом деле число отрицательное, отражен в состоянии флага cf. Дальше все зависит от алгоритма обработки.

Вычитание двоичных чисел со знаком. Здесь все несколько сложнее. Последний пример показал то, что микропроцессору незачем иметь два устройства – сложения и вычитания. Достаточно наличия только одного – устройства сложения. Но для вычитания способом сложения чисел со знаком в дополнительном коде необходимо представлять оба операнда – и уменьшаемое, и вычитаемое. Результат тоже нужно рассматривать как значение в дополнительном коде. Но здесь возникают сложности. Прежде всего, они связаны с тем, что старший бит операнда рассматривается как знаковый.

Отследить ситуацию переполнения мантиссы можно по содержимому флага переполнения of. Его установка в 1 говорит о том, что результат вышел за диапазон представления знаковых чисел (то есть изменился старший бит) для операнда данного размера, и программист должен предусмотреть действия по корректировке результата.

Умножение чисел без знака. Для умножения чисел без знака предназначена команда

mul сомножитель_1

Как видите, в команде указан всего лишь один операнд-сомножитель. Второй операнд – сомножитель_2 задан неявно. Его местоположение фиксировано и зависит от размера сомножителей. Так как в общем случае результат умножения больше, чем любой из его сомножителей, то его размер и местоположение должны быть тоже

определены однозначно. Варианты размеров сомножителей и размещения второго операнда и результата приведены в таблице 4.

Таблица 4. Расположение операндов и результата при умножении

сомножитель_1	**сомножитель_2**	**Результат**
Байт	al	16 бит в ax: al – младшая часть результата; ah – старшая часть результата
Слово	ax	32 бит в паре dx:ax: ax – младшая часть результата; dx – старшая часть результата
Двойное слово	eax	64 бит в паре edx:eax: eax – младшая часть результата; edx – старшая часть результата

Из таблицы видно, что произведение состоит из двух частей и в зависимости от размера операндов размещается в двух местах – на месте сомножитель_2 (младшая часть) и в дополнительном регистре ah, dx, edx (старшая часть). Как же динамически (то есть во время выполнения программы) узнать, что результат достаточно мал и уместился в одном регистре или что он превысил размерность регистра, и старшая часть оказалась в другом регистре? Для этого привлекаются уже известные нам по предыдущему обсуждению флаги переноса cf и переполнения of:

- если старшая часть результата нулевая, то после операции произведения флаги cf = 0 и of = 0;
- если же эти флаги ненулевые, то это означает, что результат вышел за пределы младшей части произведения и состоит из двух частей, что и нужно учитывать при дальнейшей работе.

Умножение чисел со знаком. Для умножения чисел со знаком предназначена команда

imul операнд_1[,операнд_2,операнд_3]

Эта команда выполняется так же, как и команда mul. Отличительной особенностью команды imul является только формирование знака.

Если результат мал и умещается в одном регистре (то есть если cf = of = 0), то содержимое другого регистра (старшей части) является

расширением знака – все его биты равны старшему биту (знаковому разряду) младшей части результата.

В противном случае, (если cf = of = 1) знаком результата является знаковый бит старшей части результата, а знаковый бит младшей части является значащим битом двоичного кода результата.

Если вы посмотрите описание команды imul, то увидите, что она допускает более широкие возможности по заданию местоположения операндов. Это сделано для удобства использования.

Деление чисел без знака. Для деления чисел без знака предназначена команда:

div делитель

Делитель может находиться в памяти или в регистре и иметь размер 8, 16 или 32 бит. Местонахождение делимого фиксировано и так же, как в команде умножения, зависит от размера операндов. Результатом команды деления являются значения частного и остатка.

Варианты местоположения и размеров операндов операции деления показаны в таблице 5.

Таблица 5. Расположение операндов и результата при делении

Делимое	**Делитель**	**Частное**	**Остаток**
16 бит в регистре ax	Байт регистр или ячейка памяти	Байт в регистре al	Байт в регистре ah
32 бит dx – старшая часть ax – младшая часть	Слово 16 бит регистр или ячейка памяти	Слово 16 бит в регистре ax	Слово 16 бит в регистре dx
64 бит edx – старшая часть eax – младшая часть	Двойное слово 32 бит регистр или ячейка памяти	Двойное слово 32 бит в регистре eax	Двойное слово 32 бит в регистре edx

После выполнения команды деления содержимое флагов неопределено, но возможно возникновение прерывания с номером 0, называемого “деление на ноль”. Этот вид прерывания относится к так

