

Машинно-зависимые языки программирования

Лабораторная работа №1

“Работа с файлами. Командная строка”

Теоретическая и справочная информация

Физический диск - отдельное устройство для долговременного хранения данных. Большинство устройств делится на физические единицы - **сектора** (блоки размером 512 байт). Для упрощения адресации сектора могут объединяться в **кластеры**, например, по 8 секторов (4 КБ).

Логический диск - концепция организации пространства на физическом диске в ОС DOS, Windows. Обозначаются буквами латинского алфавита. Со времён эпохи дискет и MS-DOS принято, что A: - дисковод 5,25", B: - дисковод 3,5", C: - системный логический диск (раздел жёсткого диска, содержащий операционную систему), далее - произвольно. В Windows буквы дисков можно переназначать с помощью системной программы “Управление дисками” (доступна через контекстное меню кнопки Пуск или значка “Мой компьютер” при наличии прав администратора).

Файл - поименованное место на диске. Содержимое файла располагается в кластерах логических дисков (не обязательно последовательно) так, что один кластер может принадлежать только одному файлу. Например, при размере кластеров на диске 4096 байт и размере файла 10 байт он займёт 1 кластер, при размере 4097 байт - 2 кластера, и т. д.

Каталог - файл специального типа, содержимым которого является список файлов и других каталогов, вложенных в него.

Именованние файлов. В ОС DOS и Windows имена файлов являются регистронезависимыми, то есть abc и ABC - считается одним и тем же именем. При этом в Windows имя хранится в заданном при создании файла регистре, а в DOS - переводится в заглавные буквы. Длина имени DOS ограничена 11 символами: 8 - имя и 3 - расширение. Более длинные имена для совместимости обрезаются до 8 символов либо, если сокращённое имя уже занято, - до меньшего количества, а хвостовые символы заменяются на ~ и уникальное число (1, 2, 3 и т. д.)

Файловая система — способ организации, хранения и именования данных на диске (в первую очередь - распределения файлов по кластерам).

Главный “файл” любого логического диска - **корневой каталог**. Расположен по фиксированному адресу и не может быть удалён. Все прочие файлы и каталоги могут быть удалены, переименованы, перемещены.

Любой файл или каталог идентифицируется **путём** к нему.

Текущий каталог - путь к каталогу, который является “рабочим” в настоящий момент. С каждой запущенной программой связан свой текущий каталог.

Текущий диск - логический диск, на котором расположен текущий каталог.

Практическое задание по Проводнику Windows

1. Создайте текстовый файл двумя способами.
2. Скопируйте файл средствами Проводника 4 различными способами.
3. Переименуйте файл двумя различными способами.
4. Переместите файл двумя различными способами.
5. Удалите файл 3 различными способами.

Теоретическая и справочная информация по командной строке

Путь к файлу может быть **относительным** - считается от текущего каталога и начинается с любого символа, кроме \, и **абсолютным**. Разделителем каталогов в пути служит символ \.

Стандартный путь DOS может состоять из трех компонентов:

- Буква тома или диска, после которой следует разделитель томов (:).
- Имя каталога. [Символ разделителя каталогов](#) служит для разделения подкаталогов во внутренней иерархии каталога.
- Необязательное имя файла. [Символ разделителя каталогов](#) служит для разделения пути к файлу и его имени.

Если присутствуют все три компонента, путь является абсолютным. Если буква тома или диска не указана и имя каталога начинается с [символа разделителя каталогов](#), такой путь задан относительно корня текущего диска. В противном случае путь задан относительно текущего каталога.

(<https://docs.microsoft.com/ru-ru/dotnet/standard/io/file-path-formats>)

Оболочка операционной системы - интерпретатор команд операционной системы, обеспечивающий интерфейс для взаимодействия пользователя с функциями системы.

Командный интерпретатор, интерпретатор командной строки — программа, являющаяся частью операционной системы, обеспечивающая базовые возможности управления компьютером посредством интерактивного ввода команд через интерфейс командной строки или последовательного исполнения пакетных командных файлов.

В MS-DOS и старых версиях Windows интерпретатор - COMMAND.COM, в семействе Windows NT (NT, 2000, XP и далее) - cmd.exe.

Основные возможности интерпретатора командной строки - просмотр каталогов, операции манипулирования файлами, запуск программ.

Для запуска интерпретатора вызовите cmd.exe через меню Пуск.

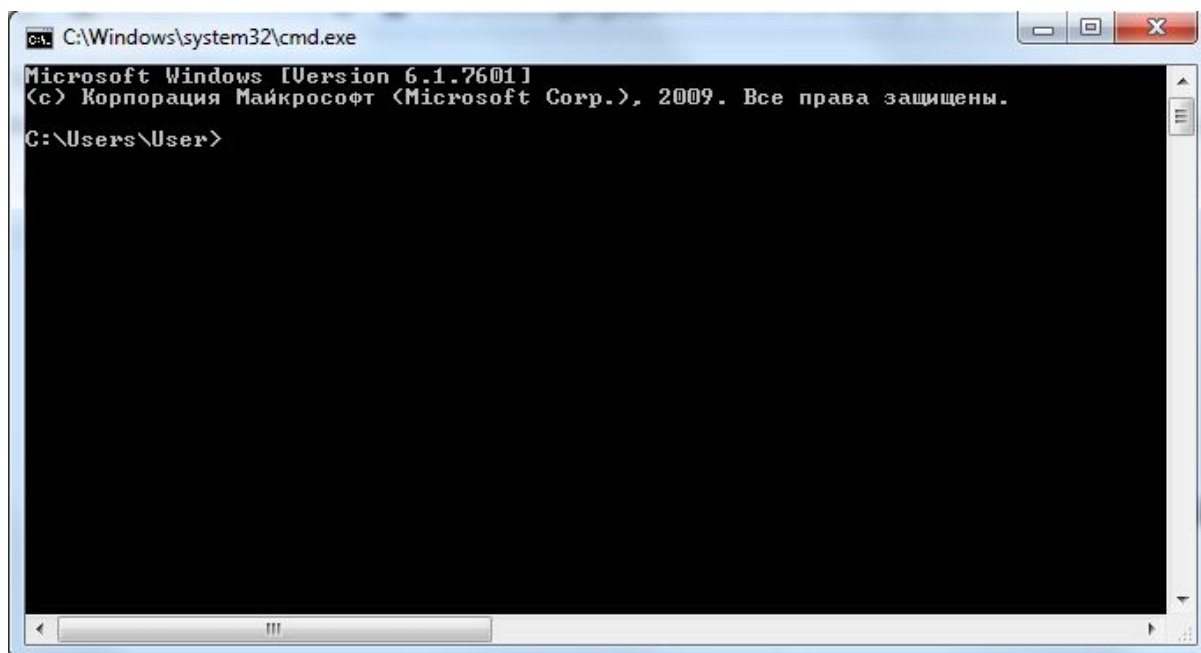


Рис. 1. Примерный вид окна интерпретатора командной строки в Windows 7.

В начале строки командной оболочки находится приглашение ввода. По умолчанию там указывается текущий каталог. Приглашение заканчивается символом >, за которым расположен курсор.

Курсор показывает положение на экране, в которое будут попадать символы, вводимые с клавиатуры, а также сведения, выводимые программой.

Команды интерпретатора делятся на внутренние и внешние. **Внутренние** распознаются и выполняются непосредственно самим командным интерпретатором. **Внешние** команды являются отдельными программами - исполняемыми файлами.

Путь "." - текущий каталог, ".." - родительский каталог.

Структура команды. Ключи. Параметры.

Первое и обязательное слово при вызове команды - это её имя. Далее могут идти ключи, начинаются со знака "/". Далее - параметры.

Практически у любой команды доступен ключ /?, позволяющий получить справку по её назначению и способам вызова.

Стандартные потоки ввода и вывода, перенаправление ввода-вывода.

С каждой программой при запуске связываются так называемые стандартные потоки ввода-вывода. По умолчанию стандартный поток ввода - данные, поступающие с клавиатуры, а стандартный поток вывода - монитор (в современных системах - окно командного интерпретатора).

DOS и Windows позволяют перенаправить поток вывода в файл с помощью символа ">". Пример: echo test output redirect > 123.txt

Существуют специальные файлы: con - клавиатура, prn - принтер, и ряд других. Таким образом, в терминале доступна возможность создания файла с клавиатуры. Для этого используется команда копирования файлов сору, но первым параметром вместо имени файла указывается имя устройства клавиатуры con. Пример: sory con 456.txt

Эта команда будет осуществлять запись в указанный файл всех вводимых символов до завершения ввода с помощью комбинации Ctrl+Z.

Практическое задание по командной оболочке

Список команд для самостоятельного изучения:

- help
- echo
- cls
- exit
- dir
- cd
- copy
- move
- ren
- del
- md
- rd
- more
- path
- type
- prompt
- set
- fc
- find
- title
- ver

С помощью этих команд переместитесь в директорию, предназначенную для создания пользовательских файлов. Создайте структуру каталогов с 3 уровнями вложенности, 3 новых различных файла и опробуйте возможности переименования и перемещения с использованием относительных и абсолютных путей.

Изучите возможности автодополнения путей с помощью клавиши Tab.

Замените приглашение ввода на строку формата “(время) текущие диск и путь>”.

Кроме команд самой оболочки, интерпретатор командной строки позволяет запустить любую программу, доступную в системе. Можно указать путь к программе, но также можно запустить программу без указания пути, если её местоположение хранится в системной переменной PATH. Эту переменную можно модифицировать, например, так: **set PATH=%PATH%;C:\your\path\here**

Контрольные вопросы

1. Вопросы по теоретическим сведениям.
2. Как узнать содержимое переменной PATH?
3. Как сформировать файл 3.txt, объединив в нём содержимое файлов 1.txt и 2.txt?
4. Зачем нужна команда more?
5. Чем отличаются переименование и перемещение?
6. Как вызываемая программа различает ключи и параметры? Как задать путь к файлу, если он содержит пробел?

Машинно-зависимые языки программирования

Лабораторная работа №2

“Создание простейшей программы на ассемблере. Отладчик”

Теоретическая и справочная информация

Язык ассемблера — машинно-ориентированный язык программирования низкого уровня. Его команды прямо соответствуют отдельным командам процессора.

Исполняемый файл — файл, содержащий программу в виде, в котором она может быть исполнена компьютером. Перед исполнением программа загружается в память, и выполняются некоторые подготовительные операции (настройка окружения, загрузка библиотек). Основным формат исполняемых файлов в Windows - .EXE.

Компилятор — это специальная программа, которая переводит текст программы, написанный на языке программирования, в набор машинных кодов.

Компоновщик (линковщик, линкер) — инструментальная программа, которая производит компоновку («линковку»): принимает на вход один или несколько объектных модулей и собирает по ним исполняемый модуль.

Отладчик — компьютерная программа для автоматизации процесса отладки: поиска ошибок в других программах. В зависимости от встроенных возможностей, отладчик позволяет выполнять трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать и удалять контрольные точки или условия остановки и так далее.

Macro Assembler (MASM) — ассемблер для процессоров семейства x86. Включает компилятор, линковщик и дополнительные инструменты, в т.ч. отладчик. Другие распространённые (сейчас или в прошлом) ассемблеры - TASM, NASM, FASM и др.

.COM (англ. command) — расширение файла, в системах DOS COM-файл — простой тип исполняемого файла, при выполнении которого *данные, код* и стек находятся в одном и том же 16-битном сегменте. Поэтому размер файла не может превышать 65280 байт (что на 256 байт меньше размера сегмента — 2^{16} байт). COM-файлы для DOS можно выполнять в некоторых версиях Windows, а также на эмуляторах.

.COM — один из простейших форматов исполняемых файлов для процессоров семейства x86. Программа, загруженная в память для исполнения, является точной копией файла на диске.

Регистры процессора — блок ячеек памяти, образующий сверхбыструю оперативную память внутри процессора. Большинство команд процессора манипулируют данными, хранящимися в регистрах.

Система команд (также набор команд) — соглашение о предоставляемых архитектурой средствах программирования, в том числе, типах данных, наборе инструкций, системе регистров и т.д.

Система команд x86 процессора включает в себя следующие четыре основные группы команд:

- команды пересылки данных;
- арифметические команды;
- логические команды;
- команды переходов.

Регистр указателя команд (**IP, Instruction pointer**) - специальный регистр, который всегда хранит в себе смещение команды, которая будет выполнена следующей. Меняется автоматически по ходу выполнения программы и не может быть изменён программно.

Регистры общего назначения процессора (РОН) 8086. Регистры общего назначения - группа регистров, доступная для чтения/записи основными командами.

Предназначены для временного хранения данных, записи параметров машинных команд, арифметической обработки и т.д. Существует всего 4 РОН: AX, BX, CX, DX. Каждый содержит в себе 16 бит и делится на 2 части по 8 бит - старшую (high, H) и младшую (low, L). Обращаться можно как к регистру целиком, так и к его половинам по отдельности.

AX		аккумулятор - умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода);
АН	AL	
BX		базовый регистр в вычислениях адреса, часто указывает на начальный адрес (называемый базой) структуры в памяти;
ВН	BL	
CX		счетчик циклов, определяет количество повторов некоторой операции;
СН	CL	
DX		определение адреса ввода/вывода, так же может содержать данные, передаваемые для обработки в подпрограммы.
ДН	DL	

Основные команды

MOV - команда пересылки. Формат: MOV <приёмник>, <источник>. Записывает значение, хранящееся в источнике, в приёмник. Источником может быть константа, регистр или адрес в памяти. Приёмником - регистр или адрес в памяти.

ADD, SUB, MUL, DIV - команды сложения, вычитания, умножения, деления.

Практическое задание

1. Создайте файл hello.asm с текстом

```
.MODEL TINY
.DOSSEG
.DATA
    MSG DB "Hello, World!", 0Dh, 0Ah, '$'
.CODE
.STARTUP
    MOV BX, 1
    MOV AH, 09h
    MOV DX, OFFSET MSG
    ADD BX, 10
    INT 21h
    MOV AH, 4Ch
    INT 21h
END
```

2. Запустите ML.EXE /AT hello.asm.
3. Посмотрите, какие файлы были созданы компилятором. Определите их размер. Просмотрите файл в любом hex-viewer'e, проанализируйте содержимое файла и изучите возможности просмотрщика.
4. Запустите скомпилированную программу.
5. Запустите AFDPRO.EXE HELLO.COM.
6. Изучите возможности отладчика: шаг с заходом, шаг с обходом, перезапуск программы, создание точек останова, наблюдение за регистрами. Справочная информация по отладчику есть в файле ASM1_AFD.pdf (выполнять задание из него не нужно).

```

1. НАБРАТЬ ПРОГРАММУ
; *****
; *   Лабораторная работа N1   *
; *   Изучение отладчика AFD   *
; *****
; -----
; Примечание: Программа выводит на дисплей сообщение и
;              и ожидает нажатия клавиши , код символа
;              помещается в регистр AL
; Справка....: DS:DX - адрес строки;
;              Функции DOS :
;              09h выдать на дисплей строку,
;              07h ввести символ без эха,
;              4Ch завершить процесс ;
;              INT 21h - вызов функции DOS
;
StkSeg  SEGMENT PARA STACK 'STACK'
        DB      200h DUP (?)
StkSeg  ENDS
;
DataS   SEGMENT WORD 'DATA'
HelloMessage  DB      13              ;курсор поместить в нач. строки
               DB      10              ;перевести курсор на нов. строку
               DB      'Hello, world !' ;текст сообщения
               DB      '$'            ;ограничитель для функции DOS
DataS     ENDS
;
Code     SEGMENT WORD 'CODE'
        ASSUME CS:Code, DS:DataS
DispMsg:
        mov     AX,DataS              ;загрузка в AX адреса сегмента данных
        mov     DS,AX                ;установка DS
        mov     DX,OFFSET HelloMessage ;DS:DX - адрес строки
        mov     AH,9                 ;AH=09h выдать на дисплей строку
        int     21h                  ;вызов функции DOS
        mov     AH,7                 ;AH=07h ввести символ без эха
        int     21h                  ;вызов функции DOS
        mov     AH,4Ch                ;AH=4Ch завершить процесс
        int     21h                  ;вызов функции DOS
Code     ENDS
        END     DispMsg

```

2. ТРАНСЛИРОВАТЬ ПРОГРАММУ

3. СОЗДАТЬ ИСПОЛНЯЕМЫЙ ФАЙЛ

4. ВЫПОЛНИТЬ ПРОГРАММУ ПОД УПРАВЛЕНИЕМ ОТЛАДЧИКА AFD

Отладчик AFD позволяет программисту проверять и модифицировать :

- а) необработанные шестнадцатеричные данные,
- б) содержимое стека,
- в) содержимое регистров процессора,
- г) состояние флагов процессора.

После старта программы AFD на экране появляются 7 окон :

Верхний ряд : левое - регистры процессора,
 среднее - стек,
 правое - флаги.

Средний ряд : левое верхнее - окно команд отладчика,
 левое нижнее - окно кода программы,
 правое - окно данных.

Нижний ряд: окно данных, которое отображает выбранную область
 памяти в двух форматах : шестнадцатеричном и
 символьном

Нижняя строка экрана предназначена для управления режимом работы отладчика с помощью функциональной клавиатуры :

- F1 - пошаговый режим (шаг - команда),
- F2 - процедурный режим (шаг - процедура),
- F3 - просмотр буфера команд отладчика,
- F4 - включение/выключение подсказки,
- F5 - вызов меню обработки точек прерывания,
- F7 - F10 - выбор активного окна.

Управление работой отладчика может осуществляться с помощью команд :
Команды отладчика AFD

Команда	Действие
? {%= выражение	калькулятор (? %= десятичный тип)
D адрес	выдать код, начиная с данного адреса
F адрес,длина,значение	заполнить область памяти значением
G {адрес_нач.},{адрес_прер.}	выполнить программу
GC {адрес_нач.},{адрес_прер.}	продолжить выполнение программы без сброса счетчика прерываний
L {/адрес} имя_файла {парам}	загрузить из файла
PT {/S} {номер_нач.,колич.{,имя_файла}}	вывод содержимого буфера трассировки
QUIT	прекратить отладку
S {{адрес},строка}	искать строку, начиная с адреса
SI {{адрес},команда}	искать команду, начиная с адреса
TR {ON OFF} {CLR}	управление трассировкой
W {адрес,длина,имя_файла}	записать область памяти в файл

- 1) Вызвать AFD, набрав в командной строке AFDPRO.
- 2) Загрузить и выполнить программу.
- 3) Загрузить и выполнить программу в пошаговом режиме (F2).
- 4) Загрузить программу и выполнить установку регистра DS.
- 5) Записать 16 байт, начиная с адреса HelloMessage в файл.
- 6) Выйти из AFD.
- 7) Выдать на экран файл, записанный в п.5.
- 8) Повторить пп 3-6, изменив с помощью отладчика текст сообщения.
- 9) Загрузить и выполнить программу, проверяя содержимое регистров при каждом обращении к функциям DOS (использовать команды G и SI).

