Содержание

1 Часть 1. **2**

1.1 Область применения языков программирования низкого уровня. Поколения ПК IBM PC. 2

1.2 Базовая архитектура ПК IBM PC. 3

1.3 Организация памяти в реальном режиме работы. Сегментые регистры. Понятие исполняемого и физического адреса. . . . 4

1.4 Процессор с точки зрения программиста. Регистры общего назначения. Регистры флагов. 4

1.5 Понятие команды и директивы в Ассемблере. Формат команды и директивы. 6

1.6 Структура программы на Ассемблере с исполением стандартных директив сегментации. 7

1.7 Основные элементы языка Ассемблера: имена, константы, переменные, выражения. 8

1.8 Команды пересылки, особенности их использования. Команды пересылки безусловной и условной, команды загрузки адреса. 10

1.9 Сегмент стека, команды для работы со стеком, команды прерывания. 11

1.10 Модели памяти, организация программы с помощью точечных директив. 12

2 Часть 2. **13**

2.1 Команды безусловной передачи управления. Обращение к подпрограммам и возврат из них. 13

1 Часть 1.

1.1 Область применения языков программирования низкого уровня. Поколения ПК IBM PC.

1. Область применения языков программирования низкого уровня.

Низкоуровневые языки программирования используются везде, где необходима максимальная производительность:

1. Там, где требуется максимальная скорость выполнения: ядра ОС и программы, включаемые в них, основные компоненты компьютерных игр;
2. То, что непосредственно взаимодействует с внешними устройствами: драйвера для внешних устройств;
3. Всё, что должно максимально использовать возможности процессора: ядра многозадачных ОС, программы перевода в защищённый режим;
4. Всё что использует возможности ОС: вирусы, антивирусы, программы защиты и взлома защит;
5. Программы, предназначенные для обработки больших объёмов информации и требующие максимальной эффективности, например, программы, управляющие БД.

2. Поколения ПК IBM PC:

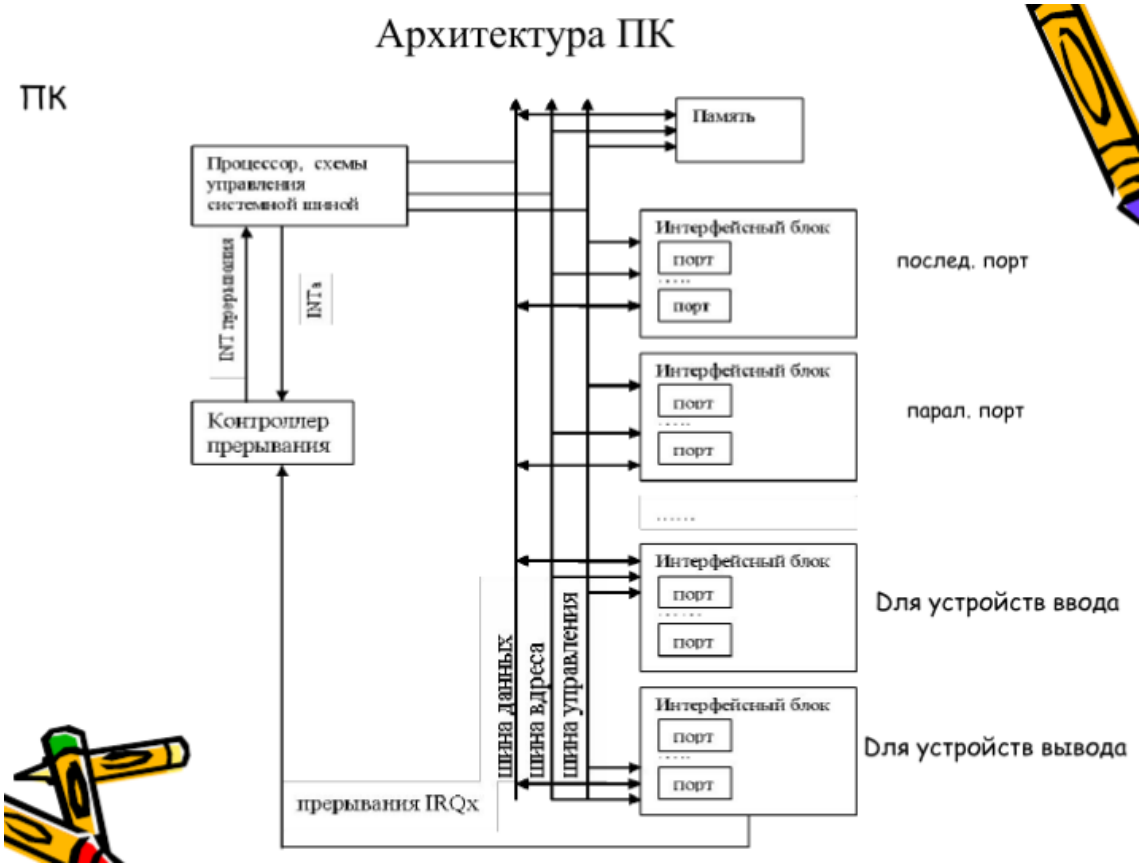
1. 1981 г. — IBM PC;
2. 1984 г. — IBM PC AT (Advenced Technology);
3. 1987 г. — 32-разрядный i386;
4. 1990 г. — i486: 1,5 млн транзисторов, 1Мкм технология, 5-ти стадийный конвейер для выполнения команд, кэш-память на кристалле процессора 8Кбайт;
5. 1993 г. — 64-разрядный МП "Pentium": 3,1 млн транзисторов, 0,8 Мкм технология;
6. Pentium Pro, Pentium 2, Pentium 3 с тактовой частотой 300-600 МГц;
7. "Willmate"800-1200 МГц, кэш до 1 Мбайта 2000 г. С 2002 г. — 0,13 Мкм, 146 мм^2 55 млн транзисторов.

