Полетаев Игорь Алексеевич
Полетаева Ольга Александровна

ОРГАНИЗАЦИЯ ЭВМ

Методические указания к лабораторным работам
Для студентов специальности 230101

Технический редактор И.А.Полетаев
Компьютерная верстка И.А.Полетаев

Формат 60x84/16. Гарнитура Bookman. Уч. изд. п.л. 4,4
Тираж 100 экз. Заказ _____

Адрес издательства:
Россия, 180680, Псков, ул. Л.Толстого 4.
Издательство ППИ

Министерство образования и науки Российской Федерации
Федеральное агентство по образованию

ПСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

И.А. Полетаев, О.А. Полетаева

Организация ЭВМ

Методические указания к лабораторным работам

Рекомендовано к изданию научно-методическим советом Псковского государственного политехнического института

Псков «ППИ»
2005

УДК 681.32
П 49
ББК 32.973.26-02

Рецензенты: Козловский Г.В., Каган Р.Л.

Полетаев И.А., Полетаева О.А.
Организация ЭВМ: Методические указания к лабораторным работам. ППИ, 2005 – 76 с., ил.

В методических указаниях приведены основные сведения по логической и структурной организации персональных компьютеров. Основное внимание уделяется организации процессоров архитектуры фирмы Intel: представление информации, режимы адресации, структура команд и так далее. Каждой из шести лабораторных работ предшествует вводная часть с описанием теоретических положений.

Целью выполнения работ является изучение организации персонального компьютера программными средствами, то есть для практического освоения курса предусматривается работа на IBM-совместимых персональных компьютерах.

Методические указания предназначены для практических занятий (лабораторных работ) по дисциплине «Организация ЭВМ» (ОПД.Ф.08) для студентов специальности 230101 «Вычислительные машины, комплексы, системы и сети».

УДК 681.32
П 49
ББК 32.973.26-02

Список рекомендуемой литературы

М.Гук, В.Юров. Процессоры Pentium 4, Athlon и Duron. – СПб.: Питер, 2001. – 512 с.: ил.

М.Гук. Аппаратные средства IBM PC. Энциклопедия, 2-е изд. – СПб.: Питер, 2003. – 928 с; ил.

Э.Таненбаум. Архитектура компьютера. – СПб.: Питер, 2003.

В.Л. Григорьев. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). – М.: ГРАНАЛ, 1993.

П.Брамм, Д.Брамм. Микропроцессор 80386 и его программирование. – М.: Мир, 1990, – 448 с., ил.

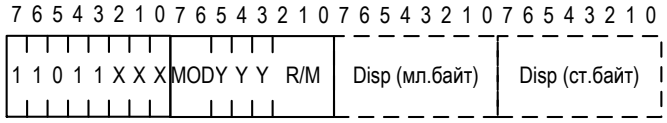
Ю-Чжен Лю, Г.Гибсон. Микропроцессоры семейства 8086/8088. – М.: Радио и связь, 1987. – 512 с.: ил.

Л.Дао. Программирование микропроцессора 8086. – М.: Мир, 1988. – 357 с.: ил.

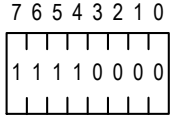
Рудометов Е., Рудометов В. Материнские платы и чипсеты. – СПб.: Питер, 2000. – 256 с.: ил.

Юров В. Assembler. Учебник. – СПб.: Питер, 2000. – 624 с.: ил.

ESC – переключиться на внешнее устройство (сопроцессор, поля XXX и YYY – код операции ESC)



LOCK – префикс блокировки шины



СОДЕРЖАНИЕ

1. ЛАБОРАТОРНАЯ РАБОТА № 1. Определение конфигурации и производительности ПЭВМ программными средствами 4
2. ЛАБОРАТОРНАЯ РАБОТА № 2. Представление информации в ЭВМ 13
3. ЛАБОРАТОРНАЯ РАБОТА № 3. Пользовательские регистры процессора, язык ассемблер, компоновщик и отладчик 20
4. ЛАБОРАТОРНАЯ РАБОТА № 4. Режимы адресации информации 27
5. ЛАБОРАТОРНАЯ РАБОТА № 5. Структура команд процессора 34
6. ЛАБОРАТОРНАЯ РАБОТА № 6. Регистр флагов процессора 40
7. Приложение А. Машинные коды команд микропроцессоров i8086/88 49
8. Список рекомендуемой литературы 75

ЛАБОРАТОРНАЯ РАБОТА № 1

Определение конфигурации и производительности ПЭВМ программными средствами

Одной из важных характеристик персональных ЭВМ является состав и технические характеристики оборудования, в частности тип и тактовая частота процессора, объем основной и внешней памяти, набор и типы внешних устройств и так далее.

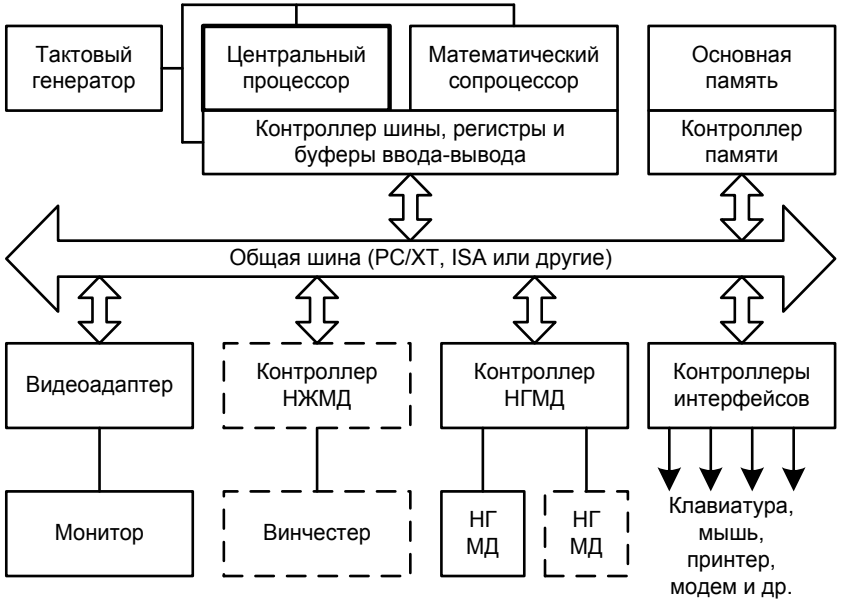
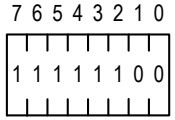


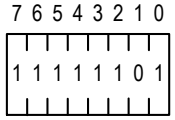
Рис. 1. Общая структура ПЭВМ IBM PC/XT/AT.

Эти характеристики напрямую связаны со структурой ЭВМ. Персональные ЭВМ прошли несколько этапов своего развития, их структура претерпевает изменения. Первые **персональные IBM-совместимые компьютеры** имели магистральную архитектуру, или архитектуру с общей шиной.

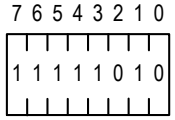
CLD – сбросить направление



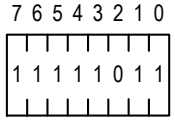
STD – установить направление



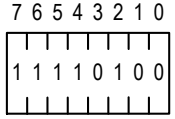
CLI – сбросить прерывание



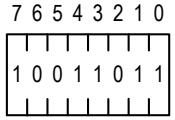
STI – установить прерывание



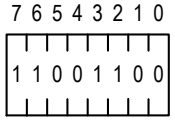
HLT – остановить



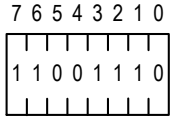
WAIT – ожидать



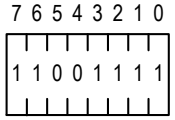
INT3 – вызвать прерывание типа 3



INT0 – вызвать прерывание если есть переполнение

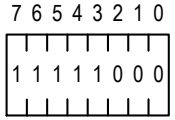


IRET – возвратиться из прерывания

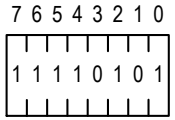


Команды управления процессом

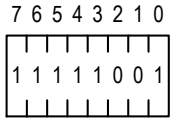
CLC – сбросить перенос



CMC – инвертировать (дополнить) перенос



STC – установить перенос



Все устройства: процессор, память, накопители на жестких и гибких магнитных дисках, монитор и другие подключались к системному интерфейсу, – общей шине, через контроллеры или адаптеры. Общая структура такой ПЭВМ представлена на рис. 1.

Так как интерфейс «Общая шина» менялся от одного типа ПЭВМ к другому, то приходилось для каждого устройства использовать разные контроллеры и адаптеры. Набор таких контроллеров и дополнительных устройств представлял собой набор микросхем, – **чипсет** (Chip Set), а их количество составляло 50-80 штук. В дальнейшем, благодаря повышению степени интеграции микросхем, их количество дошло до двух, а в некоторых случаях до одной микросхемы, по традиции называемых чипсетом.

Так как общая шина могла быть разной, то не любые устройства можно было использовать с данным типом компьютера, – требовалась унификация. Сначала в качестве такого стандарта для IBM-совместимых ПЭВМ выступали шины ISA и ее расширение EISA (Extended Industry Standard Architecture), но основные потребители пропускной способности интерфейса, – процессоры и память, непрерывно совершенствовались. Поэтому фирма Intel с выпуском процессоров Pentium в 1992 году предложила процессорно-независимую шину PCI (Peripheral Component Interconnect local bus), предназначенную только для подключения различных устройств, – **консольную шину**. Процессор с памятью стали общаться напрямую через чипсет, а точнее часть чипсета – переходную схему локальных интерфейсов процессора и памяти: мост процессор-память.

В итоге именно чипсет стал определять структуру ПЭВМ, которая и приводилась при описании чипсета, например, в соответствии с рис. 2.

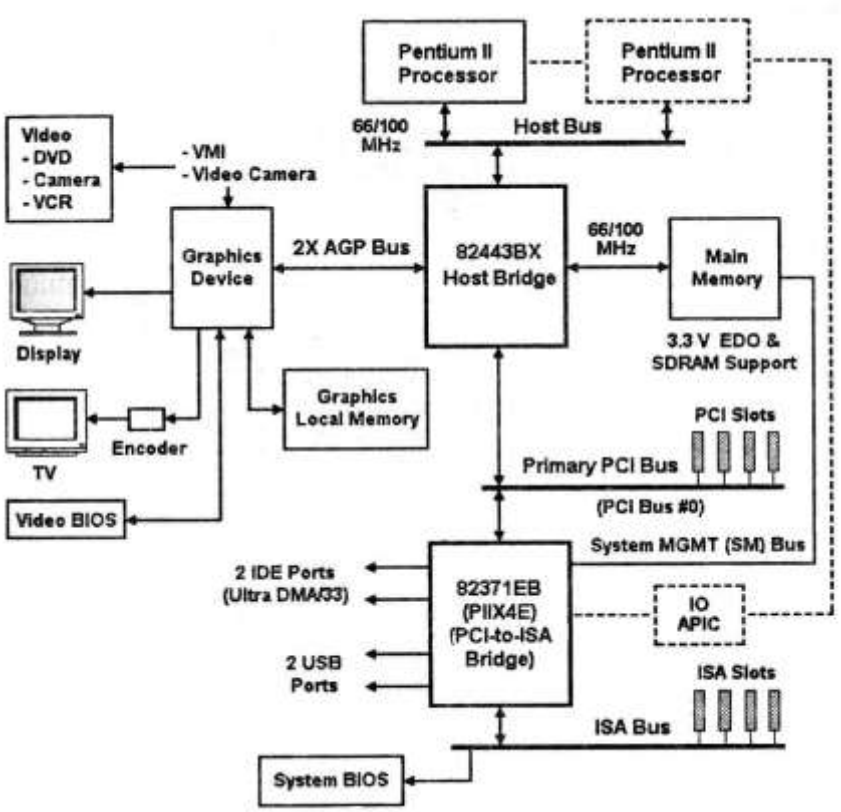
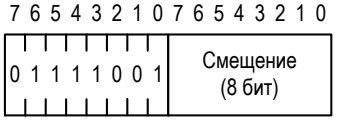


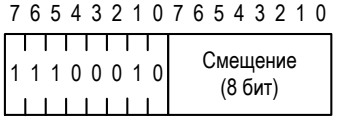
Рис 2. Структура компьютера с чипсетом i440BX AGPset.

Так как шина PCI делила структурную схему компьютера приблизительно на две половины: устройства с быстродействующими интерфейсами и со своей управляющей микросхемой (половиной чипсета) изображались сверху, а более медленные периферийные устройства с другой половиной чипсета – снизу. Поэтому консольную шину PCI стали называть экватором, а части чипсета – **северным и южным мостами** (преобразователями одних интерфейсов в другие). Это наглядно отражается на структурах компьютеров, предлагаемых разработчиками чипсетов, например, в соответствии с рис. 3.