5. ВЫПОЛНИТЬ ПРОГРАММУ С УСТАНОВКОЙ ТОЧЕК ПРЕРЫВАНИЯ

Описание меню обработки точек прерывания :

- F1 - просмотр трассировки программы,
- F3 - загрузить настройку точек прерывания,
- F4 - включение/выключение подсказки,
- F5 - возврат в командный режим,
- F7 - сохранить настройку точек прерывания,
- F9 - сбросить настройку точек прерывания,
- F10 - сбросить счетчик.

Для настройки каждой точки прерывания необходимо установить :
адрес, условие, количество, ответное действие (S - сохранить содержимое регистров, B - прервать выполнение программы).

- 1) Вызвать AFD.
- 2) Установить точки прерывания :
 - а) для проверки содержимого строки перед выводом на экран,
 - б) для проверки кода символа, введенного с клавиатуры (регистр AL).
 - в) для проверки наличия кода перевода курсора (ASCII код 13) .
- 3) Выполнить программу.
- 4) Проверить содержимое области памяти, начиная с адреса HelloMessage.
- 5) Определите символ введенный с клавиатуры.
- 6) Сохранить настройку точек прерывания.
- 7) Сбросить настройку точек прерывания.
- 8) Загрузить настройку точек прерывания.
- 9) Сохранить содержимое регистров для п.2б в файле CODE.KEY
- 10) Выйти из AFD.
- 11) Выдать на экран файл, записанный в п.9.

Машинно-зависимые языки программирования

Лабораторная работа №3

“EXE-файлы. Сегменты. Простейший ввод-вывод средствами DOS”

Справочная информация

Логическая структура памяти. Сегменты

Любая программа состоит из одного или нескольких сегментов (блоков памяти размером до 64 КБ). Сегменты могут быть следующих типов: кода, данных, стека. За адрес начала сегмента отвечают сегментные регистры: для кода - CS, для стека - SS, для данных - DS и дополнительные ES, FS, HS.

Составление программ на ассемблере

Как и на других языках программирования, программа на ассемблере может состоять из нескольких файлов - модулей. При компиляции (трансляции) каждый модуль превращается в объектный файл, далее при компоновке объектные файлы соединяются в единый исполняемый модуль.

Директивы ассемблера - ключевые слова в тексте программы на языке ассемблера, влияющие на процесс ассемблирования или свойства выходного файла.

Модули обычно состоят из описания сегментов будущей программы с помощью директивы SEGMENT.

Пример:

```
имя SEGMENT [READONLY] выравнивание тип разряд 'класс'
...
имя ENDS
```

Параметры:

- Выравнивание - расположение начала сегмента с адреса, кратного какому-либо значению. Варианты: BYTE, WORD (2 байта), DWORD (4 байта), **PARA (16 байт, по умолчанию)**, PAGE (256 байт).
- Тип: PUBLIC (сегменты с одним именем объединятся в один); STACK (для стека); COMMON (сегменты будут “наложены” друг на друга по одним и тем же адресам памяти); AT <начало> - расположение по фиксированному физическому адресу, параметр - сегментная часть этого адреса; PRIVATE - вариант по умолчанию.
- Класс - метка, позволяющая объединить сегменты (расположить в памяти друг за другом).

Выделение памяти

Директивы выделения памяти: DB (байт), DW (слово), DD (двойное слово).

Описание строки программы

метка команда/директива операнды ; комментарий

Любое поле может быть опущено.

Метка в коде заканчивается двоеточием и обозначают ссылку на команду, расположенную за ней.

```
mov cx, 5
label1:
    add ax, bx
    loop label1
```

Метка в описании данных является ссылкой на переменную, расположенную после неё. Метка не является директивой выделения памяти (см. л/р 2).

метка label тип

Допустимые типы: BYTE, WORD, DWORD, FWORD, QWORD, TBYTE (для данных), NEAR, FAR (для указателей на команды).

Команда организации цикла

Команда LOOP <метка> - команда организации цикла со счётчиком. Уменьшает регистр CX на 1 и выполняет переход на метку, если новое значение регистра не равно нулю.

Директива ASSUME

ASSUME регистр:имя сегмента

Является инструкцией компилятору, указывающей, какой сегментный регистр с каким сегментом будет связан во время работы программы. Используется для контроля правильности обращения к переменным и автоматического определения сегментного префикса в машинных командах работы с памятью.

EXE-файлы

Исполняемые файлы с расширением EXE, используемые в ОС DOS и Windows (и некоторых других).

Может быть нескольких форматов:

- MZ - 16-битный формат, основной для DOS;
- NE - 16-битный формат старых версий Windows;
- LE, LX - формат OS/2;
- PE - 32- и 64-битный формат современных Windows (начиная с Windows 95).

MZ — стандартный формат 16-битных исполняемых файлов с расширением .EXE для DOS. Назван так по сигнатуре — ASCII-символам MZ (4D 5A) в первых двух байтах. Эта сигнатура — инициалы Марка Збиковски, одного из создателей MS-DOS.

Формат был разработан как замена устаревшему формату .COM. Исполняемые файлы MZ включают метаданные, могут иметь размер больше 64 Кбайт и использовать несколько сегментов памяти различного типа (кода, данных и стека), точка входа в программу также может быть в любом месте (в файлах .COM выполнение команд всегда начинается непосредственно с начала файла). Метод загрузки исполняемого файла определяется по сигнатуре: при её наличии обрабатывается MZ-заголовок, при отсутствии файл запускается как .COM — независимо от расширения файла (например, в последних версиях MS-DOS интерпретатор командной строки COMMAND.COM на самом деле является EXE-файлом).

Исполняемые файлы более поздних форматов для Windows начинаются с MZ-заглушки. Обычно заглушка, добавляемая компиляторами, выводит сообщение наподобие «This program cannot be run in DOS mode» («Эту программу невозможно запустить в режиме DOS»).

([https://ru.wikipedia.org/wiki/MZ_\(%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82\)](https://ru.wikipedia.org/wiki/MZ_(%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82)))

EXE-файлы состоят из заголовка и собственно загружаемой части (тела файла).
Заголовок MS-DOS (размер 40H байт):

Адрес	Тип	Имя	Описание
00h	word	Сигнатура	Магическая сигнатура DOS-файла - два символа "MZ"
02h	word	Extra bytes	Количество байт на последней странице файла
04h	word	Pages	Количество страниц в файле
06h	word	Relocation items	Количество релокейшенов
08h	word	Header size	Размер заголовка в параграфах
0Ah	word	Minimum allocation	Мин. выделение памяти в параграфах
0Ch	word	Maximum allocation	Макс. выделение памяти в параграфах
0Eh	word	Initial SS	Начальное (относительное) значение регистра SS
10h	word	Initial SP	Начальное значение регистра SP
12h	word	Checksum	Контрольная сумма

14h	word	Initial IP	Начальное значение регистра IP
16h	word	Initial CS	Начальное (относительное) значение регистра CS
18h	word	Relocation table	Адрес на релокейшны и программу-заглушку
1Ah	word	Overlay	Количество оверлеев
1Ch	word	Overlay information	Зарезервировано
24h	word	OEMIdentifier	Для OEMInfo
26h	word	OEMInfo	Информация о программе
28h	word	Res1[10]	Зарезервировано
3Ch	dword	PEHeaderAddr	Адрес в файле заголовка PE

(<http://mzc.narod.ru/Creating/Step008.htm>, <https://wiki.osdev.org/MZ>)

Практическое задание

1. Набрать программу из файла ASM1_AFD.pdf, скомпилировать и сформировать исполняемый файл с помощью команд `masm+link` или `ml`.
2. Проанализировать получившиеся файлы.
3. Запустить программу, убедиться в работоспособности.
4. Запустить программу в отладчике, пронаблюдать изменение сегментного регистра DS и изучить содержимое сегментов кода и данных.
5. Дописать исходный текст программы так, чтобы строка выводилась на экран 3 раза.

Машинно-зависимые языки программирования

Лабораторная работа №4

“Многомодульные программы”

Справочная информация

Директивы глобальных объявлений

`PUBLIC` идентификатор

Описывает идентификатор, как доступный из других модулей.

`EXTRN` определение [, определение] .

Указывает, что идентификатор определен в другом модуле. Определение описывает идентификатор и имеет следующий формат:

`ИМЯ : ТИП`

"Имя" - это идентификатор, который определен в другом модуле. "Тип" должен соответствовать типу идентификатора, указанному при его определении, и может быть следующим: `NEAR`, `FAR`, `PROC`, `BYTE`, `WORD`, `DWORD`, `DATAPTR`, `CODEPTR`, `FWORD`, `PWORD`, `QWORD`, `TBYTE`, `ABS` или именем структуры.

Виды переходов (передачи управления)

Короткий (`short`) - в пределах адресов `-128..+127` от текущего значения `IP` (1 байт).

Ближний (`near`) - в пределах того же сегмента (2 байта).

Дальний (`far`) - на произвольный адрес (4 байта).

Прерывания

Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передается программе-обработчику возникшего прерывания.

Виды прерываний:

- аппаратные (асинхронные) - события от внешних устройств;
- внутренние (синхронные) - события в самом процессоре, например, деление на ноль;
- программные - вызванные командой `int`.

В процессоре 8086 доступно 256 возможных прерываний. Далеко не все из них используются устройствами. Часть задействуется операционной системой для

организации взаимодействия с программами. Значительная часть не используется вообще.

Прерывание 21h - так называемое прерывание DOS, обращение к которому приводит к вызову специального обработчика прерывания, установленного операционной системой при загрузке компьютера. Этот обработчик позволяет выполнять различные функции, полезные для прикладной программы. Номер функции передаётся через регистр AH.

Часть доступных функций - ввод с клавиатуры и вывод на экран (в положение курсора).

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL – ASCII-код символа
02	Вывод символа в stdout	DL = ASCII-код символа	-
08	Считать символ без эха	-	AL – ASCII-код символа
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом \$	-
0Ah	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
4Ch	Завершить программу	AL = код завершения	-

4Ch - “особая” функция. При её вызове управление в программу не вернётся, память, занимаемая программой, будет очищена, и управление вернётся вызвавшей программе.

Режимы видеоадаптера

Режим видеоадаптера (видеокарты) - комбинация параметров, определяющая способ вывода информации на экран, а также разрешение, количество цветов, частоту обновления и т.д.

До настоящего времени видеоадаптеры, помимо графического режима, поддерживают текстовый режим работы. BIOS'ы многих компьютеров используют текстовый режим на отдельных этапах загрузки.

Текстовый видеорежим - режим видеоадаптера, в котором экран представлен не отдельными пикселями, а решёткой знакомест. В каждом из знакомест может находиться один символ из заранее загруженного набора.

Стандартный размер экрана в текстовом режиме - 25 строк по 80 символов.

Видеопамять — это внутренняя оперативная память, отведённая для хранения данных, которые используются для формирования изображения на экране монитора.

Видеопамять текстового режима доступна по адресу B8000h. Символы, выводимые на экран, представлены там в виде матрицы 25x80, по 2 байта на каждый символ. Один байт соответствует ASCII-коду символа, другой - атрибутам. Байт атрибутов имеет следующий формат: старший бит - признак мерцания символа, затем 3 бита определяют цвет фона в формате RGB (допустимые цвета - от чёрного до серого, с пониженной яркостью). Младшие 4 бита отвечают за цвет самого символа: старший из них - признак яркости, младшие - RGB. Таким образом, доступно 8 цветов фона и 16 цветов символа.

Оператор SEG

Возвращает сегментную часть адреса операнда.

Команды CALL и RET

CALL осуществляет вызов подпрограммы (передачу управления по метке с сохранением адреса возврата в стек). RET возвращает управление по адресу из стека.

Команда XCHG

Осуществляет обмен двух значений местами.

Практическое задание

- I. Скомпилировать, запустить и изучить 4 примера программ с несколькими модулями/несколькими сегментами.
- II. Составить программу согласно индивидуальному заданию. **Гарантируется, что ввод для любой программы будет корректный, позволяющий выполнить задание без дополнительных проверок.**

Перечень индивидуальных заданий

Требуется написать программу ...

ИУ7-41Б

Анохина К А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод цифры, в другом - вывод на экран этой цифры, увеличенной на 1.
-------------	---

Базалеев Ф Е	из двух модулей. В одном модуле объявить сегмент данных, в другом - сегмент кода. Ввести строку цифр и затем вывести ту цифру, номер которой в строке равен значению первой цифры.
Барков М Д	из двух модулей, в одном осуществить ввод строки, затем передать управление с помощью дальнего перехода в другой, где вывести 5-й символ на экран
Варламова Е А	с двумя сегментами кода в одном модуле. В одном сегменте ввести символ без эха, затем передать управление в другой сегмент и вывести символ
Власов Д В	из двух модулей. Точку входа расположить в первом, затем сразу передать управление во второй, где ввести строку цифр от 0 до 4 в сегмент данных второго модуля. Потом вернуть управление в первый и вывести сумму 2-й и 5-й цифр.
Воронин Е Д	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. Код в первом модуле должен позволять ввести до 10 цифр в строку из 11 байт, предварительно инициализированную пробелами и заканчивающуюся знаком \$. Код второго модуля должен вывести эту строку на экран.
Гриценко А М	которая обеспечит ввод последовательности строчных латинских букв. Затем требуется вывести заглавный аналог 4-й буквы.
Ефимова М В	из двух модулей. В первом модуле ввести латинскую букву с эхом и передать управление с помощью дальнего перехода во второй, где без эха ввести цифру К и затем через пробел вывести другую букву, расположенную в алфавите левее исходной на К позиций.
Костев Д И	с двумя сегментами кода в одном модуле. В одном сегменте ввести заглавную латинскую букву, затем передать управление в другой сегмент и вывести с новой строки строчную букву, соответствующую исходной
Котёлкин И Д	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру К с эхом. Далее во втором с новой строки ввести букву латинского алфавита и затем через пробел вывести другую букву, расположенную в алфавите правее исходной на К позиций.
Лапаев Д А	с двумя сегментами данных. Ввести в первый сегмент строку из 10 символов. Во втором сегменте подготовить строку из 6 символов, заполненную пробелами и заканчивающуюся знаком \$. Переписать из введённой строки во 2-й сегмент только символы на чётных позициях и вывести новую строку на экран.
Малышев И А	из двух модулей. Код в первом модуле должен позволять ввести до 10 цифр в строку из 11 байт, предварительно инициализированную пробелами и заканчивающуюся знаком \$. Затем управление передаётся во второй модуль, код которого должен вывести эту строку на экран.
Матвеев М И	с двумя сегментами данных. В первый ввести строку, во второй - переписать первый и последний символы этой строки и вывести их на экран

Михеев Д Н	с двумя сегментами данных. Ввести в первый сегмент строку из 5 заглавных латинских букв. Во втором сегменте подготовить строку из 6 символов, заполненную пробелами и заканчивающуюся знаком \$. Переписать из введенной строки во 2-й сегмент символы исходной строки в нижнем регистре и вывести новую строку на экран.
Мицевич М Д	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. Точку входа разместить в первом, затем сразу передать управление во 2-й. Там ввести 2 цифры, таких, что 2-я не превышает 1-ю, и посчитать значение их разности. Затем вернуть управление в 1-й модуль и вывести получившуюся разность на экран.
Тумайкина Д	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод последней цифры числа от 10 до 15, в другом - вывод на экран этого числа в 16-ричной с/с.
Фролова Е В	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод цифры, в другом - вывод на экран этой цифры, уменьшенной на 2.
Шацкий Н С	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом требуется ввести латинскую букву и цифру N, затем записать в сегмент данных, объявленный во первом модуле, букву, расположенную раньше исходной на N позиций. Во втором модуле вывести полученную букву.
Шумнов П А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру от 0 до 4, во втором - вывести через пробел удвоенное значение этой цифры.
Наврози М	с двумя сегментами кода в одном модуле. В одном сегменте ввести строчную латинскую букву, затем передать управление в другой сегмент и вывести через пробел заглавную букву, соответствующую исходной
Нямдорж Э	из двух модулей. В первом требуется ввести латинскую букву и цифру N, затем записать в сегмент данных, объявленный во втором модуле, букву, расположенную дальше исходной на N позиций. После этого передать управление с помощью дальнего перехода в сегмент кода второго модуля и там вывести полученную букву.

ИУ7-42Б

Агеев Д А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод символа, в другом - вывод этого символа на экран
Алахов А Г	из двух модулей, в одном осуществить ввод символа без эха и передать управление с помощью дальнего перехода в другой, где вывести введенный символ на экран

Батраков Д В	в которой ввести строку и затем вывести её 5-й символ
Бобров М Э	в которой ввести строку и затем вывести первые 10 её символов
Богатырев И С	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру от 1 до 5 без эха, во втором - вывести её, увеличенную на 3.
Боренко А Д	из двух модулей, в одном осуществить ввод символа без эха и передать управление с помощью дальнего перехода в другой. В первом модуле ввести цифру от 1 до 5 без эха, во втором - вывести её, увеличенную на 3.
Зайцева А А	в которой ввести строку в один сегмент данных, затем скопировать первые 4 символа в переменную в другом сегменте данных и вывести 2-й из них на экран.
Климов И С	из двух модулей, в первом ввести строчную букву латинского алфавита и сохранить ей в сегмент данных, объявленный во втором. Дальним переходом передать управление во второй модуль и там вывести заглавный вариант введённой буквы через пробел.
Козлова И В	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести заглавную букву латинского алфавита, в другом - вывод строчного варианта той же буквы с новой строки.
Короткая В М	в которой ввести в переменную последовательность строчных латинских букв и затем вывести с новой строки заглавный вариант 3-й буквы.
Лисневский А В	из двух модулей, в первом ввести 2 цифры от 1 до 4, во втором - вывести сумму этих цифр. Сегменты коды должны объединяться в единый.
Межеровский А С	из двух модулей, в первом ввести 2 цифры от 5 до 9 и от 1 до 5, во втором - вывести разность этих цифр. Сегменты коды должны объединяться в единый.
Мухамедиев Р Н	в которой ввести строку цифр от 0 до 5 и затем вывести сумму 2-й и 4-й цифр.
Никуленко И В	с двумя сегментами данных. Ввести строку цифр от 0 до 5 в первый сегмент, далее во 2-й байт второго сегмента записать сумму 1-й и 3-й цифр и вывести её на экран.
Нисхизов В В	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод строки в сегмент данных, объявленный во втором модуле, во втором - вывести 2-й символ этой строки.
Пилипчук М В	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку в сегмент данных, объявленный также в первом модуле, во втором - вывести 2, 4, 6, 8, 10-й символы
Пронин А С	из двух модулей. В первом ввести строку цифр от 0 до 5 в сегмент данных, объявленный во втором модуле, затем передать управление

	дальним переходом и вывести сумму 2-й и 3-й цифр.
Скотников Д А	из двух модулей. Точку входа разместить в первом, затем передать управление во второй, где ввести символ без эха и вернуть управление в первый. В первом вывести символ.
Сучкова Т М	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести символ без эха и вернуть управление в первый. В первом вывести символ.
Сысоева В Р	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести строку, объявленную в сегменте данных первого модуля, и вернуть туда управление. В первом вывести 3-й символ введенной строки.
Трунов А Р	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести 2 цифры от 1 до 5, вернуть управление в 1-й и вывести сумму этих цифр.
Черненко В Д	из двух модулей, сегменты кода которых должны объединяться в единый. В первом ввести строку заглавных латинских букв, во втором вывести строчный аналог 2-й буквы.
Жигжид Т	из двух модулей, сегменты кода которых должны объединяться в единый. Первый должен обеспечить ввод двух цифр от 0 до 5 в сегмент данных второго, второй - вывести сумму этих цифр.
Чыонг Н	в которой ввести 2 цифры, одна от 3 до 9, вторая от 0 до 3, и сохранить их в переменных. Вывести с новой строки разность этих цифр.