называемым исключениям. Эта разновидность прерываний возникает внутри микропроцессора из-за некоторых аномалий во время вычислительного процесса. Прерывание 0, “деление на ноль”, при выполнении команды div может возникнуть по одной из следующих причин:

- делитель равен нулю;
- частное не входит в отведенную под него разрядную сетку, что может случиться в следующих случаях:
- при делении делимого величиной в слово на делитель величиной в байт, причем значение делимого в более чем 256 раз больше значения делителя;
- при делении делимого величиной в двойное слово на делитель величиной в слово, причем значение делимого в более чем 65 536 раз больше значения делителя;
- при делении делимого величиной в учетверенное слово на делитель величиной в двойное слово, причем значение делимого в более чем 4 294 967 296 раз больше значения делителя.

Деление чисел со знаком. Для деления чисел со знаком предназначена команда

idiv делитель

Для этой команды справедливы все рассмотренные положения, касающиеся команд и чисел со знаком. Отметим лишь особенности возникновения исключения 0, “деление на ноль”, в случае чисел со знаком. Оно возникает при выполнении команды idiv по одной из следующих причин:

- делитель равен нулю;
- частное не входит в отведенную для него разрядную сетку. Последнее в свою очередь может произойти:
- при делении делимого величиной в слово со знаком на делитель величиной в байт со знаком, причем значение делимого в более чем 128 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от –128 до +127);
- при делении делимого величиной в двойное слово со знаком на делитель величиной в слово со знаком, причем значение делимого в более чем 32 768 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от –32 768 до +32 768);
- при делении делимого величиной в учетверенное слово со знаком на делитель величиной в двойное слово со знаком, причем значе-

ние делимого в более чем 2 147 483 648 раз больше значения делителя (таким образом, частное не должно находиться вне диапазона от –2 147 483 648 до +2 147 483 647).

3.6. Логические команды

В системе команд микропроцессора есть следующий набор команд, поддерживающих работу с логическими данными:

and операнд_1,операнд_2 – операция логического умножения. Команда выполняет поразрядно логическую операцию И (конъюнкцию) над битами операндов операнд_1 и операнд_2. Результат записывается на место операнд_1.

or операнд_1,операнд_2 – операция логического сложения. Команда выполняет поразрядно логическую операцию ИЛИ (дизъюнкцию) над битами операндов операнд_1 и операнд_2. Результат записывается на место операнд_1.

xor операнд_1,операнд_2 – операция логического исключающего сложения. Команда выполняет поразрядно логическую операцию исключающего ИЛИ над битами операндов операнд_1 и операнд_2. Результат записывается на место операнд_1.

test операнд_1,операнд_2 – операция “проверить” (способом логического умножения). Команда выполняет поразрядно логическую операцию И над битами операндов операнд_1 и операнд_2. Состояние операндов остается прежним, изменяются только флаги zf, sf, и pf, что дает возможность анализировать состояние отдельных битов операнда без изменения их состояния.

not операнд – операция логического отрицания. Команда выполняет поразрядное инвертирование (замену значения на обратное) каждого бита операнда. Результат записывается на место операнда.

3.7. Команды сдвига

Команды этой группы также обеспечивают манипуляции над отдельными битами операндов, но иным способом, чем логические команды, рассмотренные выше. Все команды сдвига перемещают биты в поле операнда влево или вправо в зависимости от кода операции.

Количество сдвигаемых разрядов – счетчик_сдвигов – располагается, как видите, на месте второго операнда и может задаваться двумя способами:

- статически, что предполагает задание фиксированного значения с помощью непосредственного операнда;
- динамически, что означает занесение значения счетчика сдвигов в регистр cl перед выполнением команды сдвига.

Команды линейного сдвига делятся на два подтипа:

- команды логического линейного сдвига;
- команды арифметического линейного сдвига.

К командам логического линейного сдвига относятся:

- shl операнд,счетчик_сдвигов (Shift Logical Left) – логический сдвиг влево. Содержимое операнда сдвигается влево на количество битов, определяемое значением счетчик_сдвигов. Справа (в позицию младшего бита) вписываются нули;
- shr операнд,счетчик_сдвигов (Shift Logical Right) – логический сдвиг вправо. Содержимое операнда сдвигается вправо на количество битов, определяемое значением счетчик_сдвигов. Слева (в позицию старшего, знакового бита) вписываются нули.