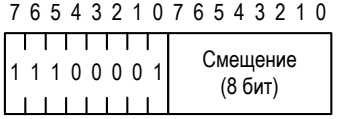
JNS – перейти, если знак сброшен



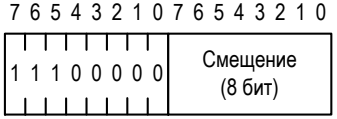
LOOP – зациклить (CX) раз



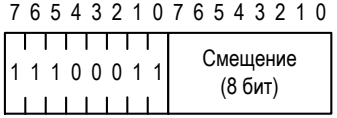
LOOPZ/ LOOPE – зациклить, пока нуль/ равно



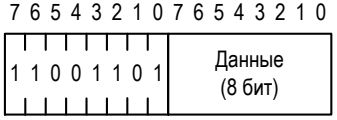
LOOPNZ/ LOOPNE – зациклить, пока не нуль/ не равно



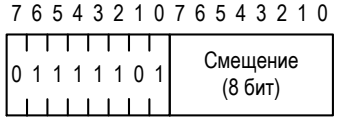
JCXZ – перейти, если (CX) равно нулю



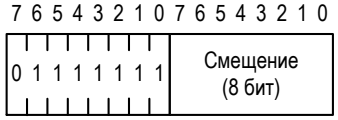
INT – вызвать прерывание определенного типа



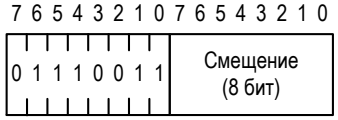
JNL/ JGE – перейти, если не меньше/ больше или равно



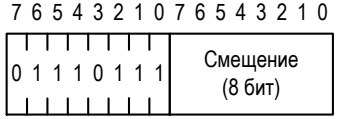
JNLE/ JG – перейти, если не меньше или равно/ больше



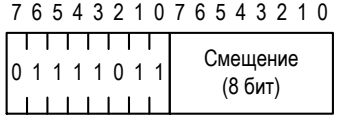
JNB/ JAE – перейти, если не ниже/ выше или равно



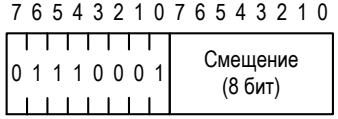
JNBE/ JA – перейти, если не ниже или равно/ выше



JNP/ JPO – перейти, если паритет сброшен (нечетный)



JNO – перейти, если нет переполнения



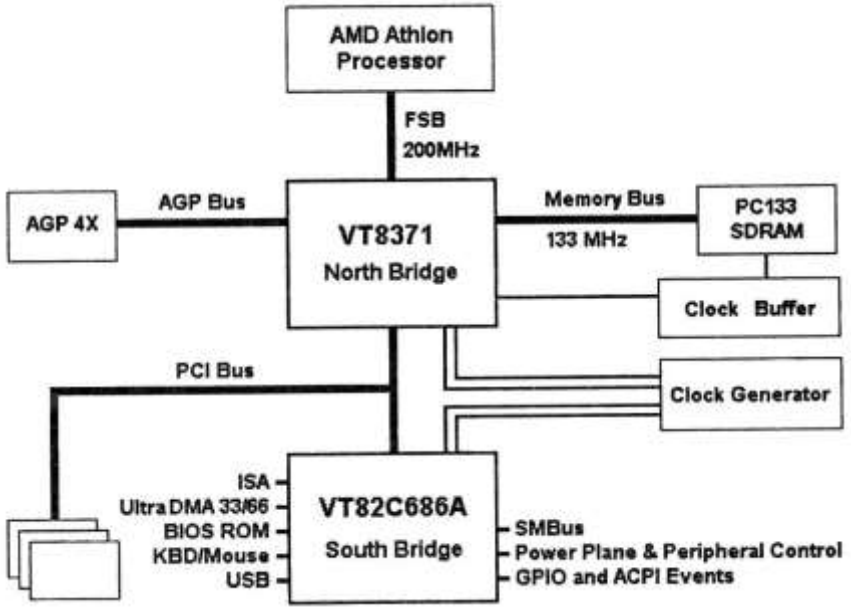


Рис. 3. Структура компьютера с чипсетом VIA Apollo KX133

Если сначала пропускной способности PCI в 132 Мбайт/с (есть модификации со скоростями до 528 Мбайт/с) хватало для всех периферийных устройств, то скоро для некоторых устройств этот интерфейс себя исчерпал. Первым оказался видеографический интерфейс: его с шины PCI перенесли в северный мост, увеличив пропускную способность. Скорость обмена северного с южным мостом необходимо было увеличить, стали между ними применять другие интерфейсы. Сейчас интерфейс PCI перешел в разряд периферийных интерфейсов, так же, как на рис. 2,3 интерфейс ISA.

Пример структуры компьютера, использующего для связи с периферийными устройствами в основном последовательные интерфейсы, приведен на рис. 4.

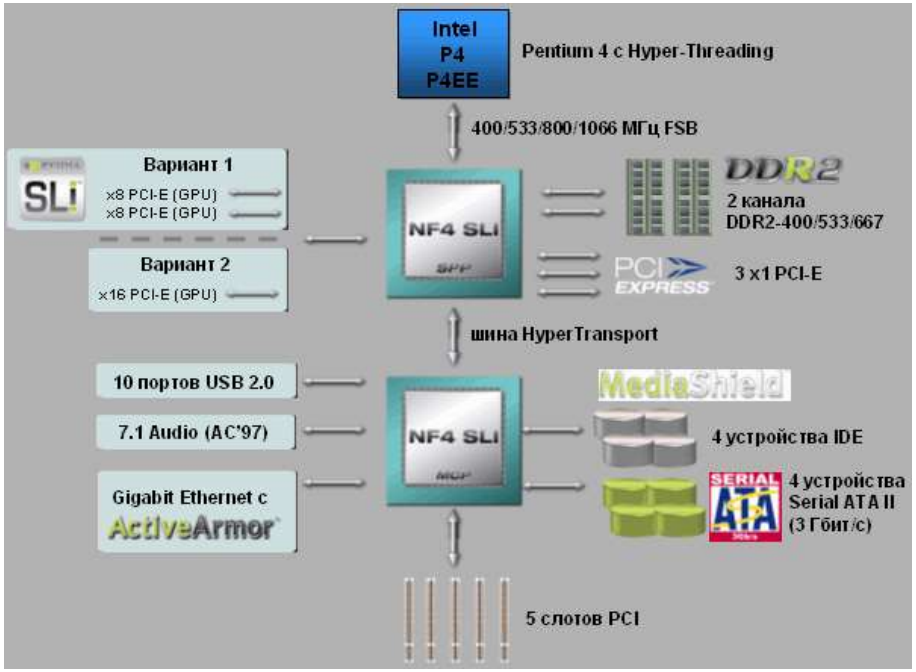
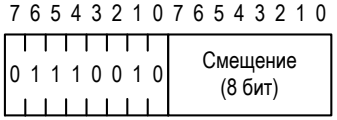


Рис. 4. Структура компьютера с чипсетом nForce4 SLI Intel Edition

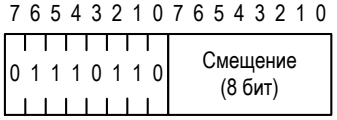
На рис. 2-4 представлены общие структуры ЭВМ с использованием определенного чипсета, реальная же структура отдельно взятой системной платы может быть либо урезана, либо расширена за счет установки дополнительных микросхем. Кроме этого в **слоты** (разъемы для установки плат) могут быть установлены самые разнообразные платы расширения, существенно влияющие на функции ПЭВМ.

Чтобы определить состав оборудования, то есть конфигурацию ЭВМ, необходимо иметь под рукой расширенную спецификацию компьютера, что бывает не всегда доступно. Но благодаря тому, что данная информация об аппаратной части хранится в специальных регистрах или может быть протестирована, состав оборудования ПЭВМ можно определить программно. Из-за важности данной информации существует множество программ, определяющих конфигурацию ПЭВМ, в том числе и включаемых в операционные системы или программные оболочки в качестве утилит.

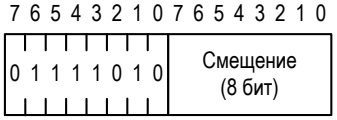
JB/ JNAE – перейти, если ниже/ не выше или равно



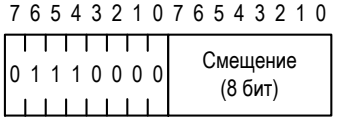
JBE/ JNA – перейти, если ниже или равно/ не выше



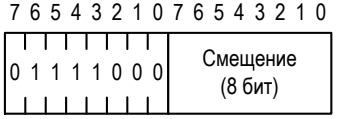
JP/ JPE – перейти, если паритет установлен (четный)



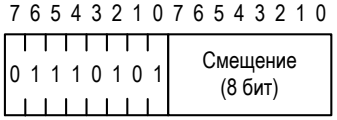
JO – перейти, если есть переполнение



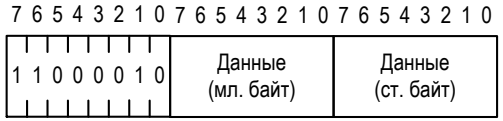
JS – перейти, если знак установлен



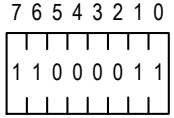
JNE/ JNZ – перейти, если не равно/ не нуль



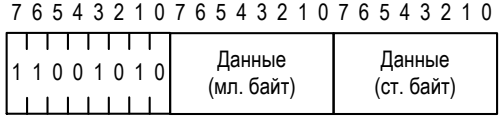
RET – возврат из процедуры внутрисегментный с прибавлением непосредственного операнда к SP



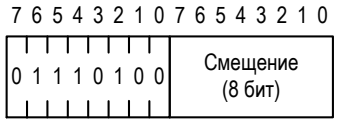
RET – возврат из процедуры межсегментный



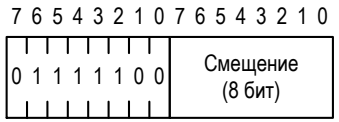
RET – возврат из процедуры межсегментный с прибавлением непосредственного операнда к SP



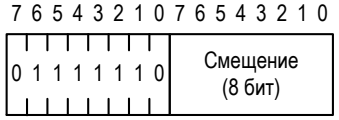
JE/ JZ – перейти, если равно/ нуль



JL/ JNGE – перейти, если меньше/ не больше или равно



JLE/ JNG – перейти, если меньше или равно/ не больше



Одними из первых для IBM-совместимых компьютеров использовались следующие утилиты. System Information (SYSINFO), которая определяет и отображает технические характеристики ПЭВМ, включалась в программную оболочку NORTON COMMANDER. В операционную систему, начиная с версии DOS 6.2, так же включалась информационная утилита Microsoft Diagnostics (MSD). По представляемой информации она похожа на System Information.

Операционные системы семейства Windows имеют встроенные средства определения аппаратной и программной части компьютеров. Основные сведения можно получить, например, если выбрать из меню по правой кнопке мыши «Мой компьютер>Свойства».

Но наиболее полную информацию дают специализированные утилиты, распространяемые как на платной основе, так и бесплатно. К ним относятся AIDA32, AMD Identifier, AOConfig, ASTRA, Atomic CPU Test, BIOS Agent, Central Brain Identifier, Chipinfo, CPU Identification, CpuIDMax, CPUinfo, CPU-Z, Crystal Mark, EVEREST, Fresh Diagnose, GCPUID, MetaBench, OverSoft CPU Informer, PC Wizard, Performance Test, RightMark, SiSoftware Sandra, System Information for Windows и многие другие.

Эти программы-утилиты позволяют получить не только базовую (запрограммированную) информацию по основным характеристикам отдельных устройств, но и, используя встроенные базы данных, определяются фирмы-производители, год начала выпуска, многие дополнительные характеристики, включая Web-адрес фирм-производителей конкретного устройства.

Так, по кодовой строке процессора, пользуясь базой данных, можно определить множество параметров:

1. Полное название процессора.
2. Семейство, к которому относится процессор.
3. Условное обозначение ядра процессора.
4. Номер модели процессора.
5. Фирму производитель.
6. Платформу (тип разъема, количество выводов).
7. Частоту процессора.
8. Частоту системной шины.
9. Коэффициент умножения частоты в процессоре.
10. Размер встроенного Кеша отдельно по уровням, для первого уровня отдельно по командам и данным.

11. Дополнительную информацию по организации кеш-памяти.
12. Размер элементов на микросхеме (проектные нормы)
13. Дату начала выпуска процессора.
14. Наличие дополнительных технологий обработки данных (MMX, SSE и др.).
15. Возможность понижения энергопотребления и так далее.

Это же справедливо и для других устройств: чипсета (северный и южный мост), видеосистемы (встроенного в северный мост видеоадаптера или отдельной видеоплаты), памяти, системной платы, накопителей на жестких и оптических дисках и так далее.

Наряду с составом оборудования, производительность является одной из основных характеристик ПЭВМ и непосредственно связана со стоимостью.

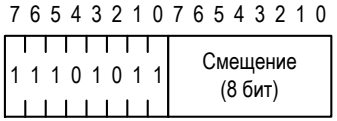
Традиционно для универсальных ЭВМ, включая персональные, основной считается производительность процессора. Она выражается либо в миллионах коротких инструкций в секунду (**MIPS**, Mega Instructions Per Second), либо, при наличии аппаратной поддержки, – математического сопроцессора, – в операциях с плавающей точкой в секунду (**FLOPS**, Floating Point Operations Per Second), обычно операций умножения.

Эти характеристики зависят как от тактовой частоты процессора, так и от количества тактов, необходимых для выполнения операций. Данные для процессоров фирмы Intel приведены в таблицах 1 и 2.

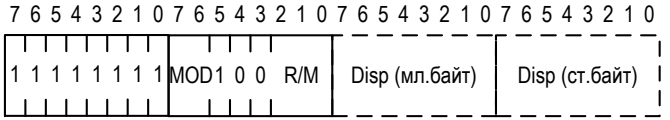
Таблица 1. Минимальное время выполнения коротких операций.