ИУ7-43Б

Артемьев И О	с двумя сегментами данных. Ввести строку цифр до 0 до 5 в первый сегмент, далее во 2-й байт второго сегмента записать сумму 2-й и 5-й цифр и вывести её на экран.
Барышникова Ю Г	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести 2 цифры от 1 до 4, во втором - вывести сумму этих цифр.
Буртелов Н Н	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где осуществить ввод строчной буквы латинского алфавита и цифры К. Затем вернуть управление в первый модуль и там вывести букву, которая в алфавите на К позиций правее исходной.
Волков М М	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом осуществить ввод строчной буквы латинского алфавита и цифры К. Во втором вывести букву, которая в алфавите на К позиций левее исходной.

Дегтярев А И	из двух модулей. В первом ввести строку цифр от 1 до 5 в сегмент данных второго модуля, затем дальним переходом передать управление в сегмент кода второго модуля, где вывести сумму 2-й и 3-й цифр.
Диордиев К	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку чередующихся символов и цифр вида СЦСЦСЦСЦ, во втором - сдвинуть 3-й символ на количество позиций алфавита, равное значению следующей за ним цифры.
Заборовская А Д	с двумя сегментами данных. В первый ввести 2 цифры от 1 до 5, во второй байт второго сегмента записать число, соответствующее сумме этих цифр, и вывести его на экран.
Кишов Г М	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку символов и цифру I, во втором - вывести I-й символ исходной строки.
Ковалец К Э	с двумя сегментами данных. Ввести строку цифр от 0 до 5 в первый сегмент, далее во 2-й байт второго сегмента записать сумму 2-й и 5-й цифр.
Костоев А И	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести строку в сегмент данных, объявленный в первом модуле, и вернуть туда управление. В первом вывести 4-й символ введенной строки.
Криков А В	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку в сегмент данных, объявленный также в первом модуле, во втором - вывести 1, 4, 7, 10-й символы
Лёшин Д А	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести заглавную букву латинского алфавита, вернуть управление в первый и вывести строчный аналог этой буквы
Маслова М Д	в которой ввести строку и затем вывести первые 7 её символов
Миронов Г А	с двумя сегментами данных. В первый букву и цифру М, затем в переменную второго сегмента записать новую букву, сдвинутую на М позиций влево в алфавите относительно исходной, и вывести её на экран.
Недолужко Д В	из двух модулей. В первом ввести строку цифр от 0 до 5 в сегмент данных, объявленный во втором модуле, затем передать управление дальним переходом и вывести разность 2-й и 3-й цифр.
Рядинский К В	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести символ в строку, объявленную в первом модуле, и вернуть туда управление. В первом вывести 4-й символ введенной строки.
Сабуров С М	в которой ввести строку и затем вывести её 3-й символ
Салатов Х Х	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку символов и

	цифру I, во втором - вывести первые I символов исходной строки через пробел
Филипенков В А	в которой ввести 2 цифры, одна от 3 до 9, вторая от 0 до 3, и сохранить их в переменных. Вывести с новой строки разность этих цифр
Хамзина Р Р	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру, во втором - строчную букву латинского алфавита с порядковым номером, соответствующим этой цифре
Цветков И А	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где осуществить ввод строки, затем вернуть управление в первый и там вывести первые 8 символов введенной строки
Шевцов Е О	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где осуществить ввод символа, затем вернуть управление в первый и там вывести символ на экран
Тэмуужин Я	из двух модулей. В первом ввести цифру N, затем передать управление с помощью дальнего перехода во второй и там вывести N раз букву A

ИУ7-44Б

Аксенов Е Ю	с двумя сегментами кода в одном модуле. В одном сегменте ввести заглавную латинскую букву, затем передать управление в другой сегмент и вывести через пробел строчную букву, соответствующую исходной
Алферова И В	из двух модулей. Точку входа расположить в первом, затем сразу передать управление во второй, где ввести строку цифр от 0 до 4 в сегмент данных второго модуля. Потом вернуть управление в первый и вывести сумму 2-й и 5-й цифр.
Андреев А А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. Код в первом модуле должен позволять ввести до 10 цифр в строку из 11 байт, предварительно инициализированную пробелами и заканчивающуюся знаком \$. Код второго модуля должен вывести эту строку на экран.
Дадобоев А А	которая обеспечит ввод последовательности строчных латинских букв. Затем требуется вывести заглавный аналог 4-й буквы.
Елгин И Ю	из двух модулей. В первом модуле ввести латинскую букву с эхом и передать управление с помощью дальнего перехода во второй, где без эха ввести цифру K и затем через пробел вывести другую букву, расположенную в алфавите левее исходной на K позиций.
Ефремов Р Д	с двумя сегментами кода в одном модуле. В одном сегменте ввести заглавную латинскую букву, затем передать управление в другой сегмент и вывести с новой строки строчную букву, соответствующую исходной

Жабин Д В	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру К с эхом. Далее во втором с новой строки ввести букву латинского алфавита и затем через пробел вывести другую букву, расположенную в алфавите правее исходной на К позиций.
Иванова М С	с двумя сегментами данных. Ввести в первый сегмент строку из 10 символов. Во втором сегменте подготовить строку из 6 символов, заполненную пробелами и заканчивающуюся знаком \$. Переписать из введённой строки во 2-й сегмент только символы на чётных позициях и вывести новую строку на экран.
Калита Н В	из двух модулей. Код в первом модуле должен позволять ввести до 10 цифр в строку из 11 байт, предварительно инициализированную пробелами и заканчивающуюся знаком \$. Затем управление передаётся во второй модуль, код которого должен вывести эту строку на экран.
Кузин А Д	с двумя сегментами данных. В первый ввести строку, во второй - переписать первый и последний символы этой строки и вывести их на экран
Ларин В Н	с двумя сегментами данных. Ввести в первый сегмент строку из 5 заглавных латинских букв. Во втором сегменте подготовить строку из 6 символов, заполненную пробелами и заканчивающуюся знаком \$. Переписать из введённой строки во 2-й сегмент символы исходной строки в нижнем регистре и вывести новую строку на экран.
Лисичкин Г О	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. Точку входа разместить в первом, затем сразу передать управление во 2-й. Там ввести 2 цифры, таких, что 2-я не превышает 1-ю, и посчитать значение их разности. Затем вернуть управление в 1-й модуль и вывести получившуюся разность на экран.
Минасян Г Г	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод последней цифры числа от 10 до 15, в другом - вывод на экран этого числа в 16-ричной с/с.
Молодняков О А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод цифры, в другом - вывод на экран этой цифры, уменьшенной на 2.
Осипов А А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом требуется ввести латинскую букву и цифру N, затем записать в сегмент данных, объявленный во первом модуле, букву, расположенную раньше исходной на N позиций. Во втором модуле вывести полученную букву.
Параскун С Д	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру от 0 до 4, во втором - вывести через пробел удвоенное значение этой цифры.
Платонович Е В	из двух модулей. В первом модуле ввести цифру от 5 до 9, затем

	передать управление во второй с помощью дальнего перехода, где вывести через пробел значение этой цифры, уменьшенное на 5.
Подпружников Н В	из двух модулей. В первом модуле ввести цифру М, затем передать управление во второй с помощью дальнего перехода, где вывести через пробел строчную латинскую букву, расположенную на М-й позиции с начала алфавита.
Сергиевич Т Г	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру от 0 до 4, во втором - вывести через пробел значение этой цифры, увеличенное на 5.
Тартыков Л Е	из двух модулей. В первом модуле ввести латинскую букву, затем передать управление во второй с помощью дальнего перехода, где вывести через пробел букву, расположенную на симметричной позиции с конца алфавита.
Халимов И С	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру М, а во втором вывести через пробел заглавную латинскую букву, расположенную на М-й позиции с конца алфавита.
Шингаров И Д	с двумя сегментами кода в одном модуле. В одном сегменте ввести строчную латинскую букву, затем передать управление в другой сегмент и вывести через пробел заглавную букву, соответствующую исходной
Юсупов Ф К	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод цифры, в другом - вывод на экран этой цифры, увеличенной на 1.

ИУ7-45Б

Александрович Г Ю	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод символа, в другом - вывод этого символа на экран
Багиров С Н	из двух модулей, в одном осуществить ввод символа без эха и передать управление с помощью дальнего перехода в другой, где вывести введенный символ на экран
Балагуров И С	в которой ввести строку и затем вывести её 5-й символ
Бугаенко А П	в которой ввести строку и затем вывести первые 10 её символов
Володин В А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру от 1 до 5 без эха, во втором - вывести её, увеличенную на 3.
Исупов А Д	из двух модулей, в одном осуществить ввод символа без эха и передать управление с помощью дальнего перехода в другой. В первом модуле ввести цифру от 1 до 5 без эха, во втором - вывести

	её, увеличенную на 3.
Клименко А К	в которой ввести строку в один сегмент данных, затем скопировать первые 4 символа в переменную в другом сегменте данных и вывести 2-й из них на экран.
Коротыч М Д	из двух модулей, в первом ввести строчную букву латинского алфавита и сохранить ей в сегмент данных, объявленный во втором. Дальним переходом передать управление во второй модуль и там вывести заглавный вариант введённой буквы через пробел.
Лемешкин Б А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести заглавную букву латинского алфавита, в другом - вывод строчного варианта той же буквы с новой строки.
Муравьев И А	в которой ввести в переменную последовательность строчных латинских букв и затем вывести с новой строки заглавный вариант 3-й буквы.
Прянишников А Н	из двух модулей, в первом ввести 2 цифры от 1 до 4, во втором - вывести сумму этих цифр. Сегменты коды должны объединяться в единый.
Серова М Н	из двух модулей, в первом ввести 2 цифры от 5 до 9 и от 1 до 5, во втором - вывести разность этих цифр. Сегменты коды должны объединяться в единый.
Ском Э П	в которой ввести строку цифр от 0 до 5 и затем вывести сумму 2-й и 4-й цифр.
Солнцева Т В	с двумя сегментами данных. Ввести строку цифр до 0 до 5 в первый сегмент, далее во 2-й байт второго сегмента записать сумму 1-й и 3-й цифр и вывести её на экран.
Тагилов А М	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В одном осуществить ввод строки в сегмент данных, объявленный во втором модуле, во втором - вывести 2-й символ этой строки.
Терехин Ф А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку в сегмент данных, объявленный также в первом модуле, во втором - вывести 2, 4, 6, 8, 10-й символы
Тонкоштан А А	из двух модулей. В первом ввести строку цифр от 0 до 5 в сегмент данных, объявленный во втором модуле, затем передать управление дальним переходом и вывести сумму 2-й и 3-й цифр.
Фролов Е	из двух модулей. Точку входа разместить в первом, затем передать управление во второй, где ввести символ без эха и вернуть управление в первый. В первом вывести символ.
Хохлов Н С	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести символ без эха и вернуть управление в первый. В первом вывести символ.
Шацкий Р Е	из двух модулей. Точку входа разместить в первом, затем передать

	управление с помощью дальнего перехода во второй, где ввести строку, объявленную в сегменте данных первого модуля, и вернуть туда управление. В первом вывести 3-й символ введенной строки.
Шелия С М	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести 2 цифры от 1 до 5, вернуть управление в 1-й и вывести сумму этих цифр.
Каримзай А	из двух модулей, сегменты кода которых должны объединяться в единый. В первом ввести строку заглавных латинских букв, во втором вывести строчный аналог 2-й буквы.
Цзинь К	из двух модулей, сегменты кода которых должны объединяться в единый. Первый должен обеспечить ввод двух цифр от 0 до 5 в сегмент данных второго, второй - вывести сумму этих цифр.

ИУ7-46Б

Алексеев А Н	с двумя сегментами данных. Ввести строку цифр от 0 до 5 в первый сегмент, далее во 2-й байт второго сегмента записать сумму 2-й и 5-й цифр и вывести её на экран.
Борисов М А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести 2 цифры от 1 до 4, во втором - вывести сумму этих цифр.
Буланый К С	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где осуществить ввод строчной буквы латинского алфавита и цифры К. Затем вернуть управление в первый модуль и там вывести букву, которая в алфавите на К позиций правее исходной.
Варин Д В	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом осуществить ввод строчной буквы латинского алфавита и цифры К. Во втором вывести букву, которая в алфавите на К позиций левее исходной.
Денисов П А	из двух модулей. В первом ввести строку цифр от 1 до 5 в сегмент данных второго модуля, затем дальним переходом передать управление в сегмент кода второго модуля, где вывести сумму 2-й и 3-й цифр.
Журавлев Е Э	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку чередующихся символов и цифр вида СЦСЦСЦСЦ, во втором - сдвинуть 3-й символ на количество позиций алфавита, равное значению следующей за ним цифры.
Ивахненко Д А	с двумя сегментами данных. В первый ввести 2 цифры от 1 до 5, во второй байт второго сегмента записать число, соответствующее сумме этих цифр, и вывести его на экран.
Казаева Т А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку символов и цифру I, во втором - вывести I-й символ исходной строки.

Клевцов Я А	с двумя сегментами данных. Ввести строку цифр до 0 до 5 в первый сегмент, далее во 2-й байт второго сегмента записать сумму 2-й и 5-й цифр.
Котцов М Д	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести строку в сегмент данных, объявленный в первом модуле, и вернуть туда управление. В первом вывести 4-й символ введенной строки.
Кузьмин К О	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку в сегмент данных, объявленный также в первом модуле, во втором - вывести 1, 4, 7, 10-й символы
Леонов В В	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести заглавную букву латинского алфавита, вернуть управление в первый и вывести строчный аналог этой буквы
Лубянская А А	в которой ввести строку и затем вывести первые 7 её символов
Мазур Е А	с двумя сегментами данных. В первый букву и цифру М, затем в переменную второго сегмента записать новую букву, сдвинутую на М позиций влево в алфавите относительно исходной, и вывести её на экран.
Петрова А А	из двух модулей. В первом ввести строку цифр от 0 до 5 в сегмент данных, объявленный во втором модуле, затем передать управление дальним переходом и вывести разность 2-й и 3-й цифр.
Пинский М Г	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести символ в строку, объявленную в первом модуле, и вернуть туда управление. В первом вывести 4-й символ введенной строки.
Порошин Д Ю	в которой ввести строку и затем вывести её 3-й символ
Руденко И А	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом ввести строку символов и цифру I, во втором - вывести первые I символов исходной строки через пробел
Сироткина П Ю	в которой ввести 2 цифры, одна от 3 до 9, вторая от 0 до 3, и сохранить их в переменных. Вывести с новой строки разность этих цифр
Слепокурова М Ф	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру, во втором - строчную букву латинского алфавита с порядковым номером, соответствующим этой цифре
Супрунова Е А	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где осуществить ввод строки, затем вернуть управление в первый и там вывести первые 8 символов введенной строки
Таламбуца А Ю	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где осуществить ввод символа, затем вернуть управление в первый и там вывести символ

	на экран
Трошкин Н Р	из двух модулей. В первом ввести цифру N, затем передать управление с помощью дальнего перехода во второй и там вывести N раз букву A
Хрюкин А А	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести цифру M, вернуть управление в первый и вывести M раз букву Z, каждый раз с новой строки
Андрич К	из двух модулей. Точку входа разместить в первом, затем передать управление с помощью дальнего перехода во второй, где ввести строчную букву латинского алфавита, вернуть управление в первый и вывести заглавный аналог этой буквы
Баттулга Б	из двух модулей, в которых объявить по сегменту кода, которые должны объединяться в единый. В первом модуле ввести цифру, во втором - заглавную букву латинского алфавита с порядковым номером, соответствующим этой цифре

Машинно-зависимые языки программирования

Лабораторная работа №8

“Программирование под x86/x86-64. Обработка строк”

Справочная информация

Соглашения о вызовах

Соглашение о вызове — формализация правил вызова подпрограмм, которое должно включать:

- способ передачи параметров;
- способ возврата результата из функции;
- способ возврата управления.

Соглашения о вызовах определяются в рамках отдельных языков высокого уровня, а также - различных программных API, в т. ч. API операционных систем.

Соглашение о вызове Си

cdecl32 — соглашение о вызовах, используемое компиляторами для языка Си на 32-разрядных системах.

1. Аргументы функций передаются через стек, справа налево.
2. Аргументы, размер которых меньше 4-х байт, расширяются до 4-х байт.
3. Очистку стека производит вызывающая программа.
4. Возврат параметров 1, 2, 4 байта (целые числа, указатели) - через `eax`.
5. Возврат больших структур, массивов, строк - указателем через `eax`.

Перед вызовом функции вставляется код, выполняющий следующие действия:

- сохранение значений регистров, используемых внутри функции;
- запись в стек аргументов функции.

После вызова функции вставляется код, выполняющий следующие действия:

- очистка стека;
- восстановление значений регистров.

В 64-разрядных системах могут применяться другие соглашения.

Ассемблерная вставка

Составной оператор языка высокого уровня, телом которого является код на языке ассемблера.