Команды арифметического линейного сдвига отличаются от команд логического сдвига тем, что они особым образом работают со знаковым разрядом операнда:

- sal операнд,счетчик_сдвигов (Shift Arithmetic Left) – арифметический сдвиг влево. Содержимое операнда сдвигается влево на количество битов, определяемое значением счетчик_сдвигов. Справа (в позицию младшего бита) вписываются нули. Команда sal не сохраняет знака, но устанавливает флаг cf в случае смены знака очередным выдвигаемым битом. В остальном команда sal полностью аналогична команде shl;
- sar операнд,счетчик_сдвигов (Shift Arithmetic Right) – арифметический сдвиг вправо. Содержимое операнда сдвигается вправо на количество битов, определяемое значением счетчик_сдвигов. Слева в операнд вписываются нули. Команда sar сохраняет знак, восстанавливая его после сдвига каждого очередного бита.

К командам циклического сдвига относятся команды, сохраняющие значения сдвигаемых бит. Есть два типа команд циклического сдвига:

- команды простого циклического сдвига;
- команды циклического сдвига через флаг переноса cf.

К командам простого циклического сдвига относятся:

- rol операнд,счетчик_сдвигов (Rotate Left) – циклический сдвиг влево. Содержимое операнда сдвигается влево на количество бит,

определяемое операндом счетчик_сдвигов. Сдвигаемые влево биты записываются в тот же операнд справа.

- ror операнд,счетчик_сдвигов (Rotate Right) – циклический сдвиг вправо. Содержимое операнда сдвигается вправо на количество бит, определяемое операндом счетчик_сдвигов. Сдвигаемые вправо биты записываются в тот же операнд

К командам циклического сдвига через флаг переноса cf относятся следующие:

- rcl операнд,счетчик_сдвигов (Rotate through Carry Left) – циклический сдвиг влево через перенос. Содержимое операнда сдвигается влево на количество бит, определяемое операндом счетчик_сдвигов. Сдвигаемые биты поочередно становятся значением флага переноса cf.
- rcr операнд,счетчик_сдвигов (Rotate through Carry Right) – циклический сдвиг вправо через перенос. Содержимое операнда сдвигается вправо на количество бит, определяемое операндом счетчик_сдвигов. Сдвигаемые биты поочередно становятся значением флага переноса cf.

3.8. Процедуры на языке ассемблера

Для оформления процедур существуют специальные директивы proc/endp и машинная команда ret – возврат управления из процедуры вызывающей программе.

Формат команды ret: ret число.

Работа команды зависит от типа процедуры:

- для процедур ближнего типа – восстановить из стека содержимое eip/ip;
- для процедур дальнего типа – последовательно восстановить из стека содержимое eip/ip и сегментного регистра cs.
- если команда ret имеет операнд, то увеличить содержимое esp/sp на величину операнда число; при этом учитывается атрибут режима адресации – use16 или use32:
  - если use16, то sp=(sp+число), то есть указатель стека сдвигается на число байт, равное значению число;
  - если use32, то sp=(sp+2*число), то есть указатель стека сдвигается на число слов, равное значению число.

Процедуры могут размещаться в программе:

- в начале программы (до первой исполняемой команды);

- в конце программы (после команды выхода в операционную систему);
- промежуточный вариант – тело процедуры располагается внутри другой процедуры или основной программы (с использованием команды безусловного перехода jmp);
- в другом модуле.

Вызов близкой или дальней процедуры с запоминанием в стеке адреса точки возврата осуществляется *командой call* (формат: call метка). Выполнение команды не влияет на флаги.

При ближней адресации – в стек заносится содержимое указателя команд eip/ip и в этот же регистр загружается новое значение адреса, соответствующее метке;

При дальней адресации – в стек заносится содержимое указателя команд eip/ip и cs. Затем в эти же регистры загружаются новые значения адресов, соответствующие дальней метке;

3.9. Передача аргументов через регистры

Существуют следующие варианты передачи аргументов в процедуру (модуль):

- *Через регистры*. Данные становятся доступными немедленно после передачи управления процедуре. Очень популярный способ при небольшом объеме передаваемых данных.
- *Через общую область памяти*. Необходимо использовать атрибут комбинирования сегментов – common (см. лаб. работу 1). Сегменты будут перекрываться в памяти и, следовательно, совместно использовать выделенную память.
- *Через стек*. Наиболее часто используемый способ. Суть его в том, что вызывающая процедура самостоятельно заносит в стек передаваемые данные, после чего производит вызов вызываемой процедуры.
- *С помощью директив extrn и public*. Директива extrn предназначена для объявления некоторого имени внешним по отношению к данному модулю. Это имя в другом модуле должно быть объявлено в директиве public. Директива public предназначена для объявления некоторого имени, определенного в этом модуле и видимого в других модулях.