Тип процессора	Количество тактов
8086	2
80286	2
80386	1
80486	1
Pentium	2 команды за такт
Pentium Pro	3 команды за такт
Pentium II	3 команды за такт
Pentium III	3 команды за такт
Pentium 4	3 команды за такт

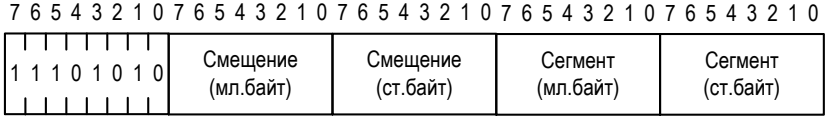
JMP – безусловный переход прямой внутрисегментный короткий



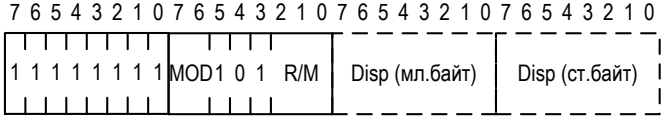
JMP – безусловный переход косвенный внутрисегментный



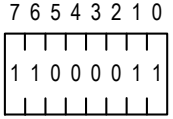
JMP – безусловный переход прямой межсегментный



JMP – безусловный переход косвенный межсегментный

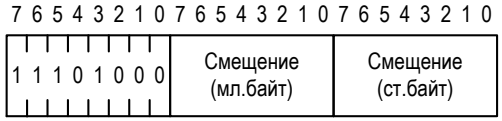


RET – возврат из процедуры внутрисегментный

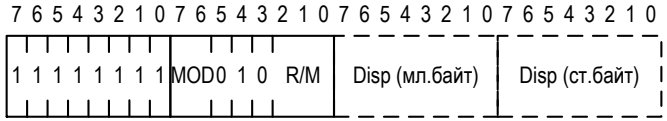


Команды передачи управления

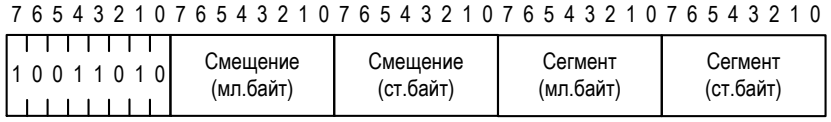
CALL – вызов процедуры прямой внутрисегментный



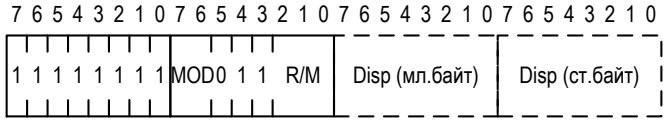
CALL – вызов процедуры косвенный внутрисегментный



CALL – вызов процедуры прямой межсегментный



CALL – вызов процедуры косвенный межсегментный



JMP – безусловный переход прямой внутрисегментный

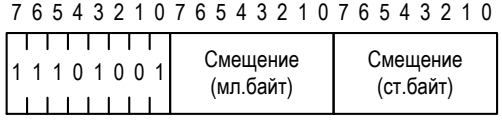


Таблица 2. Минимальное время выполнения операций с плавающей точкой для сопроцессоров.

Тип сопроцессора	Количество тактов	
	Сложение	Умножение
8087	70	90
80287	70	90
80387	20	25
80486DX	10	11
Pentium и выше	1	1

Характеристики в MIPS являются весьма условными, так как время выполнения различных команд может существенно различаться. Например, для микроконтроллеров семейства PIC16 фирмы Microchip все команды (кроме команд переходов) выполняются за один такт. В то же время для процессора i80286 даже команда безусловного длинного перехода может выполняться 42 такта.

Более того, одна и та же команда может выполняться различное время из-за разных методов доступа к операндам, нахождения данных в кеше, загруженности конвейера команд и других условий. Поэтому для оценки производительности ПЭВМ используют интегральную характеристику, как функцию вероятности выполнения каждой команды, которая сильно зависит от выполняемой задачи.

Таким образом, для оценки производительности ПЭВМ используют тестовые программы, - смеси, - ориентированные на конкретный класс задач. Наряду с тестированием непосредственно процессора, используются тесты, работающие совместно с дисководами, видеосистемами и другими устройствами. Более того, существуют отдельные программы-утилиты, тестирующие отдельно память, накопители на жестких, оптических дисках и другие.

Например, при имитации работы в текстовом редакторе Word тестируются операции открытия и закрытия файлов, выделения, разного форматирования текста и смены шрифтов. Время выполнения теста в основном зависит то НЖМД и видеоадаптера.

В системе электронных таблиц Excel проводятся пересчет таблицы, ее форматирование, построение графиков. Время теста зависит от скорости выполнения операций с плавающей точкой, быстродействия видеоадаптера и НЖМД.

Для языка Borland C++ тестируются операции от создания проекта до сборки исполняемого модуля. Время выполнения зависит в основном от скорости процессора и дисковой системы.

В любом случае такая оценка производительности является чисто статистической и дается не в абсолютных единицах, а в сравнении с другими типами компьютеров.

Тестовые программы выпускаются как отдельно, так зачастую, входят в состав утилит, определяющих состав ЭВМ. При этом они могут работать (и выдавать адекватные данные для сравнения) только на одной платформе и под определенным классом операционных систем.

Но выпускаются и универсальные тестовые программы, создаваемые на языке высокого уровня (обычно Си), а затем компилируемые под определенную платформу и класс операционных систем. Наиболее известным разработчиком таких программ является некоммерческая организация System Performance Evaluation Corporation. Она распространяет и публикует программы для разных прикладных областей и результаты тестирования многих моделей компьютеров. Для компьютеров общего назначения набор тестовых приложений был определен в 1989 году, затем он модифицировался в 1995 и 2000 годах. В них в качестве эталонного компьютера использовались для SPEC95 SUN SPARCstation 10/40, а для теста SPEC2000 – UltraSPARC10.

Задание.

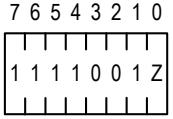
1. Исходя из информации, полученной о компьютере из утилит, указанных преподавателем, построить структурную схему по примеру рис. 2-4, но с указанием подключенных устройств и их основных технических характеристик для компьютера, используемого при выполнении лабораторной работы. Кроме этого отдельно указать расширенные характеристики по каждому устройству (если они имеются и только по аппаратной части).

2. Рассчитать производительность процессора и сопроцессора соответственно в MIPS и FLOPS.

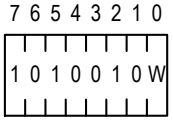
3. По информации утилиты, указанной преподавателем, построить таблицу оценки производительности устройств ПЭВМ по сравнению с другими типами компьютеров.

Команды манипуляции цепочками

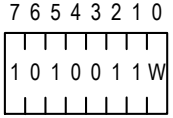
REP – повторить



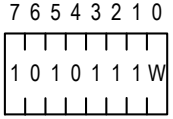
MOWS – передать байт или слово



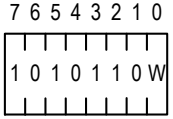
CMPS – сравнить байт или слово



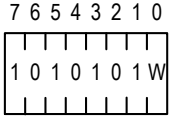
SCAS – сканировать байт или слово



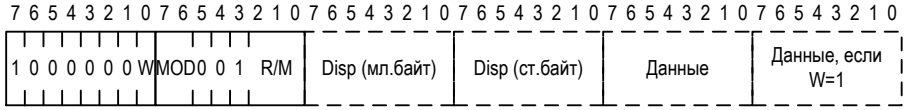
LODS – загрузить байт или слово в AL/ AX



STOS – запомнить байт или слово из AL/ AX

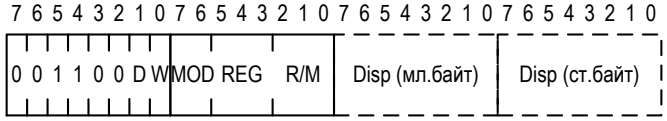


OR – объединить по «ИЛИ» непосредственный операнд с регистром или памятью

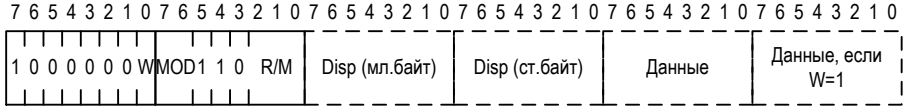


OR – объединить по «ИЛИ» непосредственный операнд с аккумулятором

XOR – сложить по модулю 2 (исключающее «ИЛИ») регистр или память с регистром и запомнить в любом из них



XOR – сложить по модулю 2 непосредственный операнд с регистром или памятью



XOR – сложить по модулю 2 непосредственный операнд с аккумулятором

ЛАБОРАТОРНАЯ РАБОТА № 2

Представление информации в ЭВМ

Любая информация, будь то числа, текст, графика и так далее, представляется в двоично-кодированном виде. Большинство современных универсальных процессоров аппаратно, на уровне команд, поддерживают несколько типов представления информации.

Например, в компьютерах с использованием **процессоров 32-разрядной архитектуры фирмы Intel (в дальнейшем IA-32)** информация представляется тремя большими классами.

1. Числа. Целые числа представлены двоичными числами длиной 8, 16 или 32 бита как со знаком, так и без знака, и двоично-кодированными десятичными числами без знака длиной 8 бит. Последние могут иметь как упакованный, так и неупакованный формат (используется код ASCII).

Благодаря наличию арифметического сопроцессора обеспечивается аппаратная поддержка дополнительных типов чисел: двоичные целые числа со знаком длиной 64 бита, три формата вещественных чисел и целые упакованные 80-битные десятичные числа.

2. Указатели. Они применяются при обращении к некоторым объектам в памяти, например адресам подпрограмм. Близкий или внутрисегментный указатель - это 32-битное смещение, далекий или межсегментный указатель применяется при передаче управления в другой сегмент и имеет длину 48 бит.

3. Цепочки (для байт – строки). Аппаратно поддерживаются цепочки бит, байт, 16- и 32-разрядных слов. Здесь под цепочкой понимается последовательность практически любой длины данных, хранящихся по соседним адресам. Например, текст, представленный в кодах ASCII, - это цепочка байт. Логический массив может быть представлен цепочкой бит. То есть любой массив, содержащий числовую, символьную, графическую и другую информацию, представляется в виде цепочки данных с определенным размером элементов и может быть обработан цепочечными операциями.

Целые числа со знаком представляются в коде, дополняющем до двух, то есть старший бит числа является признаком знака. Плюс ноль и минус ноль совпадают. Расширение со знаком выполняется элементарно просто, смена знака числа в обоих направлениях одинакова и при использовании логических операций занимает два шага:

инверсия,
инкремент.
Например +6 преобразуем в -6 и обратно в +6:

Код числа +6:	0000 0110
инверсия:	1111 1001
инкремент	1111 1010 – код числа -6
инверсия	0000 0101
инкремент	0000 0110 – код числа +6

Вещественные числа, аппаратно обрабатываемые в сопроцессорах, имеют три формата: короткое (4 байта), длинное (8 байт) и временное (10 байт) вещественные. В первом и втором форматах используется **скрытый бит**, то есть старший бит мантиссы физически не хранится.

Для короткого вещественного используется формат, приведенный на рис. 5. Заметьте, что граница порядка с мантиссой и граница байта **не совпадают**.

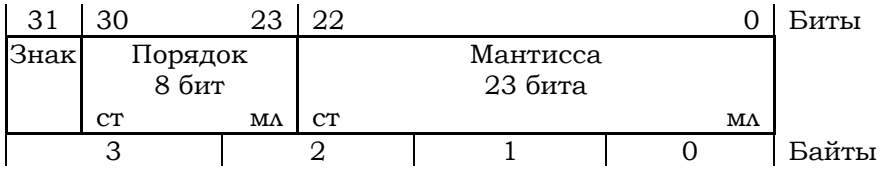
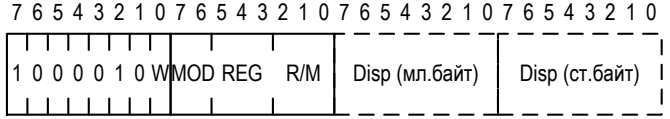


Рис. 5. Представление короткого вещественного в памяти ЭВМ.

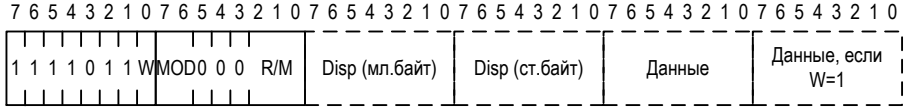
Знак мантиссы расположен отдельно, сама же она представляется положительным числом. Порядок представлен в виде числа **со смещением, которое равно 127** для одинарной точности, 1023 для двойной и 16 383 для расширенной.