Пример вставки на C++ для Visual Studio 2019:

```
#include <iostream>
```

```

int main()
{
    int i;
    __asm {
        mov eax, 5;
        mov i, eax;
    }
    std::cout << i;
    return 0;
}

```

Возможности Visual Studio

В процессе отладки доступна возможность получения дизассемблированного кода программы (не только ассемблерного, но и на Си):

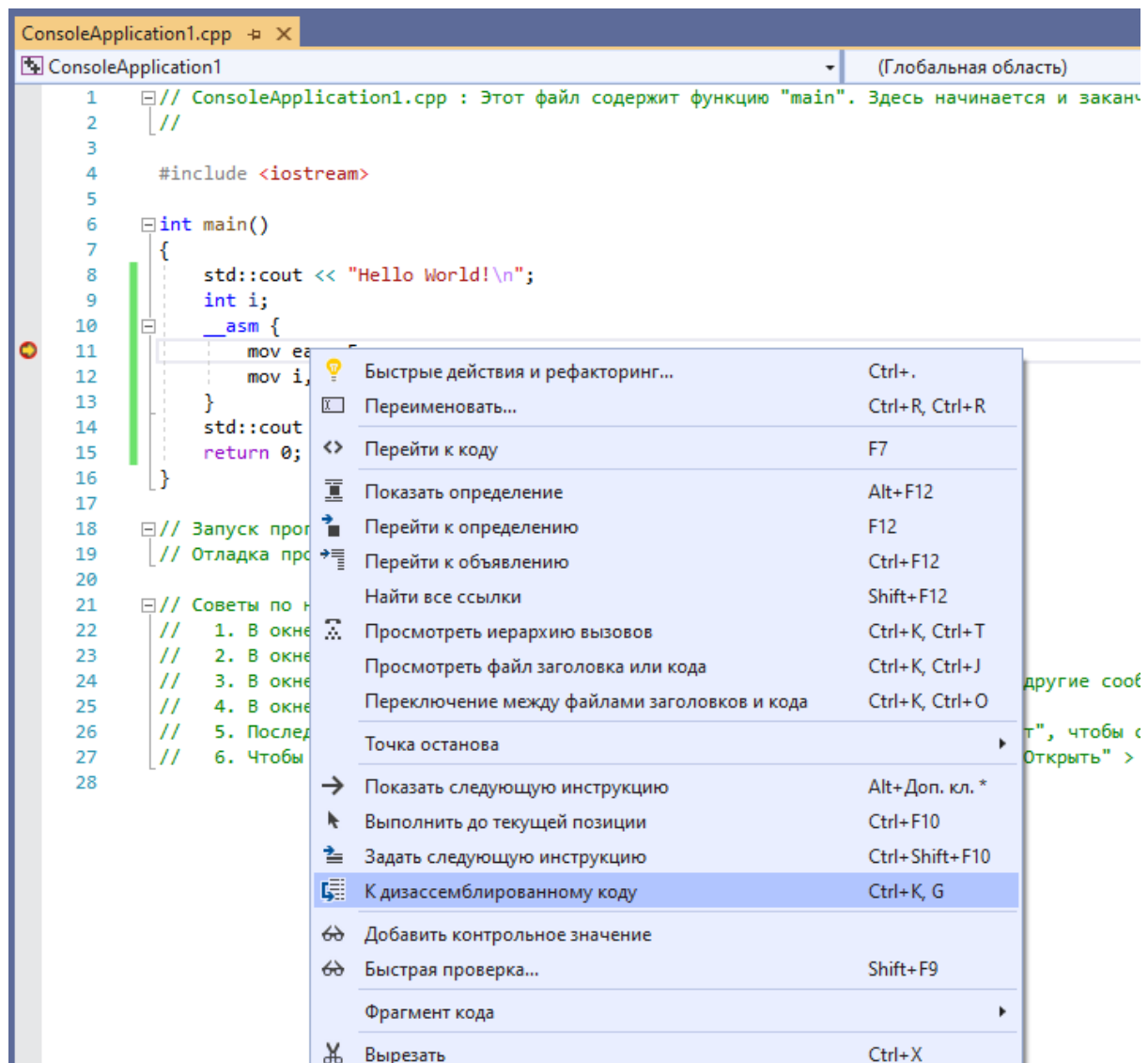


Рисунок 1. Дизассемблирование

Дизассемблированный код - X ConsoleApplication1.cpp

Адрес: main(void)

Параметры просмотра

```

{
008624B0 push     ebp
008624B1 mov      ebp,esp
008624B3 sub      esp,0D0h
008624B9 push     ebx
008624BA push     esi
008624BB push     edi
008624BC lea      edi,[ebp-0D0h]
008624C2 mov      ecx,34h
008624C7 mov      eax,0CCCCCCCCh
008624CC rep stos dword ptr es:[edi]
008624CE mov      eax,dword ptr [__security_cookie (086C004h)]
008624D3 xor      eax,ebp
008624D5 mov      dword ptr [ebp-4],eax
      std::cout << "Hello World!\n";
008624D8 push     offset string "Hello World!\n" (0869B30h)
008624DD mov      eax,dword ptr [__imp__cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A (086D0CCh)]
008624E2 push     eax
008624E3 call     std::operator<<<std::char_traits<char> > (086120Dh)
008624E8 add      esp,8
      int i;
      __asm {
008624EB mov      eax,5;
008624EB mov      eax,5
008624F0 mov      mov i, eax;
008624F0 mov      dword ptr [i],eax
      }
      std::cout << i;
008624F3 mov      esi,esp
008624F5 mov      eax,dword ptr [i]
008624F8 push     eax
008624F9 mov      ecx,dword ptr [__imp__cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A (086D0CCh)]
008624FF call     dword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (086D09Ch)]
00862505 cmp      esi,esp
00862507 call     __RTC_CheckEsp (086127Bh)

```

Рисунок 2. Дизассемблированный код программы

Под каждой строкой кода на Си(Си++) показан дизассемблированный машинный код, в который эта строка была скомпилирована. Например, на рис. 2 видно, что оператору << соответствует 5-7 машинных команд, включая передачу параметров через стек и вызов библиотечной функции.

Примечание: если пункт “К дизассемблированному коду” недоступен, может потребоваться выбрать пункт “Параметры...” в меню “Отладка” и установить флажок “Включить отладку на уровне адреса.”

Также в Visual Studio существует возможность включения в проект asm-файлов. Для этого требуется:

1. в Обзорщике решений открыть контекстное меню проекта и выбрать “Зависимости сборки” -> “Настройки сборки...” и в открывшемся окне установить флажок напротив строки `masm`
2. через Проводник (или другой файловый менеджер) создать файл `.asm` в каталоге с исходным кодом
3. в Обзорщике решений добавить созданный файл в группу “Исходные файлы”

Например, создадим проект с кодом на Си++:

```
#include <iostream>
```

```
extern "C"
```

```

{
    void testAsm(); // подключение в код на Си/Си++ функции
                    // на другом языке программирования,
                    // выполненной в соответствии с соглашениями
                    // о вызовах Си
}

int main()
{
    int i;
    __asm {
        mov eax, 5;
        mov i, eax;
    }
    std::cout << i;
    testAsm();
    __asm {
        mov i, eax;
    }
    std::cout << i;
    return 0;
}

```

и файлом test.asm

```

.686
.MODEL FLAT, C
.STACK

.CODE

testAsm PROC
    mov eax, 7
    ret
testAsm ENDP
END

```

Такая программа выведет в консоль символы 57: первая цифра задана в ассемблерной вставке, вторая - в отдельном файле.

Практическое задание

Написать программу на Си/Си++, которая вызывает 2 подпрограммы на ассемблере:

- первая принимает 1 параметр - указатель на строку, определяет длину строки и выполнена в виде ассемблерной вставки;
- вторая копирует строку с адреса, заданного одним указателем, по адресу, заданному другим указателем, и реализована в отдельном asm-файле. Функция должна принимать 3 параметра: два указателя и длину строки. Про

расположение указателей в памяти и расстояние между ними заранее ничего не известно (первая строка может начинаться раньше второй или наоборот; строки могут перекрываться).

Подпрограммы должны соответствовать соглашению о вызовах языка Си и использовать команды обработки строк с префиксом повторения.

Лабораторная работа должна быть выполнена под 32- или 64-разрядной системой. Допускается использование Visual Studio, Code::Blocks или любой другой среды под Windows либо gcc/g++ под Linux.

Машинно-зависимые языки программирования

Лабораторная работа №9

“Математический сопроцессор”

Практическое задание

1. Изучить скорость выполнения операций над вещественными числами на примерах сложения и умножения 32-разрядных (float), 64-разрядных (double), 80-разрядных, long double:
 - на C/C++ с генерацией инструкций для сопроцессора (опция -m80387)
 - на C/C++ с генерацией инструкций для сопроцессора (опция -mpo-80387)
 - с использованием ассемблерной вставки и команд работы с сопроцессором.Проанализировать дизассемблированный код, генерируемый gcc, для вариантов 1 и 2.
2. Сравнить точность вычислений $\sin \pi$ и $\sin(\pi/2)$ для приближённых значений 3.14, 3.141596 и значения, загружаемого командой сопроцессора.

Машинно-зависимые языки программирования


Лабораторная работа №10

“Streaming SIMD Extensions (SSE), Advanced Vector Extensions (AVX)”

Практическое задание

Реализовать скалярное произведение векторов, или множества пар векторов, или другую операцию над векторами, или умножение матриц, или другую операцию над матрицами в виде ассемблерной вставки с использованием расширений процессора x86-64.

Сравнить производительность решения с кодом, написанным на C/C++.



Машинно-зависимые языки программирования, лекция 1

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2021 г.



Организация курса

- видео-, аудиозапись и фотосъёмка на занятиях запрещены
- 2 модуля + экзамен
- 8 лекций, ~12 лабораторных работ
- 38 часов самостоятельной подготовки (по учебному плану)

Литература

- Зубков С. В. "Assembler. Для DOS, Windows и Unix"



Цели и программа курса

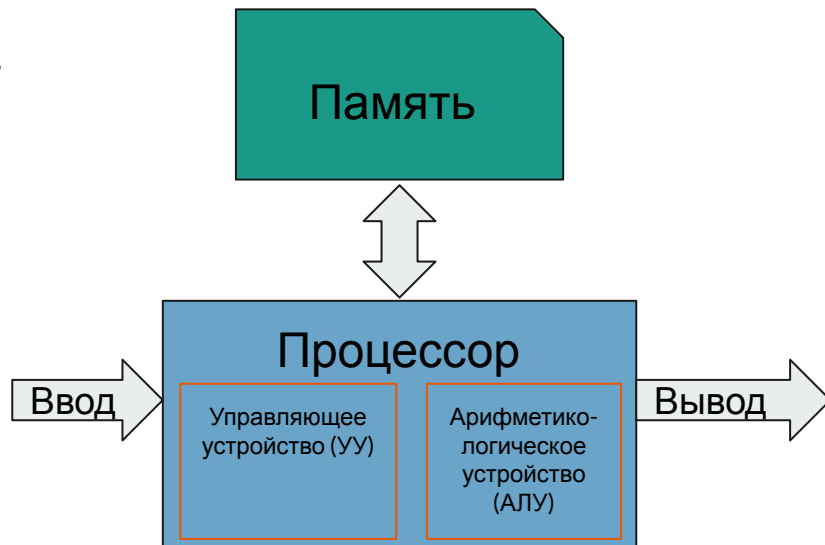
- Изучение низкоуровневого устройства ЭВМ
- Понимание исполнения программ на аппаратном уровне. Работа процессора
- Умение составлять и читать программы на языках низкого уровня, включая:
 - написание программы на низкоуровневом языке “с нуля”;
 - взаимодействие программного кода с устройствами;
 - использование расширений процессоров;
 - отладку и реверс-инжиниринг исполняемых файлов.

История создания ЭВМ. Появление вычислителей общего назначения. Архитектура фон Неймана

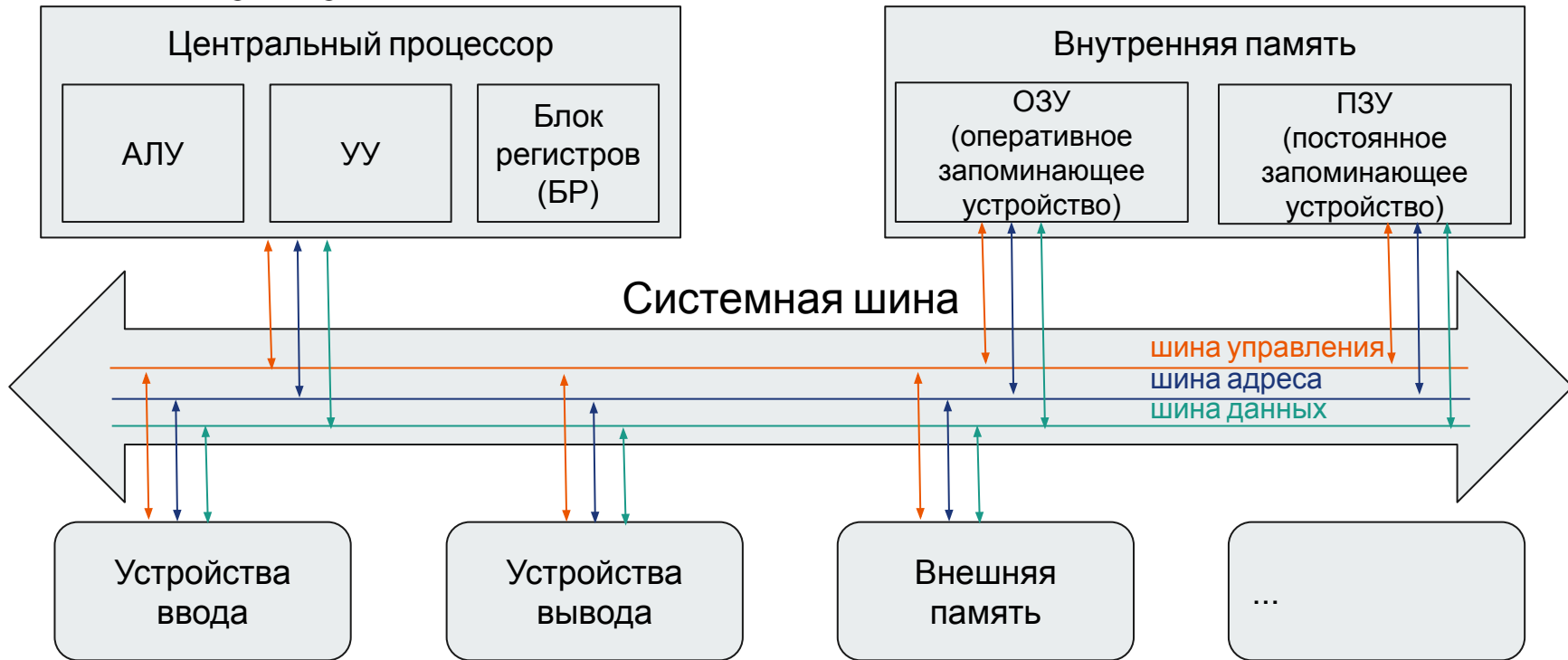
От решения частных вычислительных задач - к универсальным системам

Принципы фон Неймана:

1. Использование двоичной системы счисления в вычислительных машинах.
2. Программное управление ЭВМ.
3. Память компьютера используется не только для хранения данных, но и программ.
4. Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.
5. Возможность условного перехода в процессе выполнения программы.



Структурная схема ЭВМ



Память. Единица адресации.

Минимальная адресуемая единица памяти - байт:

- 8 бит
- $2^8=256$ значений (0..255)
- $8 = 2^3$
- $256 = 2^8=10_{16}^2=100_{16}$

Машинное слово - машинно-зависимая величина, измеряемая в битах, равная разрядности регистров и шины данных

Параграф - 16 байт

ASCII (*аскú*) - *American standard code for information interchange*, США, 1963.

ASCII Code Chart																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- 7-битная кодировка (в расширенном варианте - 8-битная)
- первые 32 символа - непечатные (служебные)
- старшие 128 символов 8-битной кодировки - национальные языки, псевдографика и т. п.



Системы счисления

Двоичная (binary)

- 0, 1, 10, 11, 100, 101...
- $2^8 = 256$
- $2^{10} = 1024$
- $2^{16} = 65536$
- Суффикс - b. Пример: 1101b

Шестнадцатеричная (hexadecimal)

- 0, 1, ..., 8, 9, A, B, C, D, E, F, 10, 11, 12, ..., 19, 1A, 1B, ...
- $2^4 = 10_{16}$
- $2^8 = 100_{16}$
- $2^{16} = 10000_{16}$
- Суффикс - h (10h - 16). Некоторые компиляторы требуют префикса 0x (0x10)

$$101101101111000_2 = B6F8_{16}$$



Представление отрицательных чисел

Знак - в старшем разряде (0 - "+", 1 - "-").

Возможные способы:

- прямой код
- обратный код (инверсия)
- **дополнительный код (инверсия и прибавление единицы)**

Примеры доп. кода на 8-разрядной сетке

-1:

1. 00000001
2. 11111110
3. 11111111

Смысл: $-1 + 1 = 0$ (хоть и с переполнением):

$$11111111 + 1 = (1)\underline{00000000}$$

-101101:

1. 00101101
2. 11010010
3. 11010011



Виды современных архитектур ЭВМ

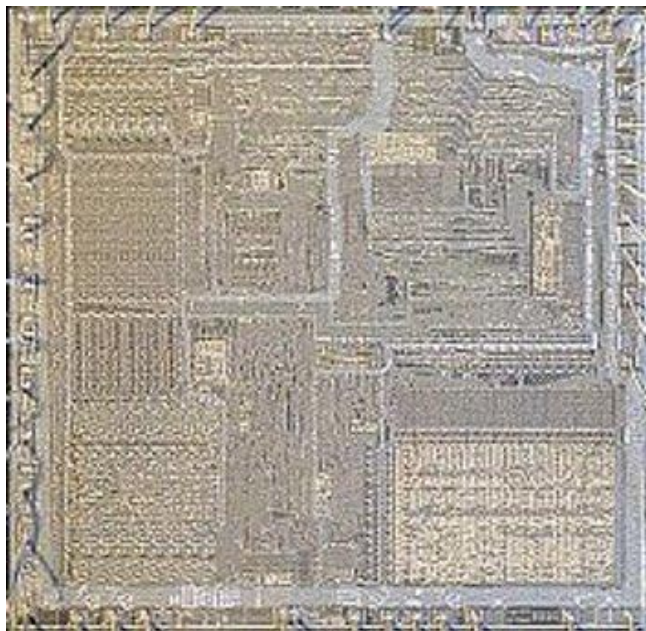
- x86-64 8086 (16-разр.) ➡ x86 (32-разр.) ➡ x86-64 (64-разр.)
- ARM
- IA64
- MIPS (включая Байкал)
- Эльбрус

Семейство процессоров x86 и x86-64

- Микропроцессор 8086: 16-разрядный, 1978 г., 5-10 МГц, 3000 нм
- Предшественники: 4004 - 4-битный, 1971 г.; 8008 - 8-битный, 1972 г.; 8080 - 1974 г.
- Требует микросхем поддержки
- 80186 - 1982 г., добавлено несколько команд, интегрированы микросхемы поддержки
- 80286 - 1982 г., 16-разрядный, добавлен защищённый режим
- 80386, 80486, Pentium, Celeron, AMD, ... - 32-разрядные, повышение быстродействия и расширение системы команд
- x86-64 (x64) - семейство с 64-разрядной архитектурой
- Отечественный аналог - K1810BM86, 1985 г.