  Синтаксис директив:

```
Extrn имя:тип, …, имя:тип
Public имя, …, имя
```

Здесь имя – идентификатор, определенный в другом модуле. В качестве идентификатора могут выступать:

- имена переменных (определенных операторами db, dw и т.д.);
- имена процедур;
- имена констант (определенных операторами = и equ).

Возможные значения типа определяются допустимыми типами объектов для этих директив:

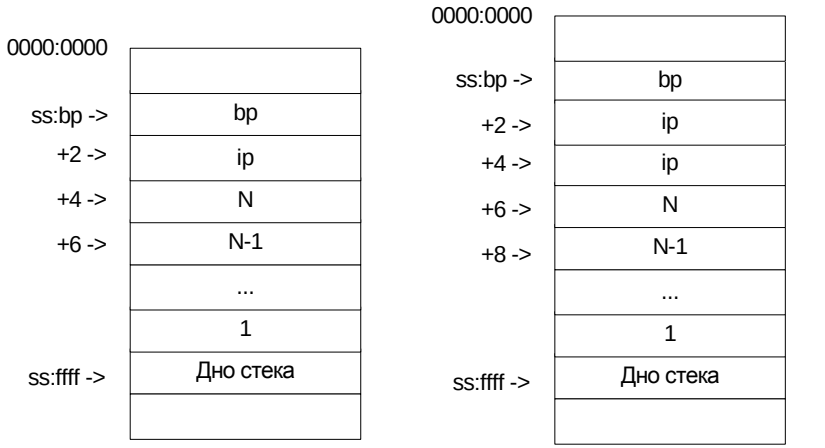
- если имя – это переменная, то тип может принимать значения byte, word, dword, pword, fword, qword и tbyte;
- если имя – это процедура, то тип может принимать значения near или far;
- если имя – это константа, то тип должен быть abs.

Остановимся боле подробно при передаче параметров через стек. Приведем пример структуры вызываемой процедуры при ближней адресации.

```
asmpr proc   near
;пролог
 push  bp
 mov   bp,sp
. . .
;доступ к элементам стека
 mov ax, [bp+4] ;доступ к N-элементу
 mov ax, [bp+6] ;доступ к N-1-элементу
 mov ax, [bp+8] ;доступ к N-2-элементу
. . .
; эпилог
 mov   sp,bp ;восстановление sp
 pop   bp ;восстановление bp

 ret    ;возврат в вызывающую программу
asmpr endp   ;конец процедуры
```

На рис. 1. показана структура стека при ближней (а) и дальней (б) адресации в нутрии вызываемой процедуры.



а) б)

Рис. 1. Структура стека в вызываемой процедуре при использовании различных видов адресации: а – ближняя адресация; б – дальняя адресация.

При использовании дальней адресации для доступа к элементам стека требуется поправка на +2.

```
;доступ к элементам стека при дальней адреса-
ции
 mov ax, [bp+6] ;доступ к N-элементу
 mov ax, [bp+8] ;доступ к N-1-элементу
 mov ax, [bp+10] ;доступ к N-2-элементу
```

3.10. Возврат результата из процедуры

Существует три варианта возврата результата из процедуры:

- С использованием регистров.
- С использованием общей памяти.
- С использованием стека. Здесь возможны два варианта:
  - Использование для возвращаемых аргументов тех же ячеек в стеке, которые применялись для передачи аргументов в процедуру.
  - Предварительное помещение в стек наряду с передаваемыми аргументами фиктивных аргументов с целью резервирования места для возвращаемого значения.

3.11. Макросредства языка ассемблера

3.11.1. Псевдооператоры equ и =

К простейшим макросредствам языка ассемблера можно отнести псевдооператоры equ и "=" (равно). Их мы уже неоднократно использовали при написании программ. Эти псевдооператоры предназначены для присвоения некоторому выражению символического имени или идентификатора. Впоследствии, когда в ходе трансляции этот идентификатор встретится в теле программы, макроассемблер подставит вместо него соответствующее выражение. В качестве выражения могут быть использованы константы, имена меток, символические имена и строки в апострофах. После присвоения этим конструкциям символического имени его можно использовать везде, где требуется размещение данной конструкции.