AND – объединить по «И» непосредственный операнд с аккумулятором

TEST – проверить регистр или память и регистр (логическое «И» без запоминания результата)

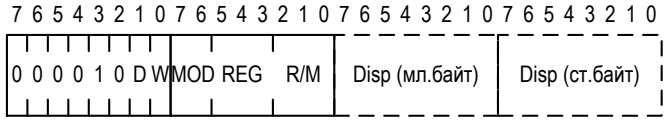


TEST – проверить непосредственный операнд и регистр или память

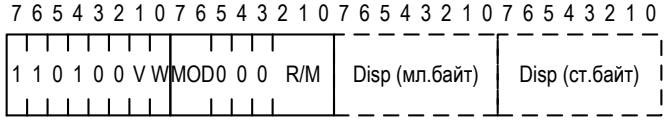


TEST – проверить непосредственный операнд и аккумулятор

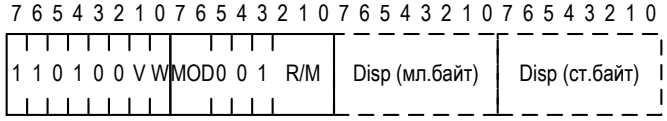
OR – объединить по «ИЛИ» регистр или память с регистром и запомнить в любом из них



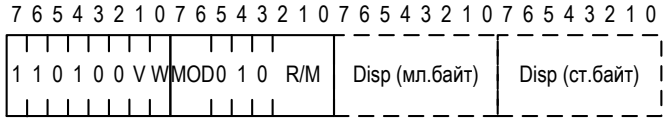
ROL – сдвинуть циклически влево



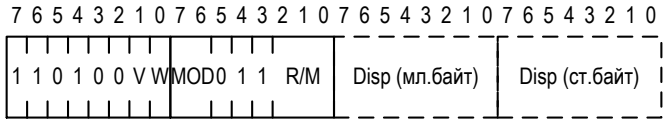
ROR – сдвинуть циклически вправо



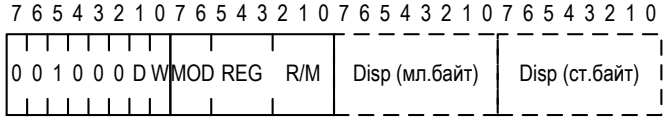
RCL – сдвинуть циклически влево через перенос



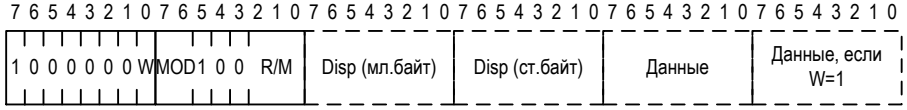
RCR – сдвинуть циклически вправо через перенос



AND – объединить по «И» регистр или память с регистром и запомнить в любом из них



AND – объединить по «И» непосредственный операнд с регистром или памятью



Пример: представление десятичного числа -243.375.
-11110111.011 - двоичное представление;

$-1.1110111011 \times 2^{+7}$ - нормализованное представление в форме с плавающей точкой;

Знаковый разряд: 1 (число отрицательное);

Мантисса со скрытым битом (первая единица с точкой не записываются – скрытый бит), в поле мантиссы составляет 23 разряда, включая незначащие правые нули:
1110111 01100000 00000000

Порядок: $127+7 = 134_{10} = 10000110_2$

Полное число в приведенном выше формате:
1 10000110 1110111 01100000 00000000

Так как при просмотре дампа памяти или файла байты располагаются по возрастанию адресов слева направо (младшие разряды по младшим адресам), то на экране данное число в двоичной форме могло бы быть отображено следующим образом:

00000000 01100000 01110111 11000011

но фактически отображается в шестнадцатеричной форме:

00 60 77 C3

Для просмотра содержимого файла можно воспользоваться в Norton Commander командой View (F3). Эта команда хотя и предназначена для визуального просмотра текстовых файлов (view – визуализатор файлов), но, благодаря тому, что она имеет возможность представлять файлы в разных форматах, можно переключиться с помощью команды Viewer (F8) на выбор любого доступного формата.

Для просмотра файла в шестнадцатеричном виде можно использовать визуализатор HEX (код/текст, F4). После его выбора содержимое файла изображается в виде двух колонок: в левой части побайтное шестнадцатеричное представление по 16 байт в строке, в правой - изображение в символах ASCII.

Нестандартное вещественное специфично для языка Паскаль и обрабатывается программными средствами этого языка.

Формат числа следующий:

старшие пять байт - мантисса со скрытым битом, знак в старшем бите мантиссы, младший байт - порядок **со смещением 129**.

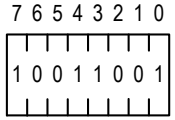
Задание.

Определить, какая информация представлена в файле lr2_<номер варианта>.dat. Фрагменты копии экрана (скриншоты) файлов представлены в конце задания в соответствии с вариантами.

Файл образован программой на языке Паскаль, то есть обозначения типов приводятся в терминах этого языка программирования. Здесь же указано побайтное расположение данных в файле:

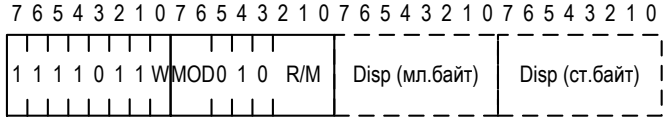
```
program type_data;
Type
  CH=array [1..16] of char;
  T=record
-------------------------------------------------
tekst:CH;        {строка 16 символов, 1й-16й байты}
ShorP:shortint;  {однобайтное целое со знаком,
                 17й байт}
ShorN:shortint;  {однобайтное целое со знаком,
                 18й байт}
IntP:integer;    {двухбайтное целое со знаком,
                 19й-20й байты}
LongN:longint;   {четырехбайтное целое со знаком,
                 21й-24й байты}
SingP:single;    {стандартное вещественное,
                 25й-28й байты}
```


CWD – преобразовать слово в двойное слово

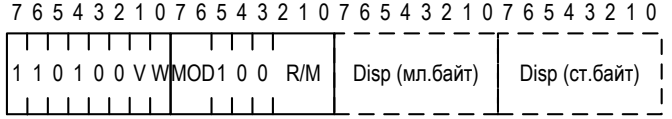


Логические команды

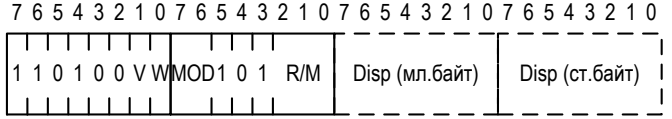
NOT – инвертировать



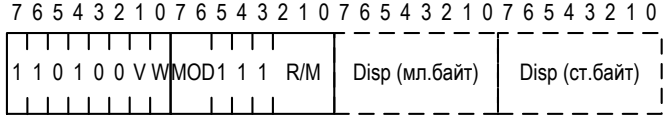
SHL/ SAL – сдвинуть арифметически или логически влево



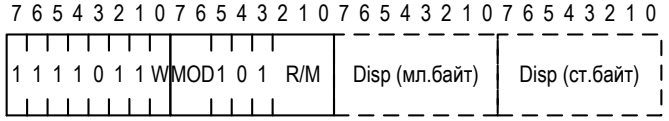
SHR – сдвинуть логически вправо



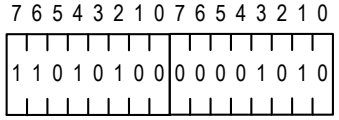
SAR – сдвинуть арифметически вправо



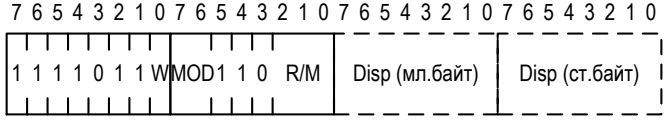
IMUL – умножить со знаком



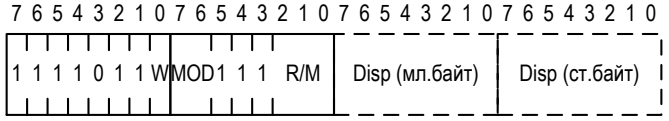
AAM – ASCII-коррекция для умножения



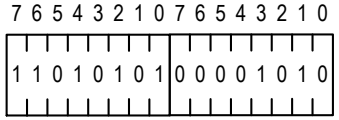
DIV – разделить без знака



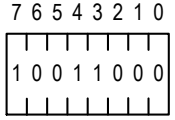
IDIV – разделить со знаком



AAD – ASCII-коррекция для деления



CWB – преобразовать байт в слово




```
  SingN:single;    {стандартное вещественное,
                   29й-32й байты}
  ------------------------------------------------
  RealP:real;      {нестандартное вещественное,
                   33й-38й байты}
  RealN:real;      {нестандартное вещественное,
                   39й-44й байты}
  PrimP:real       {Пример: PrimP=1.5 (1.1B),
                   45й-50й байты}
  ------------------------------------------------
    end;
Var X:T;
    f:file of T;
    fin:text;
    fil_in,fil_out:string[12];
    num_lr:char;
begin
    X.tekst:='Строка символов ';
    X.PrimP:=1.5;
for num_lr:='1' to '15' do
 begin
     Assign(fin,'lr3_v'+num_lr+'.txt');
     Reset(fin);
     read(fin,X.ShorP,X.ShorN,X.IntP,X.LongN,
          X.SingP,X.SingN,X.RealP,X.RealN);
     close(fin);
     Assign(f,'lr3_v'+num_lr+'.dat');
     Rewrite(f);
     write(f,X);
     close(f)
 end
end.
```

В программе формируется запись, состоящая из:
- строки символов;
- двух однобайтных целых со знаком;
- двухбайтного целого;
- четырехбайтного целого;
- двух стандартных коротких вещественных;
- двух нестандартных вещественных длиной 6 байт;
- примера нестандартного вещественного.

Их побайтное расположение в файле описано в формирующей файл программе (см. выше).

В отчете указать: код символа "пробел", код двух любых других символов, значение чисел в десятичной системе счисления: ShorP, ShorN, IntP, LongN, SingP, SingN, RealP, RealN.

Далее приводятся дампы файлов в соответствии с вариантом задания.

1.
```
LR2_V1.DAT                                DOS          50    Кол 0     100%
00000000:  91 E2 E0 AE AA A0 20 E1 | A8 AC A2 AE AB AE A2 20  Строка символов
00000010:  19 D8 63 01 94 FF FF FF | 00 00 31 42 00 80 04 C2  ↓╪c☺Ф      1B А┬
00000020:  87 00 00 00 C0 0C 84 00 | 00 00 00 E8 81 00 00 00  З    └♀Д    шБ
00000030:  00 40                   |                          @
```

2.
```
LR2_V2.DAT                                DOS          50    Кол 0     100%
00000000:  91 E2 E0 AE AA A0 20 E1 | A8 AC A2 AE AB AE A2 20  Строка символов
00000010:  3F E9 16 01 7A FF FF FF | 00 00 09 42 00 80 5C C2  ?щ▬☺z      ┘B А\┬
00000020:  87 00 00 00 80 0D 85 00 | 00 00 00 94 81 00 00 00  З   А♪Е    ФБ
00000030:  00 40                   |                          @
```

3.
```
LR2_V3.DAT                                DOS          50    Кол 0     100%
00000000:  91 E2 E0 AE AA A0 20 E1 | A8 AC A2 AE AB AE A2 20  Строка символов
00000010:  1D EE 28 01 98 FF FF FF | 00 00 C6 41 00 80 24 C2  ∟ю+☺ў      ╞A А$┬
00000020:  86 00 00 00 00 59 86 00 | 00 00 00 BE 81 00 00 00  Ж    УЖ    ╛Б
00000030:  00 40                   |                          @
```

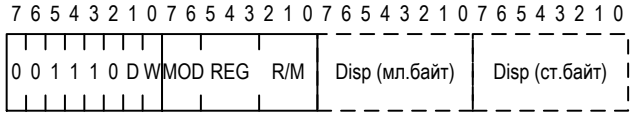
4.
```
LR2_V4.DAT                                DOS          50    Кол 0     100%
00000000:  91 E2 E0 AE AA A0 20 E1 | A8 AC A2 AE AB AE A2 20  Строка символов
00000010:  31 E9 37 01 90 FF FF FF | 00 00 FA 41 00 00 E9 C1  1щ7☺Р      ·A  щ┴
00000020:  86 00 00 00 00 1B 86 00 | 00 00 00 CE 81 00 00 00  Ж    ←Ж    ╬Б
00000030:  00 40                   |                          @
```

5.
```
LR2_V5.DAT                                DOS          50    Кол 0     100%
00000000:  91 E2 E0 AE AA A0 20 E1 | A8 AC A2 AE AB AE A2 20  Строка символов
00000010:  14 CD F4 00 7D FF FF FF | 00 00 07 42 00 80 18 C2  ¶═Ї }      B А↑┬
00000020:  85 00 00 00 00 6A 85 00 | 00 00 00 F4 81 00 00 00  Е    jЕ    ЇБ
00000030:  00 40                   |                          @
```

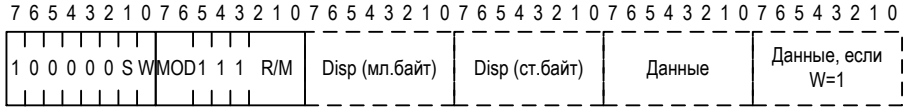
6.
```
LR2_V6.DAT                                DOS          50    Кол 0     100%
00000000:  91 E2 E0 AE AA A0 20 E1 | A8 AC A2 AE AB AE A2 20  Строка символов
00000010:  38 E7 C8 00 8A FF FF FF | 00 00 7F 42 00 00 89 C1  8ч╚ К      B  Йв
00000020:  85 00 00 00 00 14 85 00 | 00 00 00 D4 81 00 00 00  Е    ¶Е    ╘Б
00000030:  00 40                   |                          @
```

7.
```
LR2_V7.DAT                                DOS          50    Кол 0     100%
00000000:  91 E2 E0 AE AA A0 20 E1 | A8 AC A2 AE AB AE A2 20  Строка символов
00000010:  59 E5 BC 00 9B FF FF FF | 00 40 84 42 00 80 BF C2  Yх╝ ы      @ДВ А┐┬
00000020:  85 00 00 00 00 08 87 00 | 00 00 80 9A 81 00 00 00  Е    ◘З    АЪБ
00000030:  00 40                   |                          @
```


CMP – сравнить регистр или память с регистром

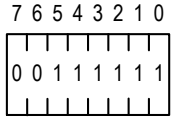


CMP – сравнить непосредственный операнд с регистром или памятью

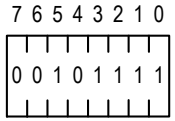


CMP – сравнить непосредственный операнд с аккумулятором

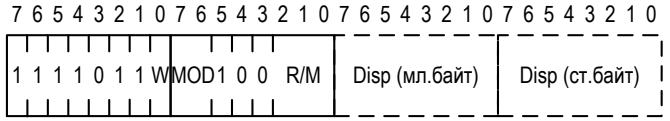
AAS – ASCII-коррекция для вычитания



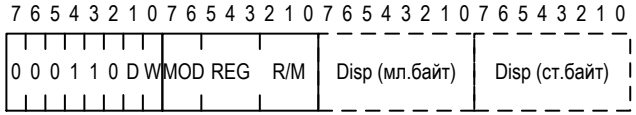
DAS – десятичная коррекция для вычитания