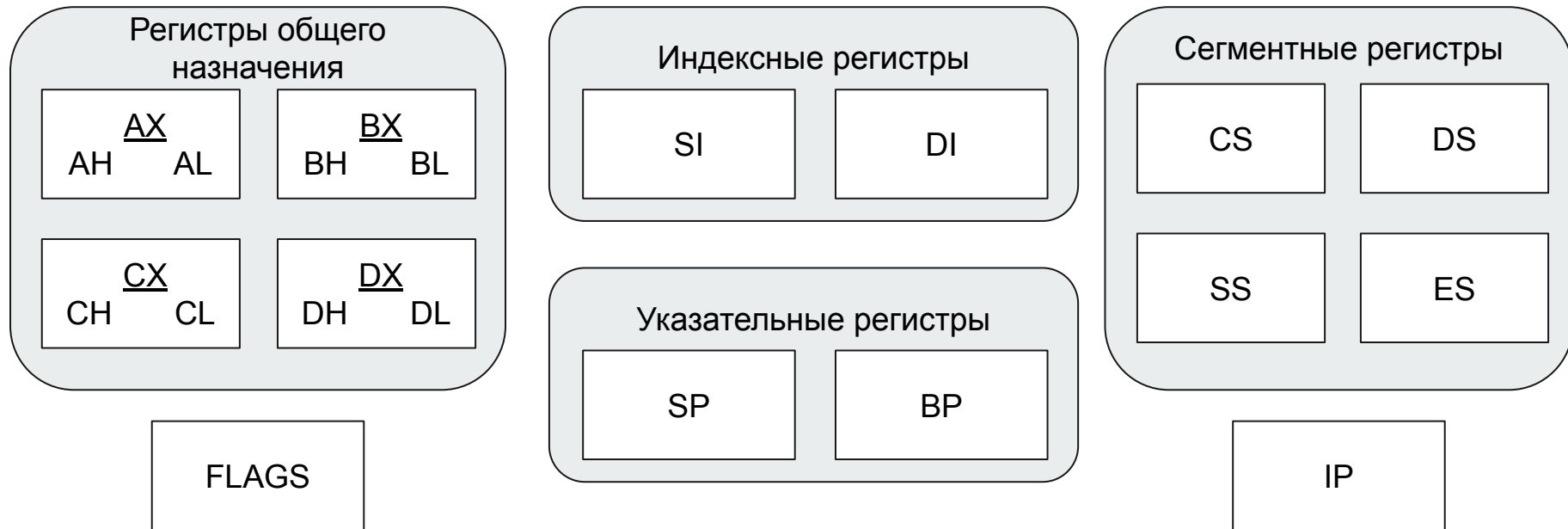


Устройство 8086



			MAX MODE	(MIN MODE)
GND	1	40	U _{CC}	
AD14	2	39	AD15	
AD13	3	38	A16/S3	
AD12	4	37	A17/S4	
AD11	5	36	A18/S5	
AD10	6	35	A19/S6	
AD9	7	34	BHE/S7	
AD8	8	33	MN/MX	
AD7	9	32	RD	
AD6	10	31	RQ/GT0	(HOLD)
AD5	11	30	RQ/GT1	(HLDA)
AD4	12	29	LOCK	(WR)
AD3	13	28	S2	(M/IO)
AD2	14	27	S1	(DT/R)
AD1	15	26	S0	(DEN)
AD0	16	25	QS0	(ALE)
NMI	17	24	QS1	(INTA)
INTR	18	23	TEST	
CLK	19	22	READY	
GND	20	21	RESET	

Архитектура 8086 с точки зрения программиста (структура блока регистров)





Язык ассемблера

Машинная команда - инструкция (в двоичном коде) из аппаратно определённого набора, которую способен выполнять процессор.


Машинный код - система команд конкретной вычислительной машины, которая интерпретируется непосредственно процессором.

Язык ассемблера - машинно-зависимый язык программирования низкого уровня, команды которого прямо соответствуют машинным командам.



Исполняемые файлы. Компиляция. Линковка

- Исполняемый файл - файл, содержащий программу в виде, в котором она может быть исполнена компьютером (то есть в машинном коде).
- Получение исполняемых файлов обычно включает в себя 2 шага: компиляцию и линковку.
- Компилятор - программа для преобразования исходного текста другой программы на определённом языке в объектный модуль.
- компоновщик (линковщик, линкер) - программа для связывания нескольких объектных файлов в исполняемый.



Исполняемые файлы. Запуск программы.

Отладчик

- В DOS и Windows - расширения .EXE и .COM
- Последовательность запуска программы операционной системой:
 1. Определение формата файла.
 2. Чтение и разбор заголовка.
 3. Считывание разделов исполняемого модуля (файла) в ОЗУ по необходимым адресам.
 4. Подготовка к запуску, если требуется (загрузка библиотек).
 5. Передача управления на точку входа.
- Отладчик - программа для автоматизации процесса отладки. Может выполнять трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать или удалять контрольные точки или условия остановки.



“Простейший” формат исполняемого файла

.COM (command) - простейший формат исполняемых файлов DOS и ранних версий Windows:

- не имеет заголовка;
- состоит из одной секции, не превышающей 64 Кб;
- загружается в ОЗУ без изменений;
- начинает выполняться с 1-го байта (точка входа всегда в начале).

Последовательность запуска COM-программы:

1. Система выделяет свободный *сегмент* памяти нужного размера и заносит его адрес во все сегментные регистры (CS, DS, ES, FS, GS, SS).
2. В первые 256 (100h) байт этого сегмента записывается служебная структура DOS, описывающая программу - PSP.
3. Непосредственно за ним загружается содержимое COM-файла без изменений.
4. Указатель стека (регистр SP) устанавливается на конец сегмента.
5. В стек записывается 0000h (начало PSP - адрес возврата для возможности завершения командой ret).
6. Управление передаётся по адресу CS:0100h.



Классификация команд процессора 8086

- Команды пересылки данных
- Арифметические и логические команды
- Команды переходов
- Команды работы с подпрограммами
- Команды управления процессором



Команда пересылки данных MOV

MOV <приёмник>, <источник>

Источник: непосредственный операнд (константа, включённая в машинный код), РОН, сегментный регистра, переменная (ячейка памяти).

Приёмник: РОН, сегментный регистр, переменная (ячейка памяти).

- MOV AX, 5
- MOV BX, DX
- MOV [1234h], CH
- MOV DS, AX

- 
- MOV [0123h], [2345h]
 - MOV DS, 1000h



Целочисленная арифметика (основные команды)

- ADD <приёмник>, <источник> - выполняет арифметическое сложение приёмника и источника. Сумма помещается в приёмник, источник не изменяется.
- SUB <приёмник>, <источник> - арифметическое вычитание источника из приёмника.
- MUL <источник> - беззнаковое умножение. Умножаются источник и AL/AX, в зависимости от размера источника. Результат помещается в AX либо DX:AX.
- DIV <источник> - целочисленное беззнаковое деление. Делится AL/AX на источник. Результат помещается в AL/AX, остаток - в AH/DX.
- INC <приёмник> - инкремент на 1
- DEC <приёмник> - декремент на 1



Побитовая арифметика (основные команды)

- AND <приёмник>, <источник> - побитовое “И”. AND al, 00001111b
- OR <приёмник>, <источник> - побитовое “ИЛИ”. OR al, 00001111b
- XOR <приёмник>, <источник> - побитовое исключающее “ИЛИ”. XOR AX, AX
- NOT <приёмник> - инверсия



Команда безусловной передачи управления JMP

JMP <операнд>

- Передаёт управление в другую точку программы (на другой адрес памяти), не сохраняя какой-либо информации для возврата.
- Операнд - непосредственный адрес (вставленный в машинный код), адрес в регистре или адрес в переменной.



Команда NOP (no operation)

- Ничего не делает
- Занимает место и время
- Размер - 1 байт, код - 90h
- Назначение - задержка выполнения либо заполнение памяти, например, для *выравнивания*

Пример

...

XOR AX, AX

MOV BX, 5

label1:

INC AX

ADD BX, AX

JMP label 1



AX	0000	SI	0000	CS	19F5	IP	0100
BX	0000	DI	0000	DS	19F5		
CX	0024	BP	0000	ES	19F5	HS	19F5
DX	0000	SP	FFFE	SS	19F5	FS	19F5
CMD >							
0100	33C0	XOR	AX,AX				
0102	BB0500	MOV	BX,0005				
0105	40	INC	AX				
0106	03D8	ADD	BX,AX				
0108	EBFB	JMP	0105				
010A	BA1401	MOV	DX,0114				
010D	CD21	INT	21				
010F	B44C	MOV	AH,4C				

Взаимодействие программы с внешней средой (ОС, пользователь, ...)

Прерывания - аппаратный механизм для приостановки выполнения текущей программы и передачи управления специальной программе - обработчику прерывания.

Основные виды:

- аппаратные
- программные

int <номер> - вызов (генерация прерывания)

21h - прерывание DOS, предоставляет прикладным программам около 70 различных функций (ввод, вывод, работа с файлами, завершение программы и т.д.)

Номер функции прерыванию 21h передаётся через регистр АН. Параметры для каждой функции передаются собственным способом, он описан в документации. Там же описан способ возврата результата из функции в программу.



Память в реальном режиме работы процессора

Реальный режим работы - режим совместимости современных процессоров с 8086.

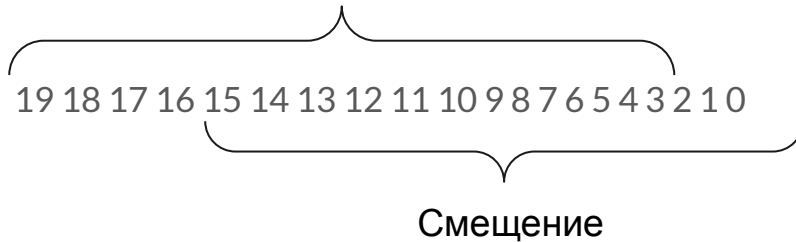
Доступен 1 Мб памяти (2^{20} байт), то есть разрядность шины адреса - 20 разрядов.

Физический адрес получается **сложением** адреса **начала сегмента** (на основе сегментного регистра) и **смещения**.

Сегментный регистр хранит в себе **старшие 16 разрядов** (из 20) адреса начала сегмента. 4 младших разряда в адресе начала сегмента всегда нулевые. Говорят, что сегментный регистр содержит в себе **номер параграфа начала сегмента**.

Память в реальном режиме работы процессора - пример

Номер параграфа начала сегмента



[SEG]:[OFFSET] => физический адрес:

1. SEG необходимо побитово сдвинуть на 4 разряда влево (или умножить на 16, что тождественно)
2. К результату прибавить OFFSET

5678h:1234h =>

56780
+1234
579B4


Вычисление физического адреса выполняется процессором аппаратно, без участия программиста.

Распространённые пары регистров: CS:IP, DS:BX, SS:SP



Структура памяти программы. Виды сегментов. Назначение отдельных сегментных регистров

- Сегмент кода - регистр CS. Командой MOV изменить невозможно, меняется автоматически по мере выполнения команд.
- Сегмент данных. Основной регистр - DS, при необходимости дополнительных сегментов данных задействуются ES, FS, GS.
- Сегмент стека - регистр SS

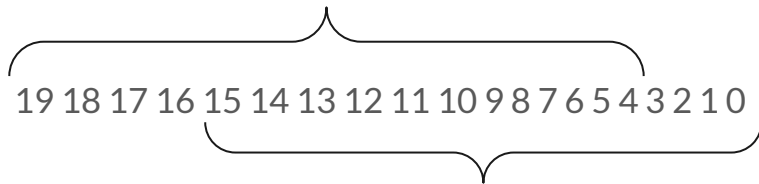


Машинно-зависимые языки программирования, лекция 2

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2021 г.

Память в реальном режиме работы процессора - пример

Номер параграфа начала сегмента



[SEG]:[OFFSET] => физический адрес:

1. SEG необходимо побитово сдвинуть на 4 разряда влево (или умножить на 16, что тождественно)
2. К результату прибавить OFFSET

5678h:1234h =>

$$\begin{array}{r} 56780 \\ + 1234 \\ \hline 579B4 \end{array}$$

Вычисление физического адреса выполняется процессором аппаратно, без участия программиста.

Распространённые пары регистров: CS:IP, DS:BX, SS:SP

Память 8086 (20-разрядная адресация)

00000	
00001	
00002	
00003	
00004	
00005	
00006	
00007	
00008	
00009	
0000A	
0000B	
0000C	
0000D	
0000E	
0000F	

Параграф 0

00010	
00011	
00012	
00013	
00014	
00015	
00016	
00017	
00018	
00019	
0001A	
0001B	
0001C	
0001D	
0001E	
0001F	

Параграф 1

...

FFFF0	
FFFF1	
FFFF2	
FFFF3	
FFFF4	
FFFF5	
FFFF6	
FFFF7	
FFFF8	
FFFF9	
FFFFA	
FFFFB	
FFFC	
FFFD	
FFFE	
FFFF	

Параграф FFFF

Сегментная модель памяти 8086

Максимальный размер
сегмента с начальной
сегментной частью адреса
8AFE

00000	
00001	
...	
00010	
00011	
...	
8AFE0	
8AFE1	
8AFE2	
...	
9AFDF	
9AFE0	
...	
FFFFF	

$8AFE:0000 = 8AFE0$

$8AFE:001F = 8AFFF$

$8AFE:1000 = 8BFE0$

$8AFE:FFFF = 8AFE0 + 10000 - 1 = 9AFDF$



Логическая структура памяти. Сегменты

- Сегмент кода (регистр CS)
- Сегменты данных (основной регистр - DS, для дополнительных сегментов - ES, FS, GS)
- Сегмент стека (регистр SS)



Команда организации цикла LOOP

LOOP метка

Уменьшает регистр CX на 1 и выполняет переход на метку, если CX не равен нулю.

Метка **не** может быть дальше -128..127 байт от команды.



Структура программы на ассемблере

(Зубков С. В. Assembler для DOS, Windows, Unix, глава 3)

- Модули (файлы исходного кода)
 - Сегменты (описание блоков памяти)
 - команды процессора;
 - инструкции описания структур данных, выделения памяти для переменных и констант;
 - макроопределения.

Полный формат строки:

метка команда / директива операнды ; комментарий



Метки

В коде

- Пример:

```
mov cx, 5

label1:

    add ax, bx

    loop label1
```

- Метки обычно используются в командах передачи управления

В данных

- label
 - *метка label тип*
 - Возможные типы: BYTE, WORD, DWORD, FWORD, QWORD, TBYTE, NEAR, FAR.
- EQU, =
 - *label EQU выражение*
 - макрос
 - вычисляет выражение в правой части и приравнивает его метке

Директивы выделения памяти

- Директива - инструкция ассемблеру, влияющая на процесс компиляции и не являющаяся командой процессора. Обычно не оставляет следов в формируемом машинном коде.
- Псевдокоманда - директива ассемблера, которая приводит к включению данных или кода в программу, но не соответствующая никакой команде процессора.
- Псевдокоманды определения данных указывают, что в соответствующем месте располагается переменная, резервируют под неё место заданного типа, заполняют значением и ставят в соответствие метку.
- Виды: DB (1), DW (2), DD (4), DF (6), DQ (8), DT (10).
- Примеры:
 - `a DB 1`
 - `float_number DD 3.5e7`
 - `text_string DB 'Hello, world!'`
- DUP - заполнение повторяющимися данными
- ? - неинициализированное значение
- `uninitialized DW 512 DUP(?)`



Описание сегментов программы

- Любая программа состоит из сегментов
- Виды сегментов:
 - сегмент кода
 - сегмент данных
 - сегмент стека
- Описание сегмента - директива SEGMENT:

```
имя SEGMENT [READONLY] [выравнивание] [тип] [разрядность] ['класс']
```

```
...
```

```
имя ENDS
```




Параметры директивы SEGMENT

Выравнивание

- BYTE
- WORD
- DWORD
- **PARA**
- PAGE

Тип

- PUBLIC
- STACK
- COMMON
- AT
- **PRIVATE**

Класс - любая метка, взятая в одинарные кавычки. Сегменты одного класса будут расположены в памяти друг за другом.



Модели памяти

`.model` модель, язык, модификатор

- Модели:
 - TINY - один сегмент на всё
 - SMALL - код в одном сегменте, данные и стек - в другом
 - COMPACT - допустимо несколько сегментов данных
 - MEDIUM - код в нескольких сегментах, данные - в одном
 - LARGE, HUGE
- Язык - C, PASCAL, BASIC, SYSCALL, STDCALL. Для связывания с ЯВУ и вызова подпрограмм.
- Модификатор - **NEARSTACK**/FARSTACK
- Определение модели позволяет использовать сокращённые формы директив определения сегментов.



Завершение описания модуля. Точка входа

.

.

.

END [точка_входа]

- точка_входа - имя метки, объявленной в сегменте кода и указывающее на команду, с которой начнётся исполнение программы.
- Если в программе несколько модулей, только один может содержать точку входа.



Сегментный префикс. Директива ASSUME

- Для обращения к переменной процессору необходимо знать обе составляющие адреса: и сегментную, и смещение.

Пример полной записи - DS:Var1

- Директива ASSUME регистр:имя сегмента устанавливает значение сегментного регистра по умолчанию

```
Data1 SEGMENT WORD 'DATA'  
Var1 DW 0  
Data1 ENDS
```

```
Data2 SEGMENT WORD 'DATA'  
Var2 DW 0  
Data2 ENDS
```

```
Code SEGMENT WORD 'CODE'  
    ASSUME CS:Code  
ProgramStart:  
    mov ax,Data1  
    mov ds,ax  
    ASSUME DS:Data1  
    mov ax,Data2  
    mov es,ax  
    ASSUME ES:Data2  
    mov ax,[Var2]  
  
    .  
    .  
    .  
Code ENDS  
END ProgramStart
```



Прочие директивы

- Задание набора допустимых команд: .8086, .186, .286, ..., .586, .686, ...
- Управление программным счётчиком:
 - ORG значение
 - EVEN
 - ALIGN значение
- Глобальные объявления
 - public, comm, extrn, global
- Условное ассемблирование

IF выражение

...

ELSE

...