Синтаксис псевдооператора equ:

имя_идентификатора equ строка или числовое_выражение

Синтаксис псевдооператора “=”:

имя_идентификатора = числовое_выражение

Несмотря на внешнее и функциональное сходство псевдооператоры equ и “=” отличаются следующим:

- из синтаксического описания видно, что с помощью equ идентификатору можно ставить в соответствие, как числовые выражения, так и текстовые строки, а псевдооператор “=” может использоваться только с числовыми выражениями;
- идентификаторы, определенные с помощью “=”, можно переопределять в исходном тексте программы, а определенные с использованием equ – нельзя.

Ассемблер всегда пытается вычислить значение строки, воспринимая ее как выражение. Для того, чтобы строка воспринималась именно как текстовая, необходимо заключить ее в угловые скобки: <строка>.

Кстати сказать, угловые скобки являются оператором ассемблера, с помощью которого транслятору сообщается, что заключенная в них строка должна трактоваться как текст, даже если в нее входят служебные слова ассемблера или операторы. Хотя в режиме Ideal это не обязательно, так как строка для equ в нем всегда трактуется как текстовая.

Псевдооператор **equ** удобно использовать для настройки программы на конкретные условия выполнения, замены сложных в обо-

значении объектов, многократно используемых в программе более простыми именами и т. п.

Псевдооператор “=” удобно использовать для определения простых абсолютных (то есть независящих от места загрузки программы в память) математических выражений. Главное условие то, чтобы транслятор мог вычислить эти выражения во время трансляции.

3.11.2. Макрокоманды

Идейно макрокоманда представляет собой дальнейшее развитие механизма замены текста. С помощью макрокоманд в текст программы можно вставлять последовательности строк (которые логически могут быть данными или командами) и даже более того – привязывать их к контексту места вставки.

Макрокоманда представляет собой строку, содержащую некоторое символическое имя – имя макрокоманды, предназначенное для того, чтобы быть замещенной одной или несколькими другими строками. Имя макрокоманды может сопровождаться параметрами.

Обычно программист сам чувствует момент, когда ему нужно использовать макрокоманды в своей программе. Если такая необходимость возникает, и нет готового ранее разработанного варианта нужной макрокоманды, то вначале необходимо задать ее шаблон-описание, который называют макроопределением.

Синтаксис макроопределения следующий:

```
имя_макрокоманды macro список_аргументов
 тело макроопределения
endm
```

Есть три варианта размещения макроопределения:

- В начале исходного текста программы до сегмента кода и данных с тем, чтобы не ухудшать читабельность программы. Этот вариант следует применять в случаях, если определяемые вами макрокоманды актуальны только в пределах одной этой программы.
- В отдельном файле. Этот вариант подходит при работе над несколькими программами одной проблемной области. Чтобы сделать доступными эти макроопределения в конкретной программе, необходимо в начале исходного текста этой программы записать директиву include имя_файла.

- В макробиблиотеке. Если у вас есть универсальные макрокоманды, которые используются практически во всех ваших программах, то их целесообразно записать в так называемую макробиблиотеку. Сделать актуальными макрокоманды из этой библиотеки можно с помощью все той же директивы include.

Если в программе некоторая макрокоманда вызывается несколько раз, то в процессе макрогенерации возникнет ситуация, когда в программе один идентификатор будет определен несколько раз, что, естественно, будет распознано транслятором как ошибка. Для выхода из подобной ситуации применяют директиву **local**, которая имеет следующий синтаксис: local список_идентификаторов.

3.12. Задание на выполнение

Данная работа рассчитана на 2 лабораторные работы (8 часов). Выполните задания в соответствии с вашим вариантом. И подготовиться к ответам на теоретическую часть лабораторной работы.

Вариант 1

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=A+B.
2. Установить все четные биты C.
3. Вывести на экран число C и все промежуточные результаты в двоичном виде.
4. Написать процедуру для вывода результата
5. Написать процедуру для ввода чисел
6. Написать макрос для расчета
7. Передача параметров через регистры
8. Используйте стандартные директивы сегментации и формат exe-программы.

Вариант 2

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=A+B.
2. Обнулить все нечетные биты C.
3. Вывести на экран число C и все промежуточные результаты в двоичном виде.
4. Написать процедуру для вывода результата
5. Написать макрос для ввода чисел

6. Написать макрос для расчета

7. Передача параметров через стек

8. Используйте упрощенные директивы сегментации и формат exe-программы.