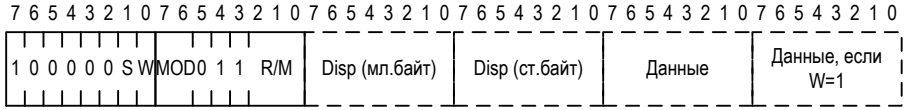
MUL – умножить без знака



SBB – вычесть с заемом регистр или память из регистра или памяти и запомнить в любом из них

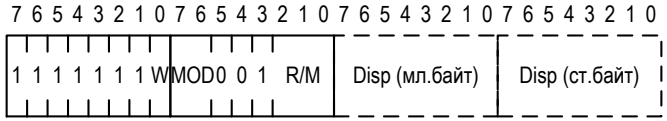


SBB – вычесть с заемом непосредственный операнд из регистра или памяти

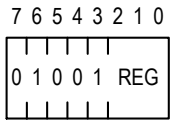


SBB – вычесть с заемом непосредственный операнд из аккумулятора

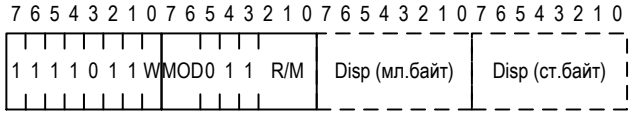
DEC – декремент регистра или памяти



DEC – декремент регистра



NEG – изменить знак



8.
```
LR2_V8.DAT                                    DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  16 EE 90 00 C9 FF FF FF  00 80 D8 41 00 00 4C C0   ▬юР ╔   А╪А  L└
00000020:  88 00 00 00 80 07 86 00  00 00 00 A7 81 00 00 00   и   А▒Ж    зБ
00000030:  00 40                                              @
```

9.
```
LR2_V9.DAT                                    DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  64 C5 BF 00 DF FF FF FF  00 00 69 41 00 80 31 C3   d┼┐ ▀     iA A1├
00000020:  87 00 00 00 80 21 84 00  00 00 00 C5 81 00 00 00   З   А!Д    ┼Б
00000030:  00 40                                              @
```

10.
```
LR2_V10.DAT                                   DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  53 D9 71 00 A1 FF FF FF  00 80 89 42 00 80 A2 C2   S┘q б    АЙВ АвТ
00000020:  86 00 00 00 40 3D 85 00  00 00 80 99 81 00 00 00   ж   @=Е    АЩБ
00000030:  00 40                                              @
```

11.
```
LR2_V11.DAT                                   DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  2F D9 5D 00 BE FF FF FF  00 80 0E 42 00 80 B2 C2   /┘] ╛    А▒В А▓Т
00000020:  87 00 00 00 80 66 85 00  00 00 00 C9 81 00 00 00   З   АfЕ    ┘Б
00000030:  00 40                                              @
```

12.
```
LR2_V12.DAT                                   DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  62 E8 D4 00 D5 FF FF FF  00 80 40 42 00 80 B8 C2   bы╘ ╒    А@В А╕Т
00000020:  86 00 00 00 40 1A 87 00  00 00 C0 98 81 00 00 00   ж   @→З    ЧЩБ
00000030:  00 40                                              @
```

13.
```
LR2_V13.DAT                                   DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  63 EB DD 00 D6 FF FF FF  00 80 8C 41 00 80 B1 C2   cы▌ ╓    АМА А▒Т
00000020:  86 00 00 00 80 1C 84 00  00 00 00 E5 81 00 00 00   ж   А∟Д    хБ
00000030:  00 40                                              @
```

14.
```
LR2_V14.DAT                                   DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  14 A3 4F 00 DB FF FF FF  00 00 CD 41 00 80 B4 C2   ¶гО ▄    ═А А┤Т
00000020:  86 00 00 00 00 7A 86 00  00 00 80 D8 81 00 00 00   ж   zж    А╪Б
00000030:  00 40                                              @
```

15.
```
LR2_V15.DAT                                   DOS      50    Кол 0    100%
00000000:  91 E2 E0 AE AA A0 20 E1  A8 AC A2 AE AB AE A2 20   Строка символов
00000010:  2C E1 48 00 CF FF FF FF  00 00 C9 41 00 80 B0 C2   ,сН ╧    ┘А А▒Т
00000020:  84 00 00 00 00 45 86 00  00 00 00 E5 81 00 00 00   Д   ЕЖ    хБ
00000030:  00 40                                              @
```

ЛАБОРАТОРНАЯ РАБОТА № 3

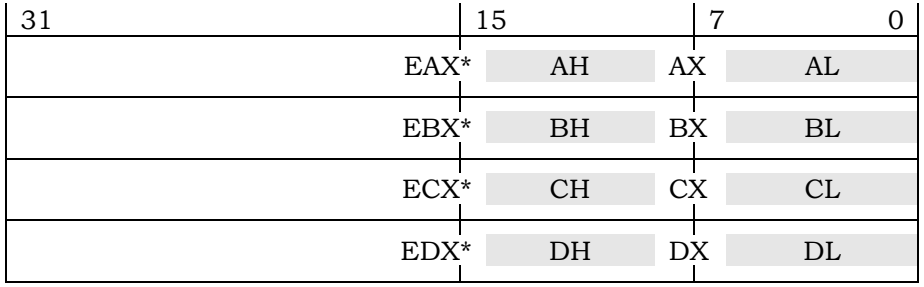
Пользовательские регистры процессора, язык ассемблер, компоновщик и отладчик

Как для детального ознакомления с архитектурой ЭВМ, так и для системного программирования необходимо знать весь перечень возможных для доступа регистров, областей ОЗУ и портов ввода - вывода. Одним из способов достижения этой цели является составление коротких программ на языке программирования, близком к машинному, например ассемблере. Для наглядности программа должна выполняться по шагам с оперативным отображением происходящих изменений, что позволяет делать отладчик (Debugger).

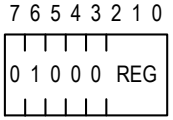
Линия микропроцессоров фирмы Intel 8086, 80286, 80386, 80486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, несмотря на различную разрядность и микроархитектуру, полностью совместима по принципу "снизу - вверх". Поэтому все основные регистры i8086 присутствуют и в i80386, и в Pentium 4, хотя в последних двух они расширены (начинаются с буквы E: от Extended – расширенный) до 32 разрядов.

Приведенные ниже основные (пользовательские) регистры, отсутствующие в i8086/286, отмечены звездочкой.

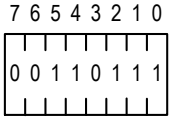
Регистры общего назначения - регистры данных.



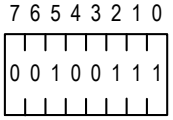
INC – инкремент регистра



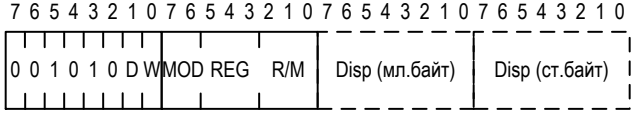
AAA – ASCII-коррекция для сложения



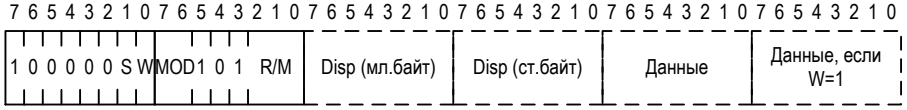
DAA – десятичная коррекция для сложения



SUB – вычесть регистр или память из регистра или памяти и запомнить в любом из них



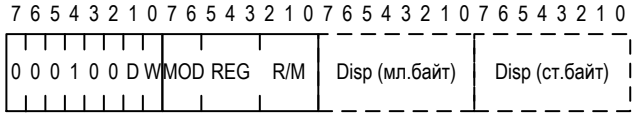
SUB – вычесть непосредственный операнд из регистра или памяти



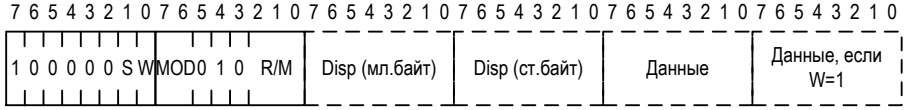
SUB – вычесть непосредственный операнд из аккумулятора

ADD – сложить непосредственный операнд с аккумулятором

ADC – сложить с переносом регистр или память с регистром и запомнить в любом из них

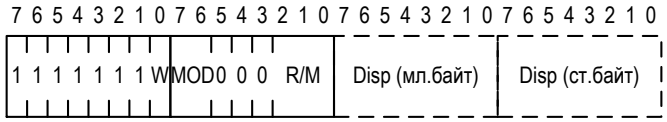


ADC – сложить с переносом непосредственный операнд с регистром или памятью

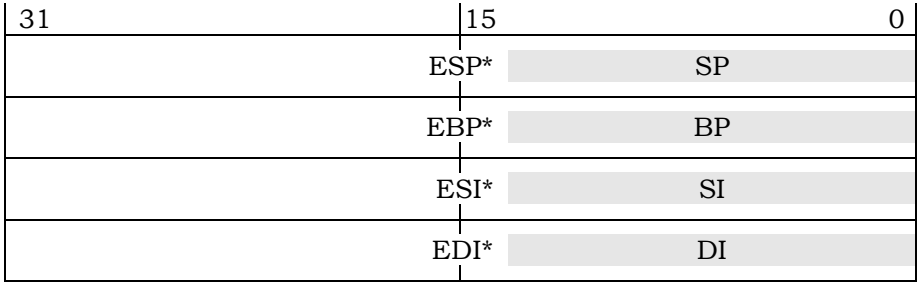


ADC – сложить с переносом непосредственный операнд с аккумулятором

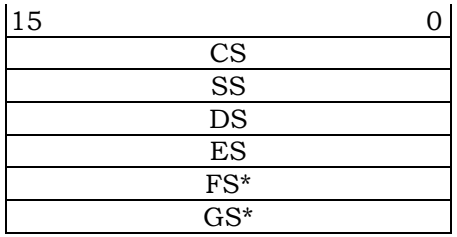
INC – инкремент регистра или памяти



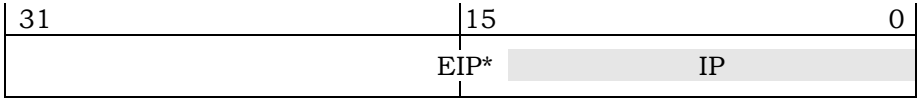
Регистры общего назначения - указательные и индексные регистры.



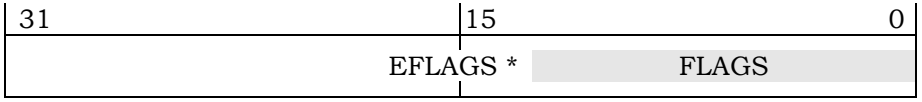
Сегментные регистры.



Указатель команды.



Регистр флагов.



Таким образом, например, регистр AX является младшей частью регистра EAX, AL - младшей частью регистра AX (L - Low, младший, H - High, старший) и младшим байтом четырехбайтного регистра EAX.

То есть процессор условно содержит восемь 32-хразрядных РОН (только для i386 и выше), восемь 16-тиразрядных РОН и восемь восьмиразрядных РОН.

Хотя эти регистры и являются универсальными, но в некоторых командах и методах адресации имеют специальное назначение, что нашло отражение в их названиях.

EAX/AX/AL - аккумулятор (Accumulator). Наиболее часто применяется для хранения промежуточных данных. Многие команды оперируют данными в аккумуляторе несколько быстрее, чем в других регистрах (для первых моделей процессоров). В операциях умножения или деления аккумулятор содержит множимое или делимое до выполнения операции и произведение или частное после операции. В командах ввода - вывода с привлечением портов хранит данные. Вся десятичная арифметика выполняется только с участием аккумулятора AL.

EBX/BX - базовый регистр (Base). Применяется для указания базового (начального) адреса объекта данных в памяти.

ECX/CX/CL - регистр - счетчик (Counter). Участвует в качестве счетчика в некоторых командах, которые производят повторяющиеся операции, например сдвиги или манипуляции цепочками.

EDX/DX - регистр данных (Data). Привлекается для хранения промежуточных данных, а так же в командах умножения и деления совместно с аккумулятором.

ESP/SP - указатель стека (Stack Pointer). Используется при стековых операциях: вызов и возврат из подпрограмм, обработка прерываний и особых случаев.

EBP/BP - указатель базы (Base Pointer). Предназначен для доступа к объектам данных, находящимся в стеке, например параметрам подпрограмм.

ESI/SI - индекс источника (Source Index). Выполняет функцию регистра адреса, в частности, при производстве цепочечных операций адресует текущий элемент цепочки - источника.

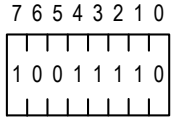
EDI/DI - индекс получателя или приемника (Destination Index). Назначение аналогично предыдущему регистру.

CS - сегментный регистр кода (Code Segment). Содержит машинные команды, образующие собственно программу.

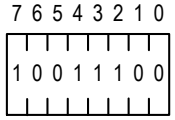
SS - сегментный регистр стека (Stack Segment).

DS - сегментный регистр данных (Data Segment). Определяет главный сегмент данных текущей программы.