1.2 Базовая архитектура ПК IBM PC.

В современных ПК реализован магистрально-модульный принцип построения. Все устройства (модули) подключены к центральной магистрали — системной шине, которая включает в себя адресную шину, шину данных и шину управления.

Шина — это набор линий связи, по которым передаётся информация от одного из источников к одному или нескольким приёмникам. Адресная шина однонаправленная, адреса передаются от процессора. Шина данных двунаправленная, данные передаются как от процессора, так и к процессору. В шину управления входят линии связи и однонаправленные и двунаправленные.

Внешние устройства работают значительно медленее процессора, поэтому для организации параллельной работы процессора и внешних устройств в архитектуру компьютера входит система прямого доступа к памяти (ДМА) и интерфейсные блоки, включающие в себя устройсва управления внешними устройствами (контроллеры, адаптеры) ...



1.3 Организация памяти в реальном режиме работы. Сегментые регистры. Понятие исполняемого и физического адреса.

1. Организация памяти в реальном режиме работы:

Процессор ix86 после включения питания устанавливается в реальный режим адрессации памяти и работы процессора.

В этом режиме процессор может работать с ОП как с непрерывным массивом байтов (модель памяти flat), так и с разделённой на много массиво — сегментов (в этом случае адрес байта состоит из 2 частей: адрес начала сегмента и смещение внутри сегмента). В реальном режиме размер сегмента фиксирован и составляет 64 Кбайта. Адрес сегмента кратен 16 и в 16-ричной СС может быть записан в виде $XXXX0_{16}$ и четыре старшие цифры адреса сегмента содержатся в сегментном регистре.

2. Сегментные регистры:
DS, ES, FS, GS, CS, SS

DS, ES, FS, GS — 16-разрядные сегментные регистры, используемые для определения начала сегментов данных.

CS — сегментный регистр кодового сегмента.

SS — сегментый регистр стека. Он устанавливается автоматически ОС и в нём хранится адрес начала сегмента стека, а указатель на вершину стека хранится в SP. Стек растёт от максимального адреса к минимальному при добавлении элементов в него. В модели памяти flat стек размещается в старших адресах, а программа в младших.

3. Понятие исполняемого и физического адреса:

Физический адрес представляет собой двадцатибитное беззнаковое целое, которое идентифицирует расположение байта в пространстве памяти.

ФА = АС + ИА

АС — адрес начала сегмента (значение сегментного регистра).

ИА — исполняемый адрес (смещение, в байтах, относительно начала сегмента).

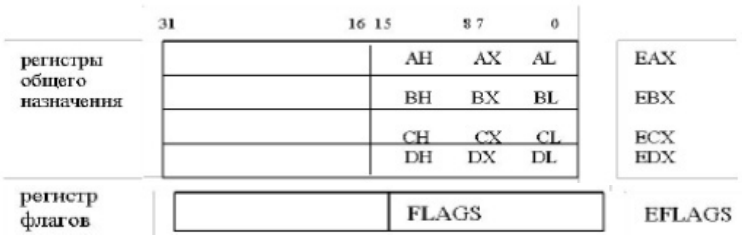
1.4 Процессор с точки зрения программиста. Регистры общего назначения. Регистры флагов.

1. Процессор с точки зрения программиста:

Процессор с точки зрения программиста — совокупность программнодоступных средств процессора.