Вариант 3

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=A+B*2.

2. Если третий бит числа C установлен, то вывести на экран C в двоичном виде, в противном случае, вывести на экран C/2 в двоичном виде.

3. Написать макрос для вывода результата

4. Написать процедуру для ввода чисел

5. Написать макрос для расчета

6. Передача параметров через стек

7. Используйте стандартные директивы сегментации и формат com-программы.

Вариант 4

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=A/2+B.

2. Установить все нечетные биты C.

3. Вывести на экран число C и все промежуточные результаты в двоичном виде.

4. Написать макрос для вывода результата

5. Написать макрос для ввода чисел

6. Написать процедуру для расчета

7. Передача параметров через стек

8. Используйте упрощенные директивы сегментации и формат com-программы.

Вариант 5

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=(A+B)/4.

2. Сбросить пятый бит числа C, если он установлен.

3. Вывести на экран число C и все промежуточные результаты в двоичном виде.

4. Написать процедуру для вывода результата
5. Написать процедуру для ввода чисел
6. Написать макрос для расчета
7. Передача параметров через стек
8. Используйте стандартные директивы сегментации и формат exe-программы.

Вариант 6

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=(A-B)*4.
2. Выполнить циклический сдвиг полученного числа C на 3 бита вправо.
3. Вывести на экран число C и все промежуточные результаты в двоичном виде.
4. Написать процедуру для вывода результата
5. Написать макрос для ввода чисел
6. Написать процедуру для расчета
7. Передача параметров через стек
8. Используйте упрощенные директивы сегментации и формат exe-программы.

Вариант 7

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A/2+B.
2. Выполнить арифметический сдвиг C на 3 бит влево.
3. Вывести на экран число C и все промежуточные результаты в двоичном виде.
4. Написать макрос для вывода результата
5. Написать процедуру для ввода чисел
6. Написать процедуру для расчета
7. Передача параметров через стек
8. Используйте стандартные директивы сегментации и формат com-программы.

Вариант 8

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A+B*2.
2. Обнулить все четные биты C.

3. Вывести на экран число C и все промежуточные результаты в двоичном виде.

4. Написать процедуру для вывода результата

5. Написать процедуру для ввода чисел

6. Написать процедуру для расчета

7. Передача параметров через стек

8. Используйте упрощенные директивы сегментации и формат com-программы.

Вариант 9

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A+(B-5h)*2.

2. Если установлен четвертый бит числа C то вывести на экран A в десятичном виде, в противном случае вывести на экран число B в десятичном виде.

3. Написать макрос для вывода результата

4. Написать макрос для ввода чисел

5. Написать макрос для расчета

6. Передача параметров через общую память

7. Используйте стандартные директивы сегментации и формат exe-программы.

Вариант 10

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A+12h)/2+B.

2. Обнулить все четные биты C.

3. Вывести на экран число C и все промежуточные результаты в двоичном виде.

4. Написать процедуру для вывода результата

5. Написать макрос для ввода чисел

6. Написать макрос для расчета

7. Передача параметров через общую память

8. Используйте упрощенные директивы сегментации и формат exe-программы.

Вариант 11

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A-14h)*4-B.

2. Установить все четные биты C.

3. Вывести на экран число C и все промежуточные результаты в десятичном виде.

4. Написать макрос для вывода результата

5. Написать процедуру для ввода чисел

6. Написать макрос для расчета

7. Передача параметров через общую память

8. Используйте стандартные директивы сегментации и формат com-программы.

Вариант 12

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A*B-4.

2. Если третий и пятый бит числа C установлены, вывести на экран A в двоичном виде, если третий и пятый бит числа C сброшены, вывести на экран B в двоичном виде, в других случаях вывести на экран число C в двоичном виде.

3. Написать макрос для вывода результата

4. Написать макрос для ввода чисел

5. Написать процедуру для расчета

6. Передача параметров через общую память

7. Используйте упрощенные директивы сегментации и формат com-программы.

Вариант 13

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A+B)/4-16.

2. Если третий и пятый бит числа C установлены, вывести на экран A в двоичном виде, если третий и пятый бит числа C сброшены, вывести на экран B в двоичном виде, в других случаях вывести на экран число C в двоичном виде.

3. Написать процедуру для вывода результата

4. Написать процедуру для ввода чисел

5. Написать макрос для расчета

6. Передача параметров через общую память

7. Используйте стандартные директивы сегментации и формат exe-программы.