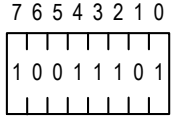
SAHF – запомнить флаги в AH



PUSHF – включить флаги в стек

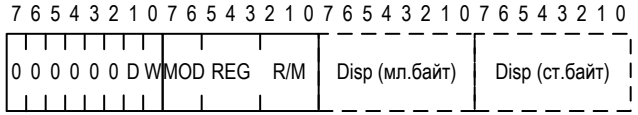


POPF – извлечь флаги из стека

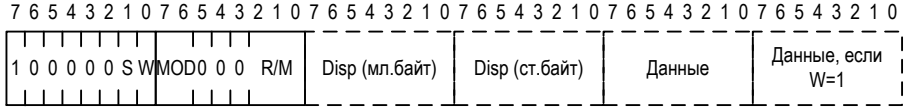


Арифметические команды

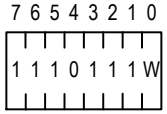
ADD – сложить регистр или память с регистром и запомнить в любом из них



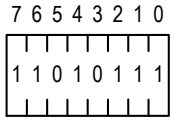
ADD – сложить непосредственный операнд с регистром или памятью



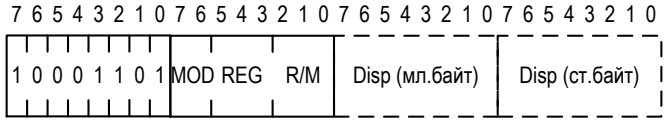
OUT – вывести в переменный порт



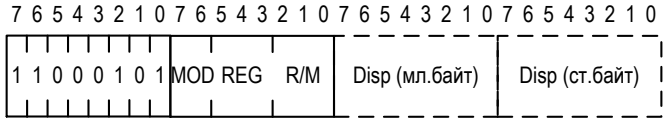
XLAT – преобразовать байт из AL



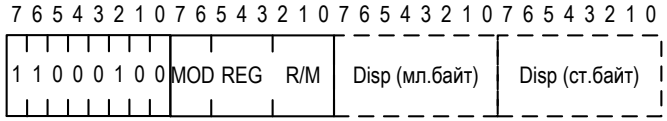
LEA – загрузить EA в регистр



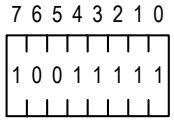
LDS – загрузить указатель в DS



LES – загрузить указатель в ES



LAHF – загрузить флаги в AH



ES, FS, GS - дополнительные сегментные регистры данных.

EIP/IP - программный счетчик команд (Instruction Pointer), содержит смещение следующей исполняемой команды относительно базы сегмента кода.

EFLAGS/FLAGS - регистр флагов. Содержит флаги состояния, сообщающие об особенностях результата после выполнения команды обработки данных, и флаги управления, воздействующие на те или иные аспекты работы процессора.

Язык ассемблер является машинно-ориентированным языком программирования низкого уровня, каждый оператор которого эквивалентен одной машинной команде. К этому добавляются некоторые атрибуты, например текстовые метки, макрокоманды, директивы и другие.

Для составления простейших программ можно использовать следующие директивы Турбо-ассемблера (приставка «Турбо» означает фирму - разработчика Borland).

.MODEL <тип> - модель создания объектного кода.
Например
.MODEL small - здесь используется 16-битная модель и ближние обращения к командам и данным.

.STACK <размер> - установка стека.
Например
.STACK 100h

.DATA
<данные> - модуль данных.

.CODE

<текст программы> - модуль программы.

END - конец текста.

Пример программы с установкой сегмента DS на данные и загрузке в BX адреса начала текста приведен в файле p1.asm:

```
.MODEL small
.STACK 100h
.DATA
label1 DB 'Данными является строка символов'
.CODE

   mov ax,@data
   mov ds,ax
   mov bx,OFFSET label1

END
```

После обработки ассемблером файла с расширением **.asm** получается одноименный объектный модуль с расширением **.obj**, если не было ошибок во время ассемблирования. Просмотреть количество ошибок и предупреждений можно, убрав панели Norton Commander, - выполняя команду Ctrl+O.

Чтобы получить загрузочный модуль, используется компоновщик (линковщик), обрабатывающий файл с расширением .obj. При этом получается загрузочный модуль **.exe**. Если при выполнении компоновки указать в строке

List File [NUL.MAP]:

имя компонуемого файла без расширения, то создастся текстовый **файл сообщений** .map, который можно просмотреть средствами Norton Commander.

Запуск ассемблера осуществляется из командной строки командой

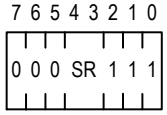
tasm <имя_файла>

Расширение файла здесь и далее указывать не обязательно. Аналогично для компоновщика

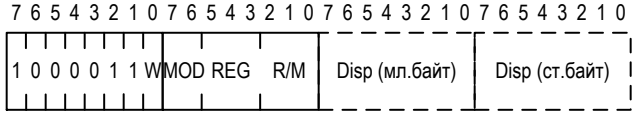
link <имя_файла>

После получения загрузочного модуля можно по шагам просмотреть результат выполнения машинной программы, полученной из текстового файла с программой на языке ассемблер. Для пошагового выполнения файлов с расширением **.exe** используется Турбо-отладчик, который запускается командой

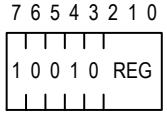
POP – извлечь из стека в сегментный регистр



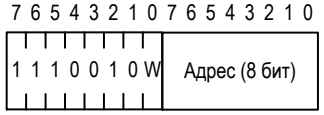
XCHG – обменять регистр или память с регистром



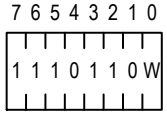
XCHG – обменять регистр с аккумулятором



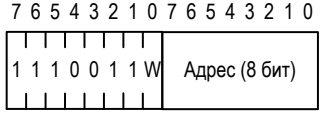
IN – ввести из фиксированного порта



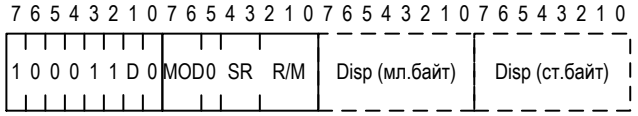
IN – ввести из переменного порта



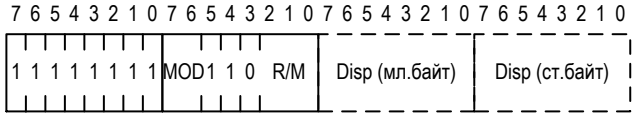
OUT – вывести в фиксированный порт



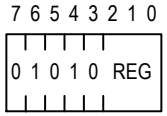
MOV – передать из регистра или памяти в сегментный регистр или наоборот



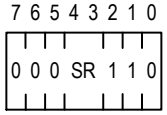
PUSH – включить в стек регистр или память



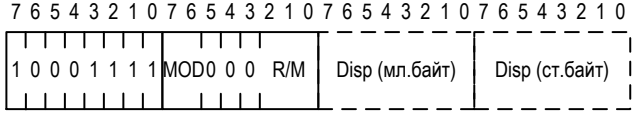
PUSH – включить в стек регистр



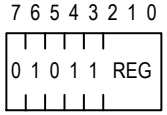
PUSH – включить в стек сегментный регистр



POP – извлечь из стека в регистр или память



POP – извлечь из стека в регистр



td <имя_файла>

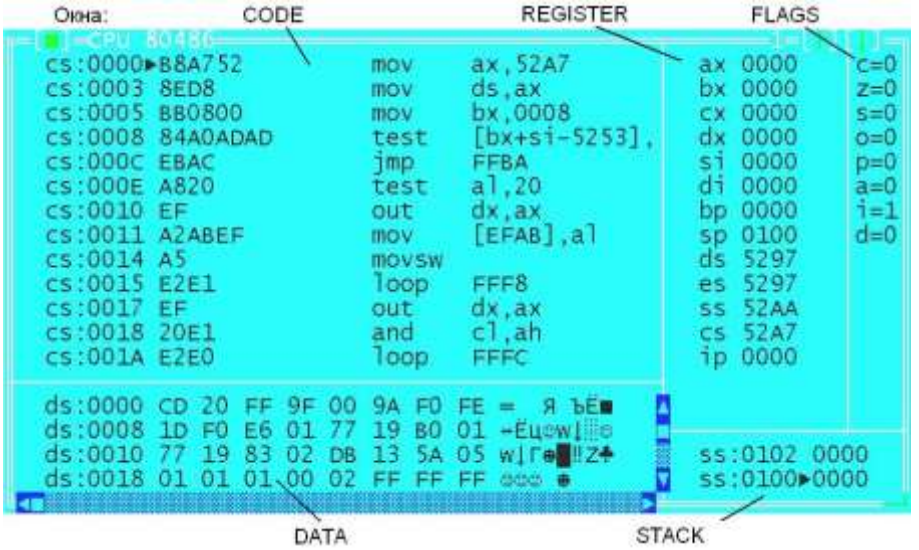


Рис. 6. Окно Турбо-отладчика версии 3.0

После запуска Турбо-отладчика появляется окно CPU в соответствии с рис. 6, состоящее из 5 панелей (по часовой стрелке):

1) CODE - отображается команда и дизассемблированный текст программы. Указатель в виде стрелки показывает текущую точку программы (CS:IP).

2) REGISTER - содержимое регистров процессора.

3) FLAGS - содержимое восьми флагов процессора.

4) STACK - показывается содержимое стека. Указатель установлен на текущей позиции в стеке (SS:SP).

4) DATA - отображается в шестнадцатеричном формате и в кодах ASCII построчный дамп любой выбранной области памяти.

Для перехода с одной панели на другую по часовой стрелке используется клавиша <Tab>, против часовой стрелки - <Shift+Tab>.

В окнах можно изменять содержимое регистров, флаги и данные прямым набором: переводом курсора на значение и заменой символов. Значение флагов меняется клавишей <Enter>.

Каждая панель имеет свое локальное меню, вызываемое клавишами <Alt+F10>. В частности, чтобы изменить сегмент в окне DATA, необходимо перейти в это окно, вызвать меню, выбрать команду Goto и набрать адрес сегмента, хранящегося в соответствующем регистре (CS, DS, ES, SS):

<содержимое_сегментного_регистра>h:<начало_данных>.

В левой части перед двоеточием можно указать просто имя регистра сегмента (что обычно используется при явной адресации).

Для пошагового выполнения команд в окне CODE используется клавиша <F8>. Выход из Турбо-отладчика – клавиши <Alt+X>.

Задание.

1). Просмотреть текстовый файл p1.asm и проанализировать в отчете его содержимое.

2). Отассемблировать файл p1.asm, проверить наличие вновь созданного (по времени) файла p1.obj, просмотреть и записать сообщения, выданные ассемблером.

3). Скомпоновать файл p1.obj с созданием файла сообщений, описать его содержимое. Проверить создание загрузочного файла p1.exe.

4). Войти в Турбо-отладчик командой **td p1**, выполнить в пошаговом режиме три команды, отметить изменение содержимого регистров (выделяются белым цветом).

5). Переместиться в окно REGISTER и изменить содержимое регистров ax и cx, в отчете описать назначение этих регистров. Аналогично изменить флаги P и Z.

6). Установить в окне DATA содержимое сегмента DS с начала данных, адрес которых находится в регистре BX, сравнить текстовое значение данных (справа от кодов) с приведенным в программе.

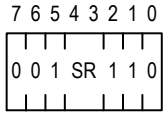
7). Выйти из Турбо - отладчика.

В командах с обращением к регистру флагов, как к 16-битному объекту, регистр обозначается FLAGS, флаги

(FLAGS)=X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

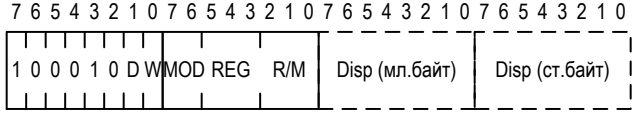
Кодирование команд

Префикс замены сегмента

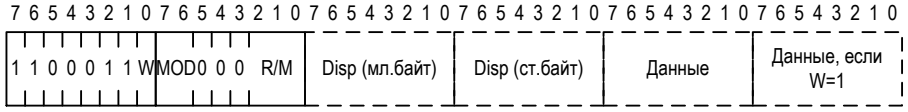


Команды передачи данных

MOV – передать из регистра в память или наоборот



MOV – передать непосредственный операнд в регистр или память



MOV – передать непосредственный операнд в регистр

MOV – передать из аккумулятора в память или наоборот

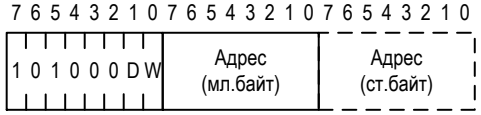


Таблица А1. Значение эффективного адреса.

Поле R/M	Значение EA	Пример операнда со значением поля MOD
000	EA=(BX)+(SI)+Disp	[BX+SI+1FFFH] , MOD=10
001	EA=(BX)+(DI)+ Disp	[BX+DI+80] , MOD=01
010	EA=(BP)+(SI)+ Disp	[BP+SI] , MOD=00
011	EA=(BP)+(DI)+ Disp	[BP+DI+2000H] , MOD=10
100	EA=(SI)+ Disp	[SI+1000] , MOD=10
101	EA=(DI)+ Disp	[DI-8] , MOD=01, Disp=F8
110	EA=(BP)+ Disp (исключение при MOD=00)	55 – константа или метка при MOD=00
111	EA=(BX)+ Disp	[BX-100H],MOD=01,Disp=FF00

Если значение полей в коде операции S:W=01, то операнд образуют 16 бит непосредственных данных. Если S:W=11, то байт непосредственных данных расширяется со знаком для образования 16-битного операнда.

Если V=0, то «счетчик» равен 1; если V=1, то «счетчик» находится в регистре CL.

X – неопределенность.

Z – применяется в цепочных примитивах для сравнения с флажком ZF.

Поле REG интерпретируется соответствии с табл. А2.

Таблица А2. Значения поля REG.