2. Регистры общего назначения:

Регистр – это набор из n устройств, способных хранить n-разрядное двоичное число.



AX — аккумулятор.

DX — регистр данных.

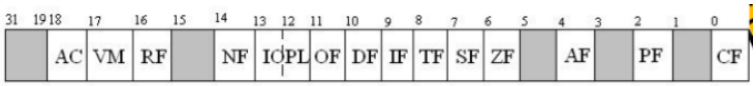
BX — регистр базы используется для организации специальной адресации операндов по базе.

CX — счётчик используется автоматически для организации циклов и при работе со строками.

Вышеперечисленные регистры могут использоваться для временного хранения адресов и данных.

3. Регистры флагов:

Регистр флагов FLAGS или EFLAGS определяет состояние процессора и программы в каждый текущий момент времени.



Биты 1, 3, 5, 15, 19 - 31 - не используются, зарезервированы.

В реальном режиме используют 9 флагов, из них 6 реагируют на результат выполнения команды, 3 определяют режим работы процессора.

В защищенном режиме используются 5 дополнительных флагов, определяющих режим работы процессора.

CF устанавливается в 1, если при выполнении команды сложения осуществляется перенос за разрядную сетку, а при вычитании требуется заем. 0FFFFh + 1 = 0000h и CF = 1 при работе со словами

PF = 1, если в младшем байте результата содержится четное количество единиц.

AF = 1, если в результате выполнения команды сложения (вычитания) осуществлялся перенос (заем) из 3-го разряда байта в 4-й (из 4-го в 3-й).

ZF = 1, если результатом выполнения операции является 0 во всех разрядах результата.

SF всегда равен знаковому разряду результата.

TF = 1 прерывает работу процессора после каждой выполненной команды.

DF определяет направление обработки строк данных, если DF= 0 – обработка строк идет в сторону увеличения адресов, 1 - в сторону уменьшения, (автоматическое увеличение или уменьшение содержимого регистров индексов SI и DI).

OF = 1, если результат команды превышает максимально допустимый для данной разрядной сетки.

IOPL = 1, если уровень привилегии текущей программы меньше значения этого флажка, то выполнение команды ввод/вывод для этой программы запрещен.

NT - определяет режим работы вложенных задач.

RF позволяет маскировать некоторые прерывания процессора.

VM - позволяет перейти из защищенного режима в режим виртуальных машин.

AC =1 приведет к сообщению об ошибке, если адреса операндов длиной в слово или двойное слово не будут кратны двум и четырем соответственно.

1.5 Понятие команды и директивы в Ассемблере. Формат команды и директивы.

1. Понятие команды и её формат:

```
[<метка>[:]] <код операции> [<операнды>, ...] [; комментарии]
```

Команда — это цифровой двоичный код, состоящий из двух подпоследовательностей двоичных цифр, одна из которых определяет код операции (сложить, умножить, переслать), вторая — определяет операнды, участвующие в операции и место хранения результата.

Команда может быть одно-, двух- и трёхадресной. Команда в памяти может занимать от 1 до 15 байт и длина команды зависит от кода операции, количества и места расположения операндов.

Операнды могут располагаться в регистрах, в памяти и непосредственно в команде и размер операндов может быть — байт, слово или двойное слово.

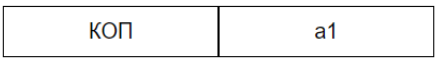
Трёхадресная команда: a1, a2 числа, участвующие в операции, al — адрес, по которому будет помещён результат.

КОП	al	a2	a3

Двухадресная команда: a1 — первый операнд, куда в дальнейшем будет помещён результат команды, a2 — второй операнд.

КОП	a1	a2

Одноадресная команда: a1 — либо число, участвующие в операции, либо адрес, по которому будет помещён результат.



Понятие директивы и формат директивы:

Директива — псевдооперация — информирует Ассемблер о чём-либо, например, какой объём памяти выделить под переменную или как следует объединять инструкции для формирования программного модуля, но сама директива в собранной программе никак не проявляется.

```
[<симв. имя>] <код псевдооперации> <операнды> [; комментарии]
```

Операндов может быть любое количество.

1.6 Структура программы на Ассемблере с исполением стандартных директив сегментации.

Исходный модуль на Ассемблере — последовательность строк, команд, директив и комментариев.

Исходный модуль просматривается Ассемблером, пока не встретится директива end. Обычно программа на Ассемблере состоит из 3 семгментов: стека, данных и кода.

Каждый сегмент начинается диретктивой начала сегмента — Segment и заканчивается диретктивой конца сегмента — ends, в операдах директивы Segment содержится информация о назначении сегмента.