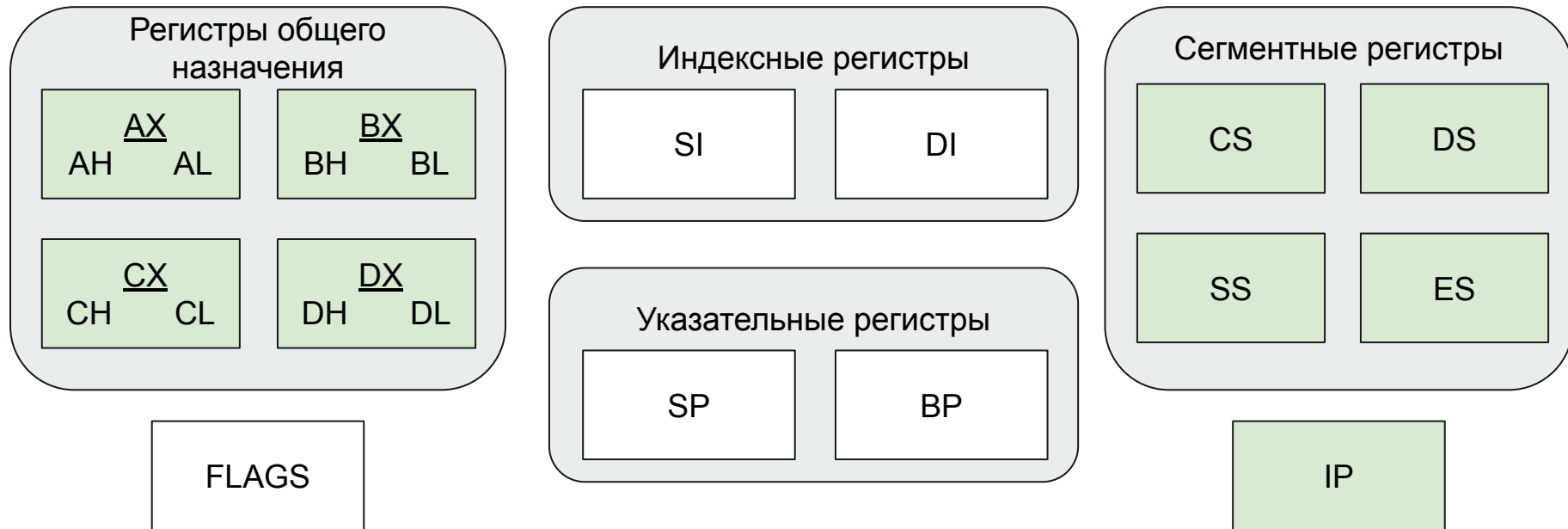
ENDIF



Виды переходов для команды JMP

- short (короткий) -128 .. +127 байт
- near (ближний) в том же сегменте (без изменения регистра CS)
- far (дальний) в другой сегмент (с изменением значения в регистре CS)
- Для короткого и ближнего переходов непосредственный операнд (константа) прибавляется к IP
- Операнды - регистры и переменные заменяют старое значение в IP (CS:IP)

Архитектура 8086 с точки зрения программиста (структура блока регистров)





Индексные регистры SI и DI

- SI - source index (индекс источника)
- DI - destination index (индекс приёмника)
- Могут использоваться в большинстве команд, как регистры общего назначения
- Применяются в специфических командах поточной обработки данных

Способы адресации

(Зубков С. В. Assembler для DOS, Windows, Unix, глава 2)

- Регистровая адресация (`mov ax, bx`)
- Непосредственная адресация (`mov ax, 2`)
- Прямая адресация (`mov ax, ds:0032`)
- Косвенная адресация (`mov ax, [bx]`). В 8086 допустимы BX, BP, SI, DI
- Адресация по базе со сдвигом (`mov ax, [bx]+2`; `mov ax, 2[bx]`).
- Адресация по базе с индексированием (допустимы BX+SI, BX+DI, BP+SI, BP+DI):
 - `mov ax, [bx+si+2]` - `mov ax, [bx][si]+2`
 - `mov ax, [bx+2][si]` - `mov ax, [bx][si+2]`
 - `mov ax, 2[bx][si]`



Регистр FLAGS

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

- CF (carry flag) - флаг переноса
- PF (parity flag) - флаг чётности
- AF (auxiliary carry flag) - вспомогательный флаг переноса
- ZF (zero flag) - флаг нуля
- SF (sign flag) - флаг знака
- TF (trap flag) - флаг трассировки
- IF (interrupt enable flag) - флаг разрешения прерываний
- DF (direction flag) - флаг направления
- OF (overflow flag) - флаг переполнения
- IOPL (I/O privilege flag) - уровень приоритета ввода-вывода
- NT (nested task) - флаг вложенности задач



Команда сравнения CMP

CMP <приёмник>, <источник>

- Источник - число, регистр или переменная
- Приёмник - регистр или переменная; не может быть переменной одновременно с источником
- Вычитает источник из приёмника, результат никуда не сохраняется, выставляются флаги CF, PF, AF, ZF, SF, OF



Команды условных переходов J..

(Зубков С. В. Assembler для DOS, Windows, Unix, глава 2)

- Переход типа short или near
- Обычно используются в паре с CMP
- Термины “выше” и “ниже” - при сравнении беззнаковых чисел
- Термины “больше” и “меньше” - при сравнении чисел со знаком



Виды условных переходов (часть 1)

Команда	Описание	Состояние флагов для выполнения перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнение	OF = 0
JS	Есть знак	SF = 1
JNS	Нет знака	SF = 0
JE, JZ	Если равно/если ноль	ZF = 1
JNE, JNZ	Не равно/не ноль	ZF = 0
JP/JPE	Есть чётность / чётное	PF = 1
JNP/JPO	Нет чётности / нечётное	PF = 0
JCXZ	CX = 0	-



Виды условных переходов (часть 2)

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JB JNAE JC	Если ниже Если не выше и не равно Если перенос	CF = 1	нет
JNB JAE JNC	Если не ниже Если выше или равно Если нет переноса	CF = 0	нет
JBE JNA	Если ниже или равно Если не выше	CF = 1 или ZF = 1	нет
JA JNBE	Если выше Если не ниже и не равно	CF = 0 и ZF = 0	нет



Виды условных переходов (часть 3)

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JL JNGE	Если меньше Если не больше и не равно	SF <> OF	да
JGE JNL	Если больше или равно Если не меньше	SF = OF	да
JLE JNG	Если меньше или равно Если не больше	ZF = 1 или SF <> OF	да
JG JNLE	Если больше Если не меньше и не равно	ZF = 0 и SF = OF	да



Команда TEST

TEST <приёмник>, <источник>

- Аналог AND, но результат не сохраняется
- Выставляются флаги SF, ZF, PF



Прерывания

- Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передаётся программе-обработчику возникшего прерывания.
- Виды прерываний:
 - аппаратные (асинхронные) - события от внешних устройств;
 - внутренние (синхронные) - события в самом процессоре, например, деление на ноль;
 - программные - вызванные командой INT.



Прерывание DOS 21h

- Аналог системного вызова в современных ОС
- Используется наподобие вызова подпрограммы
- Номер функции передаётся через АН




Прерывание DOS - вывод на экран в текстовом режиме

Функция	Назначение	Вход	Выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом \$	-

Прерывание DOS - ввод с клавиатуры

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL - ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL = FF	AL - ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL - ASCII-код символа
08	Считать символ без эха	-	AL - ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры	-	AL=0, если клавиша не была нажата, и FF, если была
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	



Машинно-зависимые языки программирования, лекция 3

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2021 г.



CMOVcc - условная пересылка данных

CMOVcc <приёмник>, <источник>

Условия аналогичны Jcc



XCHG - обмен операндов между собой

XCHG <операнд1>, <операнд2>

Выполняется над двумя регистрами либо регистром и переменной



XLAT/XLATB - трансляция в соответствии с таблицей

XLAT [адрес]

XLATB

Помещает в AL байт из таблицы по адресу DS:BX со смещением относительно начала таблицы, равным AL.

Адрес, указанный в исходном коде, не обрабатывается компилятором и служит в качестве комментария.

Если в адресе явно указан сегментный регистр, он будет использоваться вместо DS.



LEA - вычисление эффективного адреса

LEA <приёмник>, <источник>

Вычисляет эффективный адрес источника и помещает его в приёмник.

Позволяет вычислить адрес, описанный сложным методом адресации.

Иногда используется для быстрых арифметических вычислений:

```
lea bx, [bx+bx*4]
```

```
lea bx, [ax+12]
```

Эти вычисления занимают меньше памяти, чем соответствующие MOV и ADD, и не изменяют флаги.

Двоичная арифметика. ADD, ADC, SUB, SBB

ADD, SUB не делают различий между знаковыми и беззнаковыми числами.

ADC <приёмник>, <источник> - сложение с переносом. Складывает приёмник, источник и флаг CF.

SBB <приёмник>, <источник> - вычитание с займом. Вычитает из приёмника источник и дополнительно - флаг CF.

```
add ax, cx  
adc dx, bx
```

```
sub ax, cx  
sbb dx, bx
```

Арифметические флаги - CF, OF, SF, ZF, AF, PF



IMUL, MUL, IDIV, DIV

Умножение чисел со знаком:

IMUL <источник>

IMUL <приёмник>, <источник>

IMUL <приёмник>, <источник1>, <источник2>

Целочисленное деление со знаком:

IDIV <источник>

Результат округляется в сторону нуля, знак остатка совпадает со знаком делимого.



INC, DEC

INC <приёмник>

DEC <приёмник>

Увеличивает/уменьшает приёмник на 1.

В отличие от ADD, не изменяет CF.


OF, SF, ZF, AF, PF устанавливаются в соответствии с результатом.



NEG - изменение знака

NEG <приёмник>

Переводит число в дополнительный код.



Десятичная арифметика

DAA, DAS, AAA, AAS, AAM, AAD

- Неупакованное двоично-десятичное число - байт от 00h до 09h.
- Упакованное двоично-десятичное число - байт от 00h до 99h (цифры A..F не задействуются).
- При выполнении арифметических операций необходима коррекция:
 - $19h + 1 = 1Ah \Rightarrow 20h$

```
inc al
```

```
daa
```



Логические команды AND, OR, XOR, NOT, TEST

См. первую лекцию



Логический, арифметический, циклический сдвиг. **SAR, SAL, SHR, SHL, ROR, ROL, RCR, RCL**

- SAL тождественна SHL
- SHR зануляет старший бит, SAR - сохраняет (знак)
- ROR, ROL - циклический сдвиг вправо/влево
- RCR, RCL - циклический сдвиг через CF



Операции над битами и байтами

BT, BTR, BTS, BTC, BSF, BSR, SETcc


- BT <база>, <смещение> - считать в CF значение бита из битовой строки
- BTS <база>, <смещение> - установить бит в 1
- BTR <база>, <смещение> - сбросить бит в 0
- BTC <база>, <смещение> - инвертировать бит
- BSF <приёмник>, <источник> - прямой поиск бита (от младшего разряда)
- BSR <приёмник>, <источник> - обратный поиск бита (от старшего разряда)
- SETcc <приёмник> - выставляет приёмник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc



Организация циклов

- LOOP <метка> - уменьшает CX и выполняет "короткий" переход на метку, если CX не равен нулю.
- LOOPE/LOOPZ <метка> - цикл "пока равно"/"пока ноль"
- LOOPNE/LOOPNZ <метка> - цикл "пока не равно"/"пока не ноль"

Декрементируют CX и выполняют переход, если CX не ноль и если выполняется условие (ZF).



Строковые операции: копирование, сравнение, сканирование, чтение, запись

Строка-источник - DS:SI, строка-приёмник - ES:DI.

За один раз обрабатывается один байт (слово).

- MOVS/MOVSБ/MOVSВ <приёмник>, <источник> - копирование
- CMPS/CMPSБ/CMPSВ <приёмник>, <источник> - сравнение
- SCAS/SCASБ/SCASВ <приёмник> - сканирование (сравнение с AL/AX)
- LODS/LODSБ/LODSВ <источник> - чтение (в AL/AX)
- STOS/STOSБ/STOSВ <приёмник> - запись (из AL/AX)

Префиксы: REP/REPE/REPZ/REPNE/REPNZ



Управление флагами

- STC/CLC/CMC - установить/сбросить/инвертировать CF
- STD/CLD - установить/сбросить DF
- LAHF - загрузка флагов состояния в AH
- SAHF - установка флагов состояния из AH
- CLI/STI - запрет/разрешение прерываний (IF)

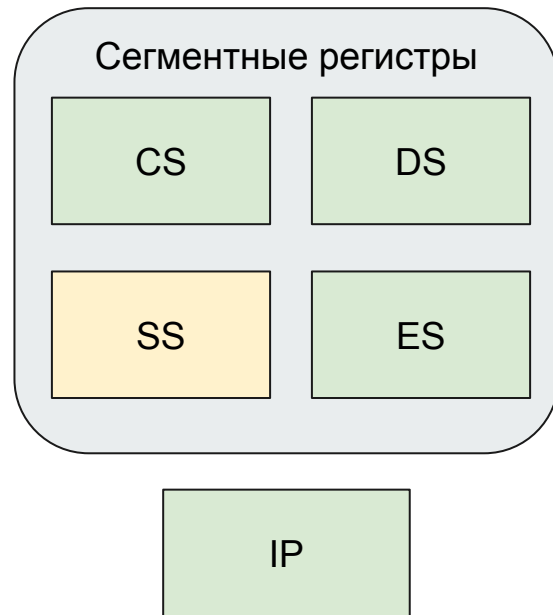
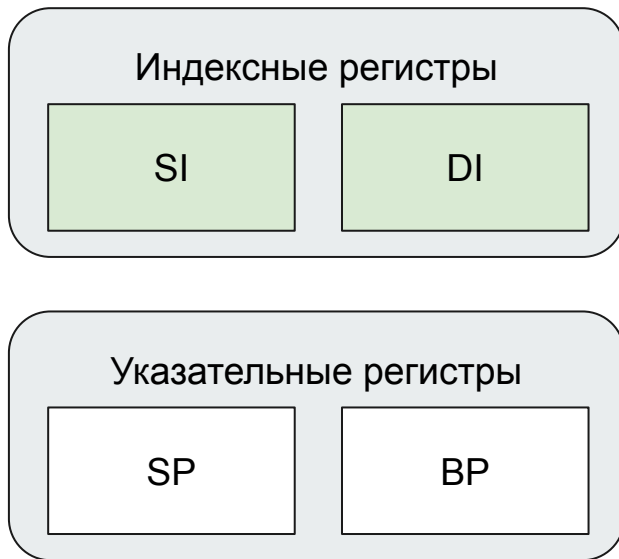
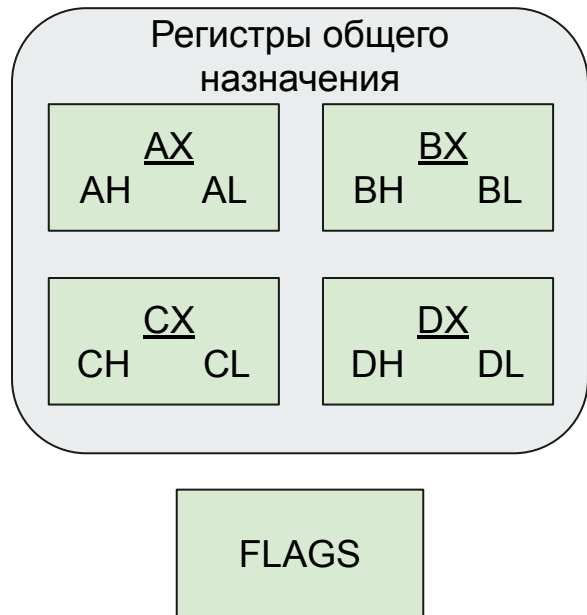


Загрузка сегментных регистров

- LDS <приёмник>, <источник> - загрузить адрес, используя DS
- LES <приёмник>, <источник> - загрузить адрес, используя ES
- LFS <приёмник>, <источник> - загрузить адрес, используя FS
- LGS <приёмник>, <источник> - загрузить адрес, используя GS
- LSS <приёмник>, <источник> - загрузить адрес, используя SS

Приёмник - регистр, источник - переменная

Регистры. Стек






Стек

- LIFO/FILO (last in, first out) - последним пришёл, первым ушёл
- Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний
- SP - указатель на вершину стека
- В x86 стек "растёт вниз", в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента



Команды непосредственной работы со стеком

- PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- POPA - загрузить регистры из стека (SP игнорируется)



CALL - вызов процедуры, RET - возврат из процедуры

CALL <операнд>

- Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передаёт управление на значение аргумента.

RET/RETN/RETF <число>

- Загружает из стека адрес возврата, увеличивает SP
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

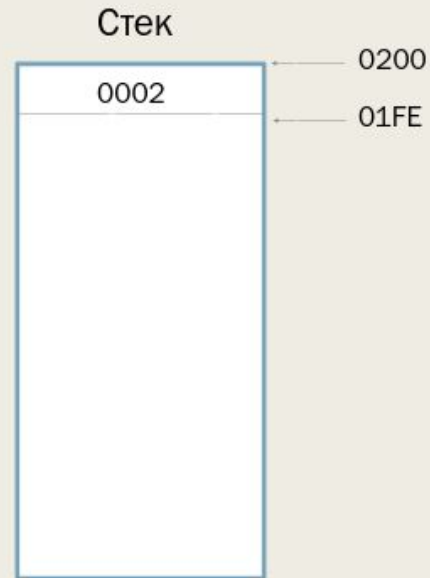


BP – base pointer

- Используется в подпрограмме для сохранения "начального" значения SP
- Адресация параметров
- Адресация локальных переменных

Пример вызова подпрограммы №1

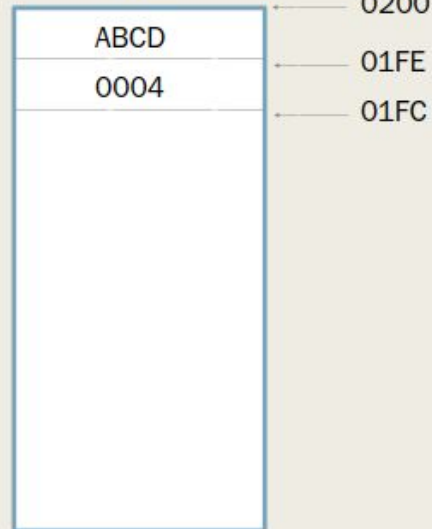
```
0. SP = 0200  
0000: CALL P1    1. SP = 01FE  
0002: MOV BX, AX  
...  
P1:  
0123: MOV AX, 5  
0125: RET        2. SP = 0200
```



Пример вызова подпрограммы N°2

```
                                0. SP = 0200
0000: PUSH ABCDh ;передача параметра
0002: CALL P1                    1. SP = 01FE
0004: POP DX                     2. SP = 01FC
0006: MOV BX, AX                 4. SP = 0200
...
P1:
0123: MOV BP, SP ;ss:[bp] - адрес возврата
      ;ss:[bp+2] - параметр
...
0223: MOV AX, 5
0225: RET                       3. SP = 01FE
```

Стек



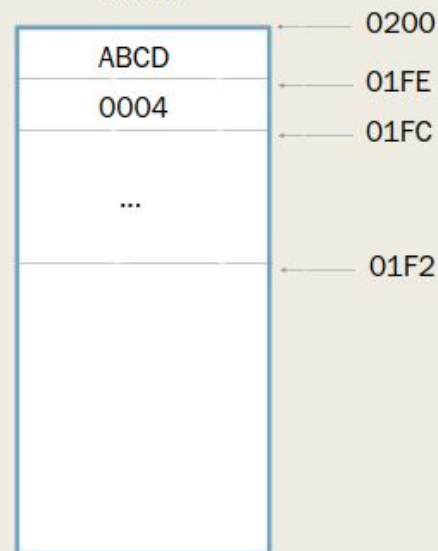
Пример вызова подпрограммы №3


```
0. SP = 0200

0000: PUSH ABCDh ; передача параметра
0002: CALL P1
0004: MOV BX, AX
...
P1:
0123: MOV BP, SP ; ss:[bp] - адрес возврата
; ss:[bp+2] - параметр
0125: SUB SP, 10 ; ss:[bp-1 .. bp-10] - локальные переменные
...
0221: ADD SP, 10
0223: MOV AX, 5
0225: RET 2

3. SP = 01F2
4. SP = 01FC
5. SP = 0200
```

Стек





Машинно-зависимые языки программирования, лекция 4

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2021 г.