16 бит W=1		8 бит W=0		Сегмент SR	
000	AX	000	AL	00	ES
001	CX	001	CL	01	CS
010	DX	010	DL	10	SS
011	BX	011	BL	11	DS
100	SP	100	AH		
101	BP	101	CH		
110	SI	110	DH		
111	DI	111	BH		

ЛАБОРАТОРНАЯ РАБОТА № 4

Режимы адресации информации

Режимом, или методом адресации называют правило определения адреса и способа доступа к операндам.

Из всех возможных режимов адресации и их комбинаций в процессорах реализуются далеко не все. Рассмотрим режимы адресации микропроцессоров IA-32. В этих процессорах режим 16- или 32-разрядных операндов устанавливается либо вне кода команды, либо указанием расширенного регистра, поэтому здесь и далее упоминаются только восьми- и 16-разрядные данные, подразумевая при этом, что в этих процессорах вместо 16-разрядных данных могут использоваться 32-разрядные при соответствующем режиме адресации.

Непосредственная адресация.

Данное длиной 8 или 16(32) бит является частью команды, то есть хранится вместе с командой в сегменте кода. Из-за этого изменить непосредственный операнд невозможно, и режим используется только в двухадресных командах. Операнд является числом в десятичной, чаще в шестнадцатеричной форме, например

```
mov ax,0f04h     ; загрузить константу в аккумулятор
```

Регистровая адресация.

Данное содержится в определяемом командой внутреннем регистре процессора. 16-битный операнд может находится в регистрах AX, BX, CX, DX, SI, DI, SP или BP, а 8-битный - в регистрах AL, AH, BL, BH, CL, CH, DL, DH. В некоторых командах могут использоваться и другие регистры, например сегментные.

В двухоперандных командах один из операндов, заданных явно, должен использовать регистровую адресацию.

```
inc si           ; увеличение адреса источника
sub cx,cx        ; сброс регистра cx
```

Неявная адресация.

Операнд задается кодом операции. В ассемблерных командах с неявной адресацией поле операнда пустое, например

```
aaa         ; скорректировать AL после сложения
cmc         ; инвертировать флаг переноса
```

Чтобы обеспечить гибкую базовую адресацию и индексирование при работе с ячейками памяти, адрес данных формируют путем сложения содержимого регистров BX или BP, содержимого регистров SI или DI и смещения (offset - в контексте языка ассемблера). В результате получается **эффективный адрес (Effective Address - EA)**. Именно этот 16-битный адрес и вычисляется или указывается в команде.

На втором этапе EA суммируется с базой соответствующего сегмента, умноженной на 16. Получившийся 20-битный линейный адрес является окончательным физическим адресом для i8086/80286. В IA-32, когда страничное преобразование запрещено (R-режим), линейный адрес выдается на шину адреса как физический адрес памяти длиной 32 бита. При разрешенном страничном преобразовании (V-режим) старшие 20 бит линейного адреса преобразуются с помощью таблиц страниц, линейный адрес формируется только внутри процессора, что и объясняет введение нового термина для этих типов процессоров.

Таким образом, указываемый в командах эффективный адрес не является физическим.

При обращении к памяти EA вычисляется с использованием следующих компонентов.

Смещение (Displacement или Disp) – 8-, 16-, или 32-битное число, включенное в команду.

База (Base) – содержимое индексного регистра. Обычно используется для указания на начало некоторого массива.

Индекс (Index) – содержимое индексного регистра. Обычно используется для выбора элемента массива.

Масштаб (Scale) – множитель (1, 2, 4 или 8), указанный в коде команды. Используется для указания размера элемента массива, но только при 32-битной адресации. Здесь не рассматривается.

В общем виде эффективный адрес вычисляется по формуле

ПРИЛОЖЕНИЕ А. МАШИННЫЕ КОДЫ КОМАНД МИКРОПРОЦЕССОРОВ I8086/88

В приложении используются следующие обозначения.
AL – 8-битный аккумулятор;
AX – 16-битный аккумулятор;
CX – регистр счетчик;
DS – сегмент данных;
ES – дополнительный сегмент.

Выше/ ниже относится к беззнаковому значению;
«больше» - более положительное, «меньше» - менее положительное (более отрицательное) знаковое значение.

Если D=1, то «в» регистр (регистр-приемник – первый операнд); если D=0, то «из» регистра (регистр-источник – второй операнд).

Если W=1, команда оперирует словом; если W=0, команда оперирует байтом.

Если **MOD=11**, ТО поле R/M интерпретируется так же, как поле REG.

Если **MOD=00**, то смещение **Disp**=0 (за исключением случая MOD=00 и R/M=110, когда эффективный адрес EA равен старшему и младшему байтам смещения), а младший и старший байты смещения отсутствуют.

Если **MOD=01**, то **Disp** равно младшему байту смещения, расширенному со знаком до 16 бит, а старший байт смещения отсутствует.

Если **MOD=10**, то **Disp** равно старшему и младшему байтам смещения (Low и High).

Disp (1 или 2 байта, начиная с младшего) находится после второго байта команды (перед данными, если они требуются команде).

Эффективный адрес вычисляется в зависимости от поля R/M и смещения Disp, описанного выше, в соответствии с табл. A1.

3. Войдя в турбо - отладчик, перейти к окну CODE (без загрузки программы) и прямым набором ввести команду PUSHF. Затем выполнить ее и из вершины стека списать значение регистра флагов. Пояснить, почему при начальном запуске отладчика некоторые флаги установлены.

4. Некоторые команды условного перехода используют несколько признаков, например JG. Определить комбинации флагов, когда переход осуществляется, а когда нет. Составить программу (см. л.р.4) из одной команды вида

```
L1:  JG L1
```

и в турбо-отладчике проверить эти комбинации, изменяя значения флагов в поле FLAGS. То есть получится таблица из 8 строк для разных комбинаций трех флагов.

Примечание. Если перехода на метку не было, перейти на начало программы можно, установив значение счетчика команд (регистра IP) на адрес этой команды перехода.

5. Составить программу для перекрывающейся пересылки данных. Для этого в сегменте данных определить символьную строку длиной не менее 20 символов, и в программе, используя побайтную пересылку с префиксом повторения, переставить символы с 6 по 15 на место с 11 по 20. Обязательно установить значения сегментных регистров DS, ES одинаковыми.

Проверить работу программы в турбо-отладчике (заметьте, что выполняющийся циклический процесс выполняется как одна простая команда). Изменились ли флаги после выполнения цепочечной команды с префиксом?

6. Модифицировать предыдущую программу для перекрывания по младшим адресам: переставить символы по два (словами) с 6 по 15 на место с 1 по 10. Проверить ее работу.

В отчете по каждому пункту привести пояснения, а для п. 5, 6 и тексты программ.

EA = Base + Index * Scale + Disp

Так как отдельные элементы в этой формуле могут отсутствовать, то возможны и разные режимы адресации при обращении к памяти.

Прямая адресация.

Эффективный адрес данного (смещение) содержится в самой команде. Численное значение адреса в ассемблерных командах заключается в квадратные скобки, например

```
mov al,[2000h]          ; передать байт в аккумулятор
                  ; из ячейки памяти 2000h сегмента DS
inc word ptr [1fh]      ; инкремент слова в памяти
```

Отметим, что во второй команде применяется **явный указатель длины операнда**, так как без него невозможно определить, с данными какой длины должна работать команда.

При программировании на ассемблере в явном виде прямая адресация используется редко, в основном для данных, расположенных по фиксированным адресам, например BIOS. Гораздо удобнее использовать **символическую адресацию с указанием меток**. В этом случае программа более наглядна, кроме того, ассемблер может учитывать типы данных и контролировать правильность их использования. При ассемблировании символическая адресация автоматически преобразуется в прямую.

Более того, некоторые ассемблеры могут воспринимать отдельный адрес без указания сегмента как непосредственную адресацию, или выдают предупреждение, что используется не символическая адресация, а прямая, и в программе возможна ошибка. В этом случае необходимо указывать адрес в соответствии с общепринятым при сегментной организации памяти

<сегмент>:<смещение>

```
mov bh,[ds:50h]
```

В некоторых типах процессоров этот метод адресации называют **абсолютной адресацией** (обращение к указанной физической ячейке), но из-за того, что в рассматриваемых процессорах адрес является не физическим, а эффективным, это название не используется.

Косвенная регистровая адресация.

Эффективный адрес данного находится в базовом регистре BX или в одном из индексных регистров SI или DI. Использование регистра для адресации указывается заключением его в квадратные скобки, например

```
mov al,[bx]           ; пересылка в al байта по адресу,
                      ; хранящемуся в bx
not byte ptr [di]     ; инверсия байта по адресу,
                      ; определяемому DI
```

Во второй команде обязателен явный указатель длины операнда.

Базовая или индексная адресация.

Здесь эффективный адрес равен сумме 8- или 16-битного смещения и содержимого одного из базовых BX, BP или индексных SI, DI регистров. Конкретное название метода зависит от использования тех или других регистров. Смещение может быть как положительным, так и отрицательным, поэтому в указании адреса может быть как знак "+", так и знак "–", например

```
add dx,[bx-2]    ; сложение содержимого dx со словом,
                 ; предыдущим адресуемому регистром bx
mov ch,[si+20h]  ; пересылка в ch байта из памяти
```

В языке ассемблер допускается использовать не только константы, но и переменные, а так же их сочетания. Здесь возможны следующие записи

```
mov ax,variab[bx+10]
mov ax,mesg[bp]
```

Из-за того, что смещение может быть как положительным, так и отрицательным, этот метод так же называют **регистровой относительной адресацией**.

Команда цепочечной пересылки имеет вид

MOVS <приемник>,<источник>

но на самом деле аргументы команды фиктивные, и указывают только на размер операндов – байт или слово, так как данные адресуются регистрами SI и DI. Поэтому при составлении программ в основном используют следующие эквивалентные обозначения этой команды:

MOVSB - для байта,
MOVSW - для слова.

Для IA-32 так же используется команда MOVSD – пересылка двойного слова. В зависимости от типа команды содержимое SI и DI увеличивается или уменьшается в зависимости от флага DF на 1, 2 или 4.

Чтобы облегчить передачу цепочек из одного сегмента в другой, **операнд - получатель**, адресуемый регистром DI (или EDI - для IA-32), всегда находится **в сегменте ES**. Замену сегмента можно использовать **для операнда - источника, по умолчанию это DS**.

Для еще большего сокращения объема программы и, соответственно, увеличения быстродействия, с цепочечными командами можно использовать префикс повторения REP, записываемый непосредственно перед командой и оказывающий воздействие только на нее.

Префикс REP, используя содержимое CX как счетчик, каждый раз уменьшает его на 1, и повторяет команду до тех пор, пока содержимое регистра CX не станет равно нулю. После этого будет выполняться следующая команда.

Задание.

1. В отчете указать по каждому флагу, к какой группе он относится: к флагу состояния или флагу управления с соответствующим комментарием.

2. Пояснить, почему только для арифметического флага переноса есть команды прямого его изменения.

Таблица 8. Команды условных переходов.

Мнемоника	Перейти, если	Состояние флагов
JA/JNBE	Выше/не ниже или равно	CF or ZF=0
JAE/JNB	Выше или равно/не ниже	CF=0
JB/JNAE	Ниже/не выше или равно	CF=1
JBE/JNA	Ниже или равно/не выше	CF or ZF=1
JC	Есть перенос	CF=1
JE/JZ	Равно/нуль	ZF=1
JG/JNLE	Больше/не меньше или равно	(SF xor OF) or ZF=0
JGE/JNL	Больше или равно/не меньше	SF xor OF=0
JL/JNGE	Меньше/не больше или равно	SF xor OF=1
JLE/JNG	Меньше или равно/не больше	(SF xor OF) or ZF=1
JNC	Нет переноса	CF=0
JNE/JNZ	Не равно/не нуль	ZF=0
JNO	Нет переполнения	OF=0
JNP/JPO	Нет паритета/паритет нечетный	PF=0
JNS	Нет знака (положительный)	SF=0
JO	Есть переполнение	OF=1
JP/JPE	Есть паритет/паритет четный	PF=1
JS	Есть знак (отрицательный)	SF=1

Арифметические флаги используются не только в командах условных переходов, но и во многих других. Например, флаг вспомогательного переноса применяется в командах коррекции для арифметических операций при работе с двоично-упакованными десятичными числами, флаг переноса – в командах при использовании многократной точности, во всех командах сдвигов и других.

Для работы с флагом направления существует целый класс команд – цепочечные команды. Простейших цепочечных команд семь, разберем команду пересылки.

Базово-индексная адресация.

Эффективный адрес равен сумме содержимого базового BX или BP и индексного SI или DI регистров. Имена регистров по отдельности или их сумма заключается в квадратные скобки, например

```
sub cx,[bx][di] ; вычитание из содержимого cx
                ; содержимого ячеки памяти по адресу,
                ; равному сумме содержимого регистров
                ; bx и di относительно сегмента DS
or al,[bp+si]   ; логическое "или" байта в AL
                ; с байтом в памяти
```

Относительная базово-индексная адресация.

Эффективный адрес равен сумме смещения, базового BX или BP регистра и индексного SI или DI регистра. Предположим, что в команде

```
mov ax,[bx+1b57h][di]
```

содержимое регистров <BX>=0158h, <DI>=10a5h, и в качестве сегментного используется по умолчанию регистр DS, содержимое которого <DS>=2100h. Тогда

EA=158+10a5+1b57=2d54
Физический адрес=2d54+21000=23d54

В программе на ассемблере при записи операндов можно использовать, например, следующие комбинации

```
lab1[bx+5][si-2]
data[bx][si]
[bp+di+100h]
```

Режимы адресации для адресов перехода включают внутрисегментные: прямой, косвенный и межсегментные: прямой, косвенный. Они работают относительно указателя команды IP.

Данные режимы справедливы только при 16-битной адресации.