Кодовый сегмент представляет собой программу решения поставленной задачи.

В сегменте стека выделяется место под стек.

В сегменте данных описываются данные, используемые в программе, выделяется место под промежуточные и окончательные результаты.

```
;сегмент стека
Sseg Segment stack 'stack'
    db 256 dup (ü)
    ;...
Sseg ends

;сегмент данных
Dseg Segment 'data'
    ;...
Dseg ends

;сегмент кода
Cseg Segment 'code'
    assume CS:Cseg, DS:Dseg, SS:Sseg    ;связь
        сегментных
```

```
                                        ;регистров
                                            и
                                        сегментов

                                        ;это
                                        позволяет

                                        Асссемблеру

                                        ;правильно

                                        добавить

                                        ;префиксы
                                        при
                                        обращении

                                        ;к
                                        регистрам.

                                        Например
                                        ,
                                        ;SP
                                        эквивалентен

                                        ;SS:SP, а
                                        SS =
                                        Sseg

    Start proc far                      ;главная
        процедура
        ;...
        ret
    Start endp
Cseg ends
end start
```

1.7 Основные элементы языка Ассемблера: имена, константы, переменные, выражения.

1. Имена:

Символические имена на ассемблере могут состоять из строчных и прописных букв латинского алфавита, цифр от 0 до 9 и некоторых символов '_', '.', '?', ...

2. Константы:

В программе на Ассемблере могут использоваться константы пяти типов: целые двоичные, десятичные, шестнадцатеричные, действительные с плавающей точкой, символьные.

Целые двоичные – это последовательности 0 и 1 со следующим за ними символом **‘b’, например, 10101010b или 11000011b.**

Целые десятичные - это обычные десятичные числа, возможно заканчивающиеся буквой d, например, **– 125 или 78d.**

Целые шестнадцатеричные числа – должны начинаться с цифры и заканчиваются всегда ‘h’, если первый символ – **‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’,** то перед ним необходимо поставить 0, иначе они будут восприниматься как символические имена.

Числа действительные с плавающей точкой представляются в виде мантиссы и порядка, например, **– 34.751e+02 – это 3475.1 или 0.547e-2 – это 0.00547.**

Строковые данные – это последовательности символов, заключенные в апострофы или двойные кавычки, например,

'abcd', 'a1b2c3', ‘567'.

Также, как и в языках высокого уровня, в Ассемблере могут использоваться именованные константы. Для этого существует специальная директива **EQU**. Например,

M EQU 27 ; директива EQU присваивает имени M значение 27.

Переменные в Ассемблере определяются с помощью директив определения данных и памяти, например,

v1 DB ?

v2 DW 34

или с помощью директивы ‘ = ’

v3 = 100

v3 = v3+1

Константы в основном используются в директивах определения или как непосредственные операнды в командах.

Выражения в Ассемблере строятся из операндов, операторов и скобок.

Операнды – это константы или переменные.

Операторы – это знаки операций (арифметических, логических, отношений и некоторых специальных)

Арифметические операции: **‘+’, ‘-‘, ‘*’, ‘/’, mod**.
Логические операции: **NOT, AND, OR, XOR**.
Операции отношений: **LT(<), LE(≤), EQ(=), NE(≠), GT(>), GE(≥).**
Операции сдвига: сдвиг влево **(SHL),** сдвиг вправо **(SHR)**
Специальные операции: **offset и PTR**
offset <имя> - ее значением является смещение операнда, а операндом может быть метка ли переменная;
PTR – определяет тип операнда:
BYTE = 1 байт,
WORD = 2 байт,
DWORD = 4 байт,
FWORD = 6 байт,
QWORD = 8 байт,
TWORD = 10 байт;

или тип вызова: **NEAR** – ближний, **FAR** – дальний.
Примеры выражений: **1) 10010101b + 37d 2) OP1 LT OP2**
3) (OP3 GE OP4) AND (OP5 LT OP6) 4) 27 SHL 3 ;

1.8 Команды пересылки, особенности их использования. Команды пересылки безусловной и условной, команды загрузки адреса.

1. Команды перессылки, особенности их использования:

Команды перессылки позволяют копировать данные из регистра в регистр, из памяти в регистр и из регистра в память.

Особенности использования команд перессылки:

1. Нельзя пересылать информацию из одной области памяти в другую;
2. Нельзя пересылать информацию из одного сегментного регистра в другой;
3. Нельзя пересылать непосредственный операнд в сегментный регистр, но если такая необходимость возникает, то нужно использовать в качестве промежуточного один из регистров общего назначения.