Прерывания

- Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передаётся программе-обработчику возникшего прерывания.
- Виды прерываний:
 - аппаратные (асинхронные) - события от внешних устройств;
 - внутренние (синхронные) - события в самом процессоре, например, деление на ноль;
 - программные - вызванные командой `int`.



Маскирование прерываний

Внешние прерывания, в зависимости от возможности запрета, делятся на:

- **маскируемые** — прерывания, которые можно запрещать установкой соответствующего флага;
- **немаскируемые** (англ. Non-maskable interrupt, NMI) — обрабатываются всегда, независимо от запретов на другие прерывания




Таблица векторов прерываний в реальном режиме работы процессора

- Вектор прерывания — номер, который идентифицирует соответствующий обработчик прерываний. Векторы прерываний объединяются в таблицу векторов прерываний, содержащую адреса обработчиков прерываний.
- Располагается в самом начале памяти, начиная с адреса 0.
- Доступно 256 прерываний.
- Каждый вектор занимает 4 байта - полный адрес.
- Размер всей таблицы - 1 Кб.



Срабатывание прерывания

- Сохранение в текущий стек регистра флагов и полного адреса возврата (адреса следующей команды) - 6 байт
- Передача управления по адресу обработчика из таблицы векторов
- *Настройка стека?*
- *Повторная входимость (реентерабельность), необходимость запрета прерываний?*



IRET - возврат из прерывания

- Используется для выхода из обработчика прерывания
- Восстанавливает FLAGS, CS:IP
- При необходимости выставить значение флага обработчик меняет его значение непосредственно в стеке



Перехват прерывания

- Сохранение адреса старого обработчика
- Изменение вектора на "свой" адрес
- Вызов старого обработчика до/после отработки своего кода
- При деактивации - восстановление адреса старого обработчика



Установка обработчика прерывания в DOS

- int 21h
 - AH=35h, AL= номер прерывания - возвращает в ES:BX адрес обработчика (в BX 0000:[AL*4], а в ES - 0000:[AL*4+2].)
 - AH=25h, AL=номер прерывания, DS:DX - адрес обработчика



Некоторые прерывания

- 0 - деление на 0
- 1 - прерывание отладчика, вызывается после каждой команды при флаге TF
- 3 - "отладочное", int 3 занимает 1 байт
- 4 - переполнение при команде INTO (команда проверки переполнения)
- 5 - при невыполнении условия в команде BOUND (команда контроля индексов массива)
- 6 - недопустимая (несуществующая) инструкция
- 7 - отсутствует FPU
- 8 - таймер
- 9 - клавиатура
- 10h - прерывание BIOS



Резидентные программы

- Резидентная программа - та, которая остаётся в памяти после возврата управления DOS
- Завершение через функцию 31h прерывания 21h / прерывание 27h
- DOS не является многозадачной операционной системой
- Резиденты - частичная реализация многозадачности
- Резидентная программа должна быть составлена так, чтобы минимизировать используемую память



Завершение с сохранением в памяти

- **int 27h**
 - DX = адрес первого байта за резидентным участком программы (смещение от PSP)
- **int 21h, ah=31h**
 - AL - код завершения
 - DX - объём памяти, оставляемой резидентной, в параграфах



Порты ввода-вывода


- Порты ввода-вывода - механизм взаимодействия программы, выполняемой процессором, с устройствами компьютера.
- IN - команда чтения данных из порта ввода
- OUT - команда записи в порт вывода
- Пример:

```
IN al, 61h  
OR al, 3  
OUT 61h, al
```



Вопросы к рубежному контролю

1. Регистры общего назначения.
2. Сегментные регистры. Адресация в реальном режиме. Понятие сегментной части адреса и смещения.
3. Регистры работы со стеком.
4. Структура программы. Сегменты.
5. Прерывание 21h. Примеры ввода-вывода.
6. стек. Назначение, примеры использования.
7. Регистр флагов.
8. Команды условной и безусловной передачи управления.
9. Организация многомодульных программ.
10. Подпрограммы. Объявление, вызов.
11. Арифметические команды.
12. Команды побитовых операций.
13. Команды работы со строками.
14. Прерывания. Обработка прерываний.
15. Работа с портами ввода-вывода.



Машинно-зависимые языки программирования, лекция 5

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2021 г.



32-разрядные процессоры (386+)

Производство x86: 1985 - ~2010

32-разрядные:

- Регистры, кроме сегментных
- Шина данных
- Шина адреса ($2^{32} = 4\text{Гб ОЗУ}$)

Режимы работы

8086 (1978 г.) -> 80186 (1982 г.)

-> 80286 (1982 г.) добавлен защищённый режим

-> 80386 (1985 г.) архитектура стала 32-разрядной

-> 80486 (1989 г.) -> Pentium -> ... -> (современные процессоры)

"Реальный" режим (режим совместимости с 8086)

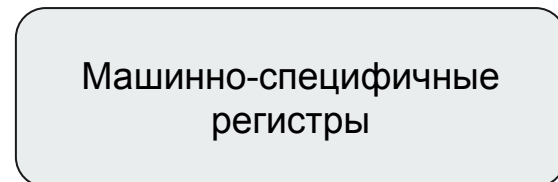
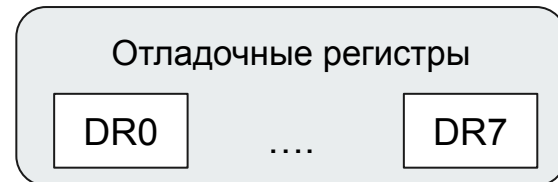
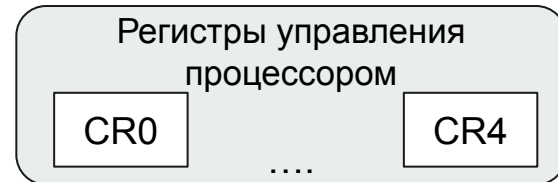
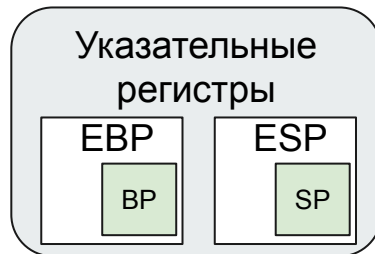
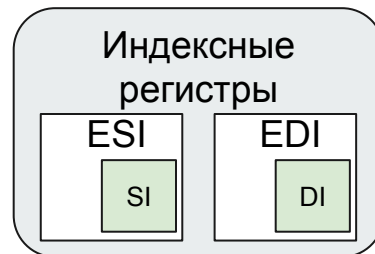
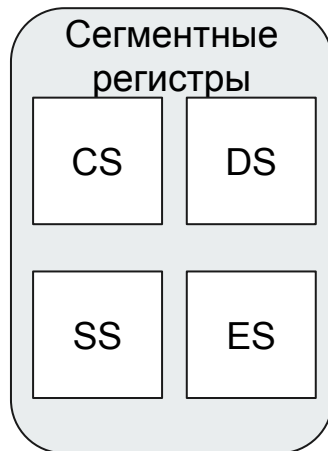
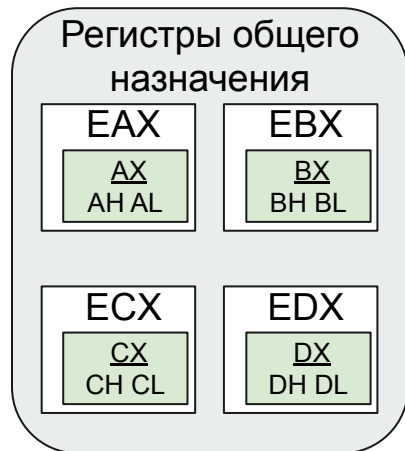
- обращение к оперативной памяти происходит по реальным (действительным) адресам, трансляция адресов не используется;
- набор доступных операций не ограничен;
- защита памяти не используется.

"Защищённый" режим

- обращение к памяти происходит по виртуальным адресам с использованием механизмов защиты памяти;
- набор доступных операций определяется уровнем привилегий (кольца защиты): системный и пользовательский уровни

Режим V86, ...

Регистры x86





Система команд

- Аналогична системе команд 16-разрядных процессоров
- Доступны как прежние команды обработки 8- и 16-разрядных аргументов, так и 32-разрядных регистров и переменных

- Пример:

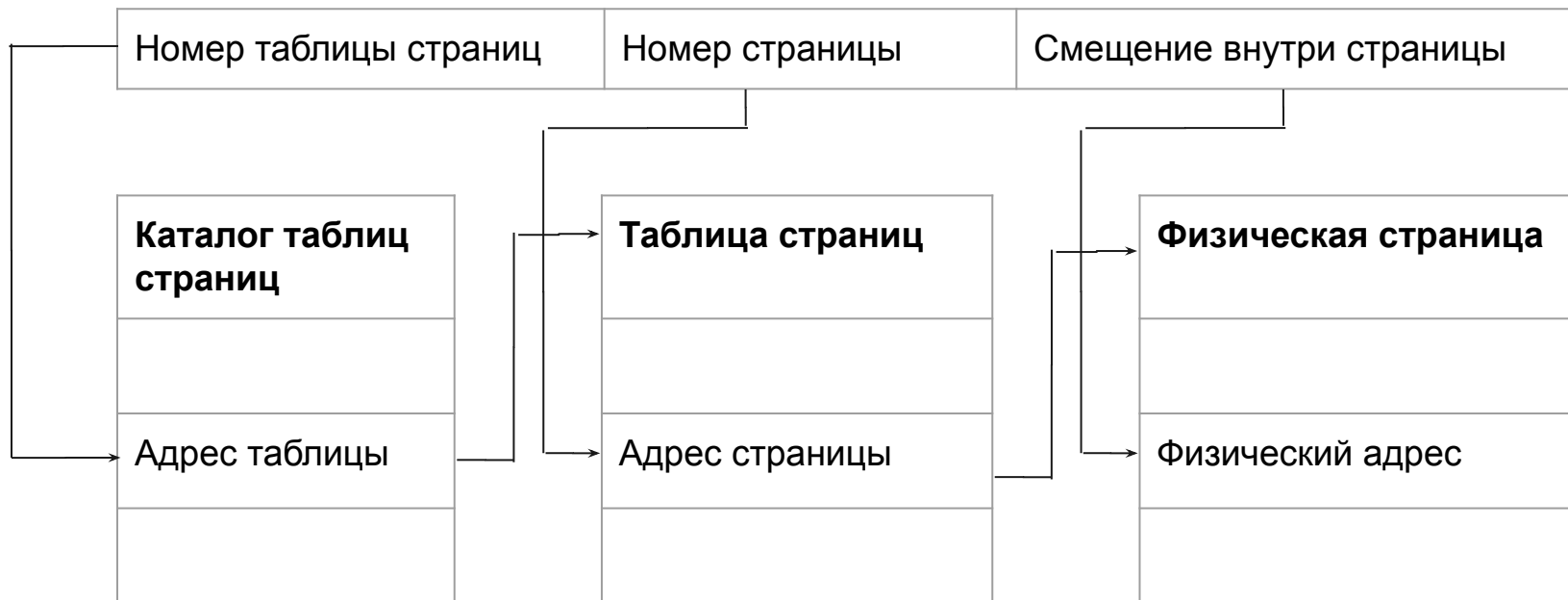
```
mov eax, 12345678h
xor ebx, ebx
mov bx, 1
add eax, ebx      ; eax=12345679h
```



Модели памяти

- Плоская - код и данные используют одно и то же пространство
- Сегментная - сложение сегмента и смещения
- Страничная - *виртуальные* адреса отображаются на физические постранично
 - виртуальная память — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (файл, или раздел подкачки)
 - основной режим для большинства современных ОС
 - в x86 минимальный размер страницы - 4096 байт
 - основывается на таблице страниц - структуре данных, используемой системой виртуальной памяти в операционной системе компьютера для хранения сопоставления между виртуальным адресом и физическим адресом. Виртуальные адреса используются выполняющимся процессом, в то время как физические адреса используются аппаратным обеспечением. Таблица страниц является ключевым компонентом преобразования виртуальных адресов, который необходим для доступа к данным в памяти.

Страничная организация памяти





Управление памятью в x86

- В сегментных регистрах - селекторы
 - 13-разрядный номер дескриптора
 - какую таблицу использовать - глобальную или локальную
 - уровень привилегий запроса 0-3
- По селектору определяется запись в одной из таблиц дескрипторов сегментов
- При включённом страничном режиме - по таблице страниц определяется физический адрес страницы либо выявляется, что она выгружена из памяти, срабатывает исключение и операционная система подгружает затребованную страницу из "подкачки" (swap)



Поддержка многозадачности

TSS (Task State Segment — сегмент состояния задачи) — специальная структура в архитектуре x86, содержащая информацию о задаче (процессе). Используется ОС для диспетчеризации задач, в т. ч. переключения на стек ядра при обработке прерываний и исключений



Исключения

- **Исключения** (Exceptions) подразделяются на отказы, ловушки и аварийные завершения.
- **Отказ** (fault) — это исключение, которое обнаруживается и обслуживается до выполнения инструкции, вызывающей ошибку. После обслуживания этого исключения управление возвращается снова на ту же инструкцию (включая все префиксы), которая вызвала отказ. Отказы, использующиеся в системе виртуальной памяти, позволяют, например, подкачать с диска в оперативную память затребованную страницу или сегмент.
- **Ловушка** (trap) — это исключение, которое обнаруживается и обслуживается после выполнения инструкции, его вызывающей. После обслуживания этого исключения управление возвращается на инструкцию, следующую за вызвавшей ловушку. К классу ловушек относятся и программные прерывания.
- **Аварийное завершение** (abort) — это исключение, которое не позволяет точно установить инструкцию, его вызвавшую. Оно используется для сообщения о серьезной ошибке, такой как аппаратная ошибка или повреждение системных таблиц.



Регистр EFLAGS

FLAGS + 5 специфических флагов



Регистры управления памятью

- GDTR: 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов (GDT) и 16-битный размер (лимит, уменьшенный на 1)
- IDTR: 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов обработчиков прерываний (IDT) и 16-битный размер (лимит, уменьшенный на 1)
- LDTR: 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий текущую таблицу локальных дескрипторов
- TR: 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий TSS текущей задачи



Регистры управления процессором

- CR0 - флаги управления системой
 - PG - включение режима страничной адресации
 - управление отдельными параметрами кеша
 - WP - запрет записи в страницы "только для чтения"
 - NE - ошибки FPU вызывают исключение, а не IRQ13
 - TS - устанавливается процессором после переключения задачи
 - PE - включение защищённого режима
- CR1 - зарезервирован
- CR2 - регистр адреса ошибки страницы - содержит линейный адрес страницы, при обращении к которой произошло исключение #PF
- CR3 - регистр основной таблицы страниц
 - 20 старших бит физического адреса начала каталога таблиц либо 27 старших бит физического адреса начала таблицы указателей на каталоги страниц, в зависимости от бита PAE в CR4
 - Управление кешированием и сквозной записью страниц
- CR4 - регистр управления новыми возможностями процессоров (с Pentium)



Отладочные регистры

- DR0..DR3 - 32-битные линейные адреса четырёх возможных точек останова по доступу к памяти
- DR4, DR5 - зарезервированы
- DR6 (DSR) - регистр состояния отладки. Содержит причину останова
- DR7 (DCR) - регистр управления отладкой. Управляет четырьмя точками останова



Машинно-специфичные регистры

- Управление кешем
- Дополнительное управление страничной адресацией
- Регистры расширений процессора: MMX и т.д.



Системные и привилегированные команды

- Выполнение ограничено, в основном, нулевым кольцом защиты
- LGDT, SGDT
- LLDT, SLDT
- LTR, STR
- LIDT, SIDT
- MOV CR0..CR4 или DR0..DR7, <источник>
- ...



Страничная адресация - преобразование линейного адреса в физический

- Линейный адрес:
 - биты 31-22 - номер таблицы страниц в каталоге
 - биты 21-12 - номер страницы в выбранной таблице
 - биты 11-0 - смещение от физического адреса начала страницы в памяти
- Каждое обращение к памяти требует двух дополнительных обращений!
- Необходим специальный кеш страниц - TLB
- Каталог таблиц/таблица страниц:
 - биты 31-12 - биты 31-12 физического адреса таблицы страниц либо самой страницы
 - атрибуты управления страницей



Механизм защиты

- Механизм защиты - ограничение доступа к сегментам или страницам в зависимости от уровня привилегий
- К типам сегментов реального режима (код, стек, данные) добавляется TSS - сегмент состояния задачи. В нём сохраняется вся информация о задаче на время приостановки выполнения. Размер - 68h байт.
- Структура:
 - селектор предыдущей задачи
 - Регистры стека 0, 1, 2 уровней привилегий
 - EIP, EFLAGS, EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, CS, DS, ES, FS, HS, SS, LDTR
 - флаги задачи
 - битовая карта ввода-вывода (контроль доступа программы к устройствам)



64-разрядные процессоры (x86-64)

AMD - с 2001, Intel - с 2003

- Режимы работы:
 - Legacy mode - совместимость с 32-разрядными процессорами
 - Long mode – 64-разрядный режим с частичной поддержкой 32-разрядных программ. Рудименты V86 и сегментной модели памяти упразднены
- Регистры:
 - целочисленные 64-битных регистры общего назначения - RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP;
 - новые целочисленные 64-битных регистры общего назначения R8 — R15
 - 64-битный указатель RIP и 64-битный регистр флагов RFLAGS.



Соглашения о вызовах

Соглашение о вызове — формализация правил вызова подпрограмм, которое должно включать:

- способ передачи параметров;
- способ возврата результата из функции;
- способ возврата управления.

Соглашения о вызовах определяются в рамках отдельных языков высокого уровня, а также - различных программных API, в т. ч. API операционных систем.



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 6

"Математический сопроцессор. Расширения процессоров x86"

ИУ7, 4-й семестр, 2020 г.