Вариант 14

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A-B)*2+1Ah.

2. Вывести на экран 1 если шестой бит числа C установлен и 0 в противном случае.

3. Написать макрос для вывода результата

4. Написать процедуру для ввода чисел

5. Написать процедуру для расчета

6. Передача параметров через общую память

7. Используйте упрощенные директивы сегментации и формат exe-программы.

Вариант 15

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=(A-14h)*4-B.

2. Сбросить третий бит числа C, если он установлен.

3. Вывести на экран число C и все промежуточные результаты в десятичном виде.

4. Написать процедуру для вывода результата

5. Написать макрос для ввода чисел

6. Написать процедуру для расчета

7. Передача параметров через общую память

8. Используйте стандартные директивы сегментации и формат com-программы.

Вариант 16

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=A+4*B.

2. Выполнить циклический сдвиг числа C на 3 бита вправо.

3. Вывести на экран число C и все промежуточные результаты в десятичном виде.

4. Написать процедуру для вывода результата

5. Написать процедуру для ввода чисел

6. Написать процедуру для расчета

7. Передача параметров через общую память

8. Используйте упрощенные директивы сегментации и формат com-программы.

Вариант 17

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=A*3+B*2.

2. Выполнить арифметический сдвиг числа C на 2 бита вправо.

3. Вывести на экран число C и все промежуточные результаты в десятичном виде.

4. Написать макрос для вывода результата

5. Написать макрос для ввода чисел

6. Написать макрос для расчета

7. Передача параметров через регистры

8. Используйте стандартные директивы сегментации и формат exe-программы.

Вариант 18

Пользователь вводит два числа A и B в десятичном виде. Программа должна:

1. Посчитать C=A*3+B*2.

2. Выполнить арифметический сдвиг числа C на 2 бита вправо.

3. Вывести на экран число C и все промежуточные результаты в шестнадцатеричном виде.

4. Написать процедуру для вывода результата

5. Написать макрос для ввода чисел

6. Написать макрос для расчета

7. Передача параметров через регистры

8. Используйте упрощенные директивы сегментации и формат exe-программы.

Вариант 19

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=A*B-4.

2. Если третий и пятый бит числа C установлены, вывести на экран A в десятичном виде, если третий и пятый бит числа C сброшены, вывести на экран B в десятичном виде, в других случаях вывести на экран число C в десятичном виде.

3. Написать макрос для вывода результата

4. Написать процедуру для ввода чисел

5. Написать макрос для расчета

6. Передача параметров через регистры

7. Используйте стандартные директивы сегментации и формат com-программы.

Вариант 20

Пользователь вводит два числа A и B в шестнадцатеричном виде. Программа должна:

1. Посчитать C=(A+17h)/2+B.

2. Обнулить все четные биты C.

3. Вывести на экран число C и все промежуточные результаты в десятичном виде.

4. Написать макрос для вывода результата

5. Написать макрос для ввода чисел

6. Написать процедуру для расчета

7. Передача параметров через регистры

8. Используйте упрощенные директивы сегментации и формат com-программы.

Часть 3. Методические указания к самостоятельной работе

1. Проработка лекционного материала из расчета 0,5 час на 1 час лекции (**17 часа самостоятельной работы**).

Семестр 2.

1. Основные понятия и концепции построения операционных систем. Классификация программных средств. Место и функции системного программного обеспечения. Вычислительный процесс. Управление ресурсами. Потоки, нити, треды. Прерывания. Классификация операционных систем – 2 часа самостоятельной работы.

2. Архитектуры операционных систем и интерфейсы прикладного программирования. Основные принципы построения ОС. Микроядерные ОС. Монолитные ОС. Принципы построения интерфейсов ОС. Интерфейс прикладного программирования. Функции API на различных уровнях реализации. Платформенно-независимый интерфейс POSIX – 2 часа самостоятельной работы.

3. Управление памятью. Основные понятия. Фон-неймановская архитектура вычислительных машин. Адресация ячеек памяти в реальном режиме. Подсистемы памяти и хранения данных. Стек. Распределение оперативной памяти. Организация режима защиты. Переключение задач и виртуальные машины. Защищенный режим и виртуальная память. Адресация ячеек памяти в защищенном режиме. Переключение между реальном и защищенном режимами. Кэширование памяти – 2 часа самостоятельной работы.