   MOV DX, 100h

   MOV DS, DX;
4. Нельзя изменять командой MOV содержимое регистра CS;
5. Данные в памяти хранятся в перевёрнутом виде, а в регистрах в нормальном виде, и команда перессылки это учитывает.

   R DW 1234h В байте с адресом R будет 34h, а в R + 1 — 12h

   MOV AX, R; 12h → AH, 34h → AL;

6. Размер передаваемых данных определяется типом операндов в команде.

   X DB ? ;один байт

   Y DW ?

   MOV Y,0 ;0 воспринимается как слово

   MOV byte PTR [SI], 0

   <тип> PTR <выражение>, где выражение — константа или адрес, а тип это размер данных;

7. Если тип обоих операндов определяется, то эти типы должны соответствовать друг другу.

   MOv AH, 500 ;ошибка

   MOV AH, 240 ;всё верно.

2. Команды пересылки безусловной и условной, команды загрузки адреса:

К командам пересылки относят команду обмена значений операндов.

XCHG OP1, OP2 ; r↔r ∨ r↔m
MOV AX, 10h ;
MOV BX, 20h ;
XCHG AX, BX ; (AX) = 20h, (BX) = 10h

Для перестановки значений байтов внутри регистра используют **BSWOP**.

(EAX) = 12345678h
BSWOP EAX ; (EAX) = 78563412h

Команды конвертирования:

CBW ; безадресная команда, (AL) → AX.
CWD ;(AX) → DX:AX
CW**E** ; (AX) → EAX (для i386 и выше)
CDF ; (EAX) → EDX:EAX (для i386 и выше)

Команды условной пересылки CMOVxx

CMOVL AL, BL ; если (AL) < (BL), то (BL) → (AL)

Загрузка адреса.

LEA OP1, OP2 ; вычисляет адрес OP2 и пересылает первому операнду, который может быть только регистром.

LEA BX, M[DX][DI]

1.9 Сегмент стека, команды для работы со стеком, команды прерывания.

1. Сегмент стека, команды для работы со стеком:

Стек определяется с помощью регистров SS и SP(ESP). Сегментный регистр SS содержит адрес начала сегмента стека. ОС сама выбирает этот адрес и пересылает его в регистр SS. Регистр SP указывает на вершину стека и при добавлении элемента стека содержимое этого регистра уменьшается на длину операнда.

Добавить элемент в стек можно с помощью команды:
PUSH <операнд>,
где операндом может быть как регистр так и переменная.
Удалить элемент с вершины стека можно с помощью операции:
POP <операнд>.

Для i186 и > PUSHA/POPA позволяют положить/удалить в стек содержимое всех регистров общего назначения в последовательности AX, BX, CX, DX, SP, BP, SI, DI.

Для i386 и > PUSHAD/POPAD позволяют сделать тоже самое, но уже с 32-разрядными регистрами

К любому элементу стека можно обратиться следующим образом:
MOV BP,SP ;(SP) → BP
MOV AX,[BP+6] ;(SS:(BP+6)) → AX.

2. Команды прерывания:

`int` — **команда прерывания**. Ее выполнение приводит к передаче управления DOS или BIOS, а после завершения какой-то системной обрабатывающей программы управление передается следующей команде.

Действия, происходящие после выполнения `int`, будут зависеть от значения операнда и от значений, хранящихся в регистрах.

Например, прерывания `int 21h` вызовут одну из DOS-функций, прерывания `int 10h` — функции видео драйвера, `int 16h` — функции драйвера клавиатуры.

1.10 Модели памяти, организация программы с помощью точечных директив.

В программе на Ассемблере могут использоваться упрощенные (точечные) директивы.

`.model <модель>` — определяет модель памяти, выделяемой для программы.
`<модель>` — одна из:

- `tiny` — под всю программу один сегмент памяти;
- `small` — по одному сегменту на данные и программу;

- `medium` — под данные один сегмент, под программу несколько;
- `compact` — под программу один сегмент, под данные несколько;
- `large` — под данные и под программу выделяется по нескольку сегментов;
- `huge` — позволяет использовать сегментов больше, чем потенциально может поместиться в оперативной памяти.

Пример:

```
1  .model small
2
3  .stack 100h
4
5  .data
6          str1 db 'Line1$'
7
8  .code
9  begin:
10         ; инициализировать DS
11         mov AX, @data
12         mov DS, AX
13
14         ; вывести строку
15         mov AH, 09h
16         mov DX, offset str1
17         int 21h
18
19         ; выйти из программы
20         mov AX, 4C00h
21         int 21h
22 end begin
```

2 Часть 2.

2.1 Команды безусловной передачи управления. Обращение к подпрограммам и возврат из них.