Сопроцессор (FPU – Floating Point Unit)

- Изначально - отдельное опциональное устройство на материнской плате, с 80486DX встроен в процессор
- Операции над 7-ю типами данных
 - целое слово (16 бит)
 - короткое целое (32 бита)
 - длинное слово (64 бита)
 - упакованное десятичное (80 бит)
 - короткое вещественное (32 бита)
 - длинное вещественное (64 бита)
 - расширенное вещественное (80 бит)

Форма представления числа с плавающей запятой в FPU

- Нормализованная форма представления числа ($1, \dots * 2^{\text{exp}}$)
- Экспонента увеличена на константу для хранения в положительном виде
- Пример представления 0,625 в коротком вещественном типе:
 - $1/4 + 1/8 = 0,101b$
 - $1,01b * 2^{-1}$
 - Бит 31 - знак мантииссы, 30-23 - экспонента, увеличенная на 127, 22-0 - мантиисса без первой цифры
 - **001111110**010000000000000000000000
- Все вычисления FPU - в расширенном 80-битном формате

Особые числа FPU

- Положительная бесконечность: знаковый - 0, мантисса - нули, экспонента - единицы
- Отрицательная бесконечность: знаковый - 1, мантисса - нули, экспонента - единицы
- NaN (Not a Number):
 - *qNaN (quiet)* - при приведении типов/отдельных сравнениях
 - *sNaN (signal)* - переполнение в большую/меньшую сторону, прочие ошибочные ситуации
- Денормализованные числа (экспонента = 0): находятся ближе к нулю, чем наименьшее представимое нормальное число

Регистры FPU

- R0..R7, адресуются не по именам, а рассматриваются в качестве стека ST. ST соответствует регистру - текущей вершине стека, ST(1)..ST(7) - прочие регистры
- SR - регистр состояний, содержит слово состояния FPU. Сигнализирует о различных ошибках, переполнениях
- CR - регистр управления. Контроль округления, точности
- TW – 8 пар битов, описывающих состояния регистров: число, ноль, не-число, пусто
- FIP, FDP - адрес последней выполненной команды и её операнда для обработки исключений

Исключения FPU

- Неточный результат - произошло округление по правилам, заданным в CR. Бит в SR хранит направление округления
- Антипереполнение - переход в денормализованное число
- Переполнение - переход в "бесконечность" соответствующего знака
- Деление на ноль - переход в "бесконечность" соответствующего знака
- Денормализованный операнд
- Недействительная операция

Команды пересылки данных FPU

- FLD - загрузить вещественное число из источника (переменная или ST(n)) в стек. Номер вершины в SR увеличивается
- FST/FSTP - скопировать/считать число с вершины стека в приёмник
- FILD - преобразовать целое число из источника в вещественное и загрузить в стек
- FIST/FISTP - преобразовать вершину в целое и скопировать/считать в приёмник
- FBLD, FBSTP - загрузить/считать десятичное BCD-число
- FXCH - обменять местами два регистра (вершину и источник) стека

Базовая арифметика FPU

- FADD, FADDP, FIADD - сложение, сложение с выталкиванием из стека, сложение целых. Один из операндов - вершина стека
- FSUB, FSUBP, FISUB - вычитание
- FSUBR, FSUBRP, FISUBR - обратное вычитание (приёмника из источника)
- FMUL, FMULP, FIMUL - умножение
- FDIV, FDIVP, FIDIV - деление
- FDIVR, FDIVRP, FIDIVR - обратное деление (источника на приёмник)
- FPREM - найти частичный остаток от деления (делится ST(0) на ST(1)). Остаток ищется цепочкой вычитаний, до 64 раз

Базовая арифметика FPU (продолжение)

- FABS - взять модуль числа
- FCHS - изменить знак
- FRNDINT - округлить до целого
- FSCALE - масштабировать по степеням двойки ($ST(0)$ умножается на $2^{ST(1)}$)
- FXTRACT - извлечь мантиссу и экспоненту. $ST(0)$ разделяется на мантиссу и экспоненту, мантисса дописывается на вершину стека
- FSQRT - вычисляет квадратный корень $ST(0)$

Команды сравнения FPU

- FCOM, FCOMP, FCOMPP - сравнить и вытолкнуть из стека
- FUCOM, FUCOMP, FUCOMPP - сравнить без учёта порядков и вытолкнуть
- FICOM, FICOMP, FICOMP - сравнить целые
- FCOMI, FCOMIP, FUCOMI, FUCOMIP (P6)
- FTST - сравнивает с нулём
- FXAM - выставляет флаги в соответствии с типом числа

Трансцендентные операции FPU

- FSIN
- FCOS
- FSINCOS
- FPTAN
- FPATAN
- F2XM1 – $2^x - 1$
- FYL2X, FYL2XP1 – $y * \log_2 x$, $y * \log_2(x+1)$

Константы FPU

- FLD1 – 1,0
- FLDZ – +0,0
- FLDPi – число Π
- FLDL2E – $\log_2 e$
- FLDL2T – $\log_2 10$
- FLDLN2 – $\ln(2)$
- FLDLG2 – $\lg(2)$

Команды управления FPU

- FINCSTP, FDECSTP - увеличить/уменьшить указатель вершины стека
- FFREE - освободить регистр
- FINIT, FNINIT - инициализировать сопроцессор / инициализировать без ожидания (очистка данных, инициализация CR и SR по умолчанию)
- FCLEX, FNCLEX - обнулить флаги исключений / обнулить без ожидания
- FSTCW, FNSTCW - сохранить CR в переменную / сохранить без ожидания
- FLDCW - загрузить CR
- FSTENV, FNSTENV – сохранить вспомогательные регистры (14/28 байт) / сохранить без ожидания
- FLDENV - загрузить вспомогательные регистры
- FSAVE, FNSAVE, FXSAVE - сохранить состояние (94/108 байт) и инициализировать, аналогично FINIT
- FRSTOR, FXRSTOR - восстановить состояние FPU
- FSTSW, FNSTSW - сохранение CR
- WAIT, FWAIT - обработка исключений
- FNOP - отсутствие операции

Команда CPUID (с 80486)

Идентификация процессора

- Если EAX = 0, то в EAX - максимальное допустимое значение (1 или 2), а EBX:ECX:EDX – 12-байтный идентификатор производителя (ASCII-строка).
- Если EAX = 1, то в EAX - версия, в EDX - информация о расширениях
 - EAX - модификация, модель, семейство
 - EDX: наличие FPU, поддержка V86, поддержка точек останова, CR4, PAE, APIC, быстрые системные вызовы, PGE, машинно-специфичный регистр, CMOVss, **MMX**, **FXSR (MMX2)**, **SSE**
- Если EAX = 2, то в EAX, EBX, ECX, EDX возвращается информация о кэшах и TLB

MMX (1997, Pentium MMX)

Увеличение эффективности обработки больших потоков данных (изображения, звук, видео...) - выполнение простых операций над массивами однотипных чисел.

- 8 64-битных регистров MM0..MM7 - **мантиссы регистров FPU**. При записи в MMn экспонента и знаковый бит заполняются единицами
- Пользоваться одновременно и FPU, и MMX не получится, требуется FSAVE+FRSTOR
- Типы данных MMX:
 - учетверённое слово (64 бита);
 - упакованные двойные слова (2);
 - упакованные слова (4);
 - упакованные байты (8).
- Команды MMX перемещают упакованные данные в память или обычные регистры целиком, но арифметические и логические операции выполняют поэлементно.
- *Насыщение* - замена переполнения/антипереполнения превращением в максимальное/минимальное значение

Команды пересылки данных MMX

- MOVD, MOVQ - пересылка двойных/учетверённых слов
- PACKSSWB, PACKSSDW - упаковка со знаковым насыщением слов в байты/двойных слов в слова. *Приёмник -> младшая половина приёмника, источник -> старшая половина приёмника*
- PACKUSWB - упаковка слов в байты с беззнаковым насыщением
- PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ - распаковка и объединение старших элементов источника и приёмника через 1

Арифметические операции MMX

- PADDB, PADDW, PADDD - поэлементное сложение, перенос игнорируется
- PADDSB, PADDSW - сложение с насыщением
- PADDSB, PADDSW - сложение с насыщением
- PADDUSB, PADDUSW - беззнаковое сложение с насыщением
- PSUBB, PSUBW, PDUBD - вычитание, заём игнорируется
- PSUBSB, PSUBSW - вычитание с насыщением
- PSUBUSB, PSUBUSW - беззнаковое вычитание с насыщением
- PMILHW, PMULLW - старшее/младшее умножение (сохраняет старшую или младшую части результата в приёмник)
- PMADDWD - умножение и сложение. Перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших

Команды сравнения MMX

- PCMPREQB, PCMPREQW, PCMPREQD - проверка на равенство. Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе - нулями
- PCMPGTB, PCMPGTW, PCMPGTD - сравнение. Если элемент приёмника больше, то заполняется единицами, иначе - нулями

Логические операции MMX

- PAND - логическое И
- PANDN - логическое НЕ-И (штрих Шеффера) (источник*НЕ(приёмник))
- POR - логическое ИЛИ
- PXOR - исключающее ИЛИ

Сдвиговые операции MMX

- PSLLW, PSLLD, PSLLQ - логический влево
- PSRLW, PSRLD, PSRLQ - логический вправо
- PSRAW, PSRAD - арифметический вправо

Расширение SSE (Pentium III, 1999)

Решение проблемы параллельной работы с FPU

- 8 128-разрядных регистров
- Свой регистр флагов
- Основной тип - вещественные одинарной точности (32 бита)
- Целочисленные команды работают с регистрами MMX
- Команды:
 - Пересылки
 - Арифметические
 - Сравнения
 - Преобразования типов
 - Логические
 - Целочисленные
 - Упаковки
 - Управления состоянием
 - Управления кэшированием
- Развитие: SSE2, SSE3...

Расширение AES (Intel Advanced Encryption Standard New Instructions; AES-NI, 2008)

Цель - ускорение шифрования по алгоритму AES

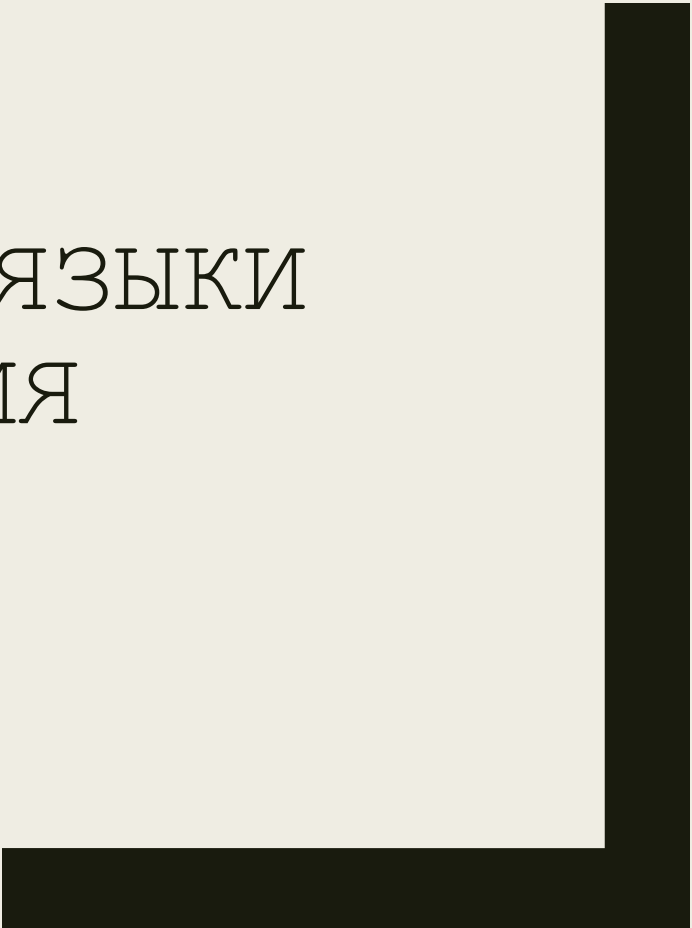
■ Команды:

- раунда шифрования;
- раунда расшифровывания;
- способствования генерации ключа



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 7
"Макроопределения"
ИУ7, 4-й семестр, 2020 г.



Макроопределения

Макроопределение (макрос) - именованный участок программы, который ассемблируется каждый раз, когда его имя встречается в тексте программы.

- Определение:

имя MACRO параметры

.....

ENDM

- Пример:

load_reg MACRO register1, register2

push register1

pop register2

ENDM

Директива присваивания =

Директива присваивания служит для создания целочисленной макропеременной или изменения её значения и имеет формат:

Макроимя = Макровыражение

- Макровыражение (или Макровыражение, или Константное выражение) - выражение, вычисляемое препроцессором, которое может включать целочисленные константы, макроимена, вызовы макрофункций, знаки операций и круглые скобки, результатом вычисления которого является целое число
- Операции: арифметические (+, -, *, /. MOD), логические, сдвигов, отношения

Директивы отождествления EQU, TEXTEQU

Директива для представления текста и чисел:

Макроимя EQU нечисловой текст и не макроимя ЛИБО число

Макроимя EQU <Операнд>

Макроимя TEXTEQU Операнд

■ Пример:

X EQU [EBP+8]

MOV ESI,X

Макрооперации

- `%` – вычисление выражение перед представлением числа в символьной форме
- `<>` – подстановка текста без изменений
- `&` – склейка текста
- `!` – считать следующий символ текстом, а не знаком операции
- `;;` – исключение строки из макроса

Блоки повторения

- REPT число ... ENDM - повтор фиксированное число раз

- IRP или FOR:

IRP form,<fact_1[,fact_2,...]> ... ENDM

Подстановка фактических параметров по списку на место формального

- IRPC или FORC:

IRPC form,fact ... ENDM

Подстановка символов строки на место формального параметра

- WHILE:

WHILE cond ... ENDM

Директивы условного ассемблирования

- IF:
IF c1
...
ELSEIF c2
...
ELSE
...
ENDIF
- IFB <par> - истинно, если параметр не определён
- IFNB <par> - истинно, если параметр определён
- IFIDN <s1>,<s2> - истинно, если строки совпадают
- IFDIF <s1>,<s2> - истинно, если строки разные
- IFDEF/IFNDEF <name> - истинно, если имя объявлено/не объявлено

Директивы управления листингом

- Листинг - файл, формируемый компилятором и содержащий текст ассемблерной программы, список определённых меток, перекрёстных ссылок и сегментов.
- `TITLE`, `SUBTTL` - заголовок, подзаголовок на каждой странице
- `PAGE` высота, ширина
- `NAME` - имя программы
- `.LALL` - включение полных макрорасширений, кроме `;;`
- `.XALL` - по умолчанию
- `.SALL` - не выводить тексты макрорасширений
- `.NOLIST` - прекратить вывод листинга

Комментарии

comment @

... многострочный текст ...

@



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 8
"Не-x86 ассемблеры"
ИУ7, 4-й семестр, 2020 г.

RISC-архитектура

Ранние архитектуры процессоров (комплексные, CISC (Complex instruction set computer)):

- большее количество команд
- разные способы адресации для упрощения написания программ на ассемблере
- поддержка конструкций языков высокого уровня

Недостатки: на практике многие возможности CISC используются компиляторами ЯВУ ограниченно, а их поддержка затратна.

RISC (*reduced instruction set computer*):

- сведение набора команд к простым типовым
- большее количество регистров (возможно за счёт общего упрощения архитектуры)
- стандартизация формата команд, упрощение конвейеризации

Семейство процессоров ARM

Свыше 90% рынка процессоров для мобильных устройств

ARMv1 – 1983 г.

Современные - ARMv7, ARMv8.

Регистры общего назначения ARMv7:

- R0-R12
- R13 – SP
- R14 – LR (регистр связи)
- R15 – PC (счётчик команд)

Регистры R8–R12 существуют в двух экземплярах:

- для режима обработки быстрого прерывания
- для остальных режимов

Регистры LR и SP для каждого режима свои (6-7 пар)

Режимы ARM

- User mode — обычный режим выполнения программ. В этом режиме выполняется большинство программ.
- Fast Interrupt (FIQ) — режим быстрого прерывания (меньшее время срабатывания).
- Interrupt (IRQ) — основной режим прерывания.
- System mode — защищённый режим для использования операционной системой.
- Abort mode — режим, в который процессор переходит при возникновении ошибки доступа к памяти (доступ к данным или к инструкции на этапе prefetch конвейера).
- Supervisor mode — привилегированный пользовательский режим.
- Undefined mode — режим, в который процессор входит при попытке выполнить неизвестную ему инструкцию

Наборы команд ARM

- Базовый
- Thumb (16-разрядные, более производительные)
- Thumb2 (32-разрядные)

Команды ветвления B, BL, BLX

- B (Branch) - переход
- BL (Branch with link) - переход с сохранением адреса возврата в LR
- BLX - переход с переключением системы команд

Архитектура VLIW. Эльбрус-8С

VLIW (*very large instruction word*) - продолжение идей RISC для многопроцессорных систем

- В каждой инструкции явно указывается, что должно делать каждое ядро процессора

Эльбрус-8С:

- 8 ядер
- В каждом ядре - 6 арифметико-логических каналов со своими АЛУ и FPU, до 24 операций за такт
- Спецификация опубликована 30.05.2020

• Широкая команда Эльбруса

Широкая команда - набор элементарных операций, которые могут быть запущены на исполнение в одном такте.

Доступны:

- 6 АЛУ (возможности различны)
- Устройство передачи управления
- 3 устройства для операций над предикатами
- 6 квалифицирующих предикатов
- 4 устройства асинхронного для команд чтения данных
- 4 32-разрядных литерала для констант

Определяющие свойства архитектуры "Эльбрус"

- Регистровый файл (рабочие регистры) - 256 регистров (32 для глобальных данных и 224 для стека процедур)
 - механизм регистровых окон: вызывающая подпрограмма выделяет вызываемой область в своём регистровом окне; на начало указывает регистр WD
 - пространство регистров подвижной базы - пространство в текущем окне, на начало указывает регистр BR
- Предикатный файл - 32 регистра со значениями true/false
- Подготовка передачи управления (disp) - подготовка к переходам при ветвлении для исключения задержек
- Асинхронный доступ к массивам

Java. Java virtual machine (JVM)

Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины.

Байт-код Java — набор инструкций, исполняемых виртуальной машиной Java. Каждый код операции байт-кода — один байт.

Группы инструкций:

- загрузка и сохранение (например, ALOAD_0, ISTORE),
- арифметические и логические операции (например, IADD, FCMPL),
- преобразование типов (например, I2B, D2I),
- создание и преобразование объекта (например, NEW, PUTFIELD),
- управление стеком (например, DUP, POP),
- операторы перехода (например, GOTO, IFEQ),
- вызовы методов и возврат (например, INVOKESTATIC, IRETURN).

Платформа .NET. CLR, CIL

.NET (2002) - платформа, основанная на CLR (Common Language Runtime, общезыковая исполняющая среда).

CLR — исполняющая среда для байт-кода CIL (MSIL), в которой компилируются программы, написанные на .NET-совместимых языках программирования.

CIL (Common Intermediate Language) — «высокоуровневый ассемблер» виртуальной машины .NET., основанный на работе со стеком.

```
ldloc.0      // push local variable 0 onto stack
ldloc.1      // push local variable 1 onto stack
add          // pop and add the top two stack items then push the
result onto the stack
stloc.0      // pop and store the top stack item to local variable
0
```


WebAssembly (wasm)

WebAssembly — это бинарный формат инструкций для стековой виртуальной машины, предназначенной для компиляции программ на ЯВУ для WEB.

Исходный код на C	«линейный ассемблерный байт-код»	бинарный код WASM
<pre>int factorial(int n) { if (n == 0) return 1; else return n * factorial(n-1); }</pre>	<pre>get_local 0 i64.eqz if i64 i64.const 1 else get_local 0 get_local 0 i64.const 1 i64.sub call 0 i64.mul end</pre>	<pre>20 00 50 04 7e 42 01 05 20 00 20 00 42 01 7d 10 00 7e 0b</pre>