4. Управление внешней памятью. Общие свойства. Основные характеристики устройств внешней памяти. Характеристики накопителей на магнитных дисках. Структура магнитного диска. Функции файловой системы. Файловые системы FAT, NTFS, HPFS, Unix, CD-ROM, DVD-ROM – 2 часа самостоятельной работы.

Итого часов самостоятельной работы во втором семестре первого курса – 8.

Семестр 3.

6. Операционные системы фирмы Microsoft. История ОС от фирмы Microsoft. MS-DOS. Две ветви Windows: Windows 9x, Windows

NT. Windows 95 (98, Millennium): основные характеристики, функции, состав, загрузка, файлы конфигурации. Технология Plug and Play (2 часа). Windows NT (2000, XP, 2003): основные характеристики, функции, состав, загрузка, файлы конфигурации – 2 часа.

7. Семейство ОС OS/2 Warp. Общее представление. Особенности архитектуры. Особенности интерфейса. Серверная ОС OS/2 Warp 4.5. Операционная система eComStation – 1 час.

8. Операционные системы семейства Unix. Система Unix. Состав. Основные свойства. Языки программирования в Unix. Версии Unix. Начало и конец сеанса работы. Формат команд. Основные простые команды. Руководство UNIX Reference Manual. Работа с каталогами и файлами. Команды и стандартные файлы. Перенаправления потоков. Фильтры. Режимы переднего и заднего плана. Программирование на языке Shell. Версии Shell. Процедуры и переменные. Структурные операторы и операторы цикла. Отладка процедур. C-Shell. Переменные и метасимволы. Командные файлы. Управляющие структуры. Связь пользователь-пользователь. Работа с текстовыми файлами. Текстовый редактор AWK. Руководство системного администратора. Спецпользователи. Работа с пользователями. Работа с файловой системой. Работа с устройствами. Подключение терминалов. Периодическое выполнение заданий. Управление операционной системой. Реконфигурация ОС. Практические советы – 3 часа.

9. Операционные системы реального времени, на примере ОС QNX. Операционные системы реального времени. Общее представление об ОС QNX. Особенности архитектуры. Основные механизмы – 1 час.

10. Программная модель микропроцессора Intel. Основные регистры. Структура памяти. Управление ресурсами с использованием низкоуровневых языков программирования. Язык Ассемблер. Структура программы на языке Ассемблер, основные команды и конструкции – 2 часов.

Итого часов самостоятельной работы в третьем семестре второго курса – 9.

Итого часов самостоятельной работы - 17 часов.

2. Подготовка к лабораторным работам и оформление отчетов по ЛР из расчета 1 час на 1 час ЛР (**34 часа самостоятельной работы**).
Для подготовки к лабораторным работам следует использовать данные методические указания (части 1 и 2), в качестве дополнительной литературы следует воспользоваться литературой [1-7], приведенной в разделе «Список литературы»
Форма контроля: Допуск к лаб. работам. Защита отчета по ЛР.

3. Изучение дополнительного материала (**16 часов самостоятельной работы**): Программирование на ассемблере в реальном и защищенном режиме.

Для изучения дополнительного материала следует воспользоваться учебным пособием [7] и дополнительным поиском ресурсов в сети Интернет.
Форма контроля: Проверка конспектов самостоятельного изучения тем.

4. Подготовка к экзамену – 36 часов.

Итого самостоятельной работы – 103 часа.

СПИСОК ЛИТЕРАТУРЫ

1. Гриценко Ю. Б. Операционные системы: учебное пособие для вузов / Томск: ТУСУР, 2004. - 243[1] с.
2. Гриценко Ю. Б. Операционные среды, системы и оболочки: учебное пособие / Томск: ТУСУР, 2005. - 281 с.
3. Гриценко Ю.Б. Операционные системы: Учебное пособие. В 2-х частях. – Томск: Томский межвузовский центр дистанционного образования, 2009. – Ч.1. – 187с.
4. Гриценко Ю.Б. Операционные системы: Учебное пособие. В 2-х частях. – Томск: Томский межвузовский центр дистанционного образования, 2009. – Ч.2. – 230с.
5. Гордеев А.В. Операционные системы: Учебник для вузов / СПб.: Питер, 2004. - 415[1] с.
6. Робачевский А. М.. Операционная система UNIX: Учебное пособие для вузов / СПб.: BHV – Санкт-Петербург, 2002. – 514с.
7. Юров В. Assembler / СПб.: Питер, 2003. – 640с. ил.