В Р-режиме процессоров IA-32 используется 32-битная адресация, являющаяся расширением 16-битной. Эти режимы упрощают разработку программ. В частности, для адресации можно использовать любой регистр общего назначения, а индекс разрешается масштабировать (умножать) на 1, 2, 4 или 8. В табл. 3 приведены методы адресации и примеры простых команд.

Таблица 3. Методы адресации процессоров IA-32.

Адресация	Пример команды
Непосредственная	mov eax,12345678h
Регистровая	mov eax,ecx
Прямая	mov eax,[3456789h]
Регистровая косвенная	mov eax,[ecx]
Базовая (или индексная) со смещением	mov eax,[ecx]+1200h
Базово-индексная со смещением	mov eax,[ecx][edx]+40h
Индексная с масштабированием и смещением	mov eax,[esi*4]+400h
Базово-индексная с масштабированием	mov eax,[edx][ecx*8]
Базово-индексная с масштабированием и смещением	mov eax,[ebx][edi*2]+20h

Задание.

1). Составить следующую программу, расширив исходный текст из предыдущей лабораторной работы. Для этого можно скопировать файл p1.asm в произвольный раздел, затем переименовать с копированием в раздел данного курса. После выполнения трех команд в регистре BX будет находиться адрес начала текста данных.

Команды условных переходов.

При сравнении двух чисел может возникнуть вопрос - является ли число 1111 1111 больше числа 0000 0000? Ответ может быть и положительным и отрицательным. Если числа считать беззнаковыми, то первое число 255 больше 0. Но если числа знаковые, то -1 меньше 0. Поэтому при сравнении знаковых чисел используют термины "меньше" и "больше", а при работе с беззнаковыми числами - "ниже" и "выше".

Следовательно, между двумя числами существуют отношения "**равны", "выше", "ниже", "меньше" и "больше**", и соответственно, противоположные им.

Все команды условных переходов производят передачу управления только в пределах текущего сегмента кода, если заданное в команде условие удовлетворяется. Переход реализуется путем прибавления находящегося в команде смещения к содержимому регистра IP (EIP).

В процессорах i8086/286 8-битное смещение обеспечивает диапазон перехода от -128 до +127 байт. В IA-32 наряду с таким допускается также полное 16- или 32-битное смещение, то есть переход в любую точку текущего сегмента кода.

Многие команды условного перехода имеют две мнемоники, подчеркивающие содержательный смысл проверяемого условия и введенные для удобства программистов (переход, связанный с переносом, имеет даже три мнемоники).

Все команды условных переходов сведены в табл. 8.

Команды операций над флагами.

Таблица 8. Команды операций над флагами.

Мнемо-ника	Описание команды	Действие
CLC	Сбросить флаг переноса	0 → CF
CMC	Инвертировать флаг переноса	1-CF → CF
STC	Установить флаг переноса	1 → CF
CLD	Сбросить флаг направления	0 → DF
STD	Установить флаг направления	1 → DF
CLI	Запретить прерывания	0 → IF
STI	Разрешить прерывания	1 → IF

Команды CLI и STI являются IOPL-чувствительными, то есть выполняющая их программа должна иметь текущий уровень привилегий, меньше или равный содержимому поля IOPL (только для P-режима IA-32).

Команды передачи флагов.

Эта подгруппа содержит несколько простых команд, которые обеспечивают доступ к флагам процессора.

Команда LAHF загрузки флагов в регистр AH копирует младший байт регистра FLAGS (EFLAGS) во второй байт аккумулятора, где флаги можно изменить, проверить или сохранить. Эта команда была оставлена в системе команд для совместимости с i8080, и в программах для последующих моделей процессоров практически не применяется. Иногда ее можно использовать для условных переходов по таким комбинациям флагов, которые не предусмотрены в стандартных командах условных переходов.

Команда SAHF сохранения регистра AH во флагах, обратная предыдущей.

Команды PUSHF и POPF позволяют целиком включать в стек и извлекать из стека содержимое регистра FLAGS. Аналогично для 32-разрядного регистра EFLAGS используются команды PUSHFD и POPFD.

Добавить следующие команды.

а). Учитывая содержимое BX, загрузить в CL 12-й символ данных, а в CH - 16-й символ.

б). Непосредственно загрузить в регистр SI порядковый номер первого пробела из цепочки данных.

в). Увеличить содержимое SI на 1.

г). Используя регистры BX и SI, прочитать в AX первые два символа второго слова.

д). Прочитать в DL последний символ второго слова относительно его начала, хранящегося в SI.

е). Поменять местами в памяти 25-й и 30-й символы.

ж). Увеличить в памяти код второго пробела на 10.

з). Переслать в AX слово с прямым указанием адреса 10.

2). Набрать программу и получить загрузочный модуль. Отметить, что происходит, если в п. з не указать сегмент.

3). Войти в отладчик, проверить установку сегмента данных на панели DATA. Сравнить дизассемблированный текст с исходным.

4). Выполнить программу по шагам, отмечая изменение содержимого регистров.

В отчете привести текст программы с указанием для каждой команды в качестве комментария метода адресации. Для п.4 составить таблицу с указанием изменения содержимого регистров и используемых в пунктах 1.е-1.з байтах сегмента данных.

ЛАБОРАТОРНАЯ РАБОТА № 5

Структура команд процессора

Структура, или формат команд, тесно связана с архитектурой процессора и системой его команд.

С точки зрения структуры командой является упорядоченная совокупность бит информации, представляющая наименование операции, инициируемое командой, и адресов операндов, участвующих в выполнении операции.

Заметим, что если после команды стоит непосредственно данное, то все равно в команде указано, что адресация идет относительно программного счетчика. От максимального количества адресов, указанных в команде, зависит адресности ЭВМ.

В современных высокопроизводительных процессорах общего применения формат команд достаточно сложен. Это особенно относится к процессорам IA-32, для которых требуется полная совместимость по системе команд ранних моделей.

В табл. 4 представлена структура форматов команд этой линии процессоров, причем поля, введенные начиная с i386, отмечены звездочкой.

Таблица 4. Формат команд процессоров IA-32.

Поля команды	Размер в байтах
* Префикс повторения	0 или 1
* Префикс размера адреса	0 или 1
* Префикс размера операнда	0 или 1
Префикс замены сегмента	0 или 1
Код операции	1 или 2*
Описание режима адресации	0 или 1
* Описание адреса	0 или 1
Смещение	0,1,2 или 4*
Данное	0,1,2 или 4*

IOPL (Input/ Output Privilege Level) – уровень привилегий ввода/вывода. Это двухбитное поле является составной частью механизма защиты процессора и действует в защищенном режиме (P-режиме). Оно показывает тот минимальный уровень привилегий выполняющейся задачи, на котором разрешается производить операции ввода/вывода. То есть механизм защиты процессора устанавливает это поле для управления доступом к адресному пространству ввода/вывода.

NT (Nested Task) – вложенная задача. Флаг работает только в многозадачном (защищенном) режиме и автоматически устанавливается, когда с помощью команды CALL происходит переключение на другую задачу, а не обслуживается прерывание. Состояние флага проверяет команда возврата из прерывания IRET и при NT=1 осуществляется переключение задачи, а при NT=0 происходит обычный возврат из прерывания.

RF (Resume Flag) – флаг возобновления. Он является составной частью средств отладки и действует совместно с регистрами отладки. Устанавливая этот флаг, можно селективно маскировать некоторые особые случаи в процессе отладки программы.

VM (Virtual Mode) – флаг режима виртуального процессора i8086 (V -режима). Установка этого флага превращает процессор IA-32 в высокопроизводительный i8086, то есть при этом эмулируется его программная среда. При сброшенном флаге процессор может работать как в режиме реальных адресов (R), так и в защищенном режиме (P).

AC (Alignment Check) – контроль выравнивания. Установка флага разрешает контроль выравнивания при обращении к памяти. Тогда особый случай контроля выравнивания возникает при обращении к слову с нечетным адресом, или к двойному слову, адрес которого не кратен четырем.

VIF (Virtual Interrupt Flag) – виртуальное разрешение прерывания для многозадачных систем.

VIP (Virtual Interrupt Pending) – виртуальный запрос прерывания.

ID (Id Flag) – флаг доступности команды идентификации.

Большинство команд влияет на флаги состояния, но в системе команд предусмотрены инструкции, работающие непосредственно с флагами.

AF (Auxiliary carry Flag) – флаг вспомогательного переноса. Устанавливается, если операция вызвала перенос (например, при сложении) или заем (при вычитании) из младшей тетрады, то есть бита 3, результата. Применяется этот флаг в десятичной арифметике соответствующими командами и явно программно недоступен.

ZF (Zero Flag) – флаг нулевого результата операции. Так же широко используется в операции сравнения: ZF=1, когда сравниваемые операнды равны.

SF (Sign Flag) – флаг знака. Устанавливается в то же состояние, в каком находится старший бит результата (7й, 15й или 31й). Используется при работе с целыми знаковыми числами, представленными в дополнительном коде.

TF (Trap Flag или Trace Flag) – флаг трассировки (пошаговой работы процессора). При TF=0 процессор работает в обычном режиме, при установленном флаге после выполнения каждой команды генерируется внутреннее прерывание (исключение 1). Этот флаг предназначен для отладки программ.

IF (Interrupt Flag) – разрешение прерывания. Если флаг установлен, то процессор распознает и обрабатывает внешние аппаратные маскируемые прерывания по входу INTR. Запрещение же прерываний влияет только на этот вход. Внешние немаскируемые прерывания со входа NMI и внутренние исключения распознаются всегда.

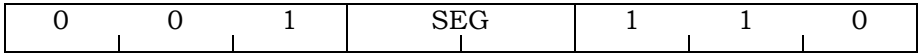
DF (Direction Flag) – флаг направления. Задает направление обработки цепочек данных цепочечными командами. При установленном флаге производится автоматический декремент индексных регистров SI и DI, адресующих цепочки, то есть обработка производится справа налево или назад (от старших адресов к младшим). При сброшенном флаге - автоинкремент, то есть обработка вперед или слева направо. Величина изменения регистров составляет 1, 2 или 4 в зависимости от размера элементов цепочек.

OF (Overflow Flag) – переполнение. Устанавливается, если выполнение операции привело к переносу (заему) в знаковый, или старший бит результата, но не привело к переносу за разрядную сетку, или наоборот. То есть значение переноса (заема) в старший бит и из него не совпадают. Используется в арифметике целых знаковых чисел.

Таким образом, если в i8086/286 длина команды может колебаться в пределах 1 - 7 байт, то для более старших моделей этот предел больше: от 1 до 16 байт.

В этой работе рассмотрим только поля, используемые в 16-разрядных процессорах i8086/286.

Префиксы, по сути, являются отдельными командами, но не обрабатывающими данные, а влияющими на выполнение последующей команды. По умолчанию при обработке данных используется сегмент DS, а в командах работы со стеком сегмент SS. При использовании префикса можно работать с любым сегментом. Формат префикса замены сегмента следующий:



То есть зарезервированы начальные коды команд:
2EH - префикс замены сегмента CS;
36H - префикс замены сегмента SS;
3EH - префикс замены сегмента DS;
26H - префикс замены сегмента ES.

Обычно код операции занимает первый байт команды, но в некоторых командах здесь же указывается регистр, а в других три бита кода операции находятся во втором байте. Большинство кодов операций имеют следующую структуру.

В коде операции имеется бит W, если команда может оперировать как с байтом (W=0), так и со словом (W=1).

В двухоперандных командах, кроме команд с непосредственным операндом, одним из операндов должен быть РОН (обязательно использование регистровой адресации). В таких командах бит D показывает, чем является регистр: источником (D=0) или получателем (D=1).

Если же в операции участвует непосредственный операнд, представленный в дополнительном коде, то совместно с битом W используется бит S. В этом случае можно сэкономить память, используя короткий 8-разрядный операнд в 16/32-разрядной операции. При S:W=11 загружаемая константа будет автоматически расширена со знаком.

В IA-32 двухбайтные коды операций введены для новых команд, у которых первый байт кода операции содержит значение 0FH.

Байт описания режима адресации имеет следующую структуру.

MOD	REG/КОП	R/M

Второе поле является переменным и может содержать либо номер РОН, либо расширение кода операции первого байта. Код операции записывается для однооперандных команд или двухоперандных, в которых один из операндов неявно определяется кодом операции. При использовании регистра, он принимает такое же значение, как и для поля R/M при MOD=11 в табл. 5.

Операнд, указываемый полями MOD и R/M, и сегмент, принимаемый по умолчанию, определяются в соответствии с табл. 5.

Таблица 5. Режимы адресации и сегментные регистры по умолчанию для различных комбинаций полей MOD и R/M.

R/M	MOD				
	00	01	10	11	
				W=0	W=1
000 Сегмент	(BX)+(SI) DS	(BX)+(SI)+D8 DS	(BX)+(SI)+D16 DS	AL	AX
001 Сегмент	(BX)+(DI) DS	(BX)+(DI)+D8 DS	(BX)+(DI)+D16 DS	CL	CX
010 Сегмент	(BP)+(SI) SS	(BP)+(SI)+D8 SS	(BP)+(SI)+D16 SS	DL	DX
011 Сегмент	(BP)+(DI) SS	(BP)+(DI)+D8 SS	(BP)+(SI)+D16 SS	BL	BX
100 Сегмент	(SI) DS	(SI)+D8 DS	(SI)+D16 DS	AH	SP
101 Сегмент	(DI) DS	(DI)+D8 DS	(DI)+D16 DS	CH	BP
110 Сегмент	D16 DS	(BP)+D8 SS	(BP)+D16 SS	DH	SI
111 Сегмент	(BX) DS	(BX)+D8 DS	(BX)+D16 DS	BH	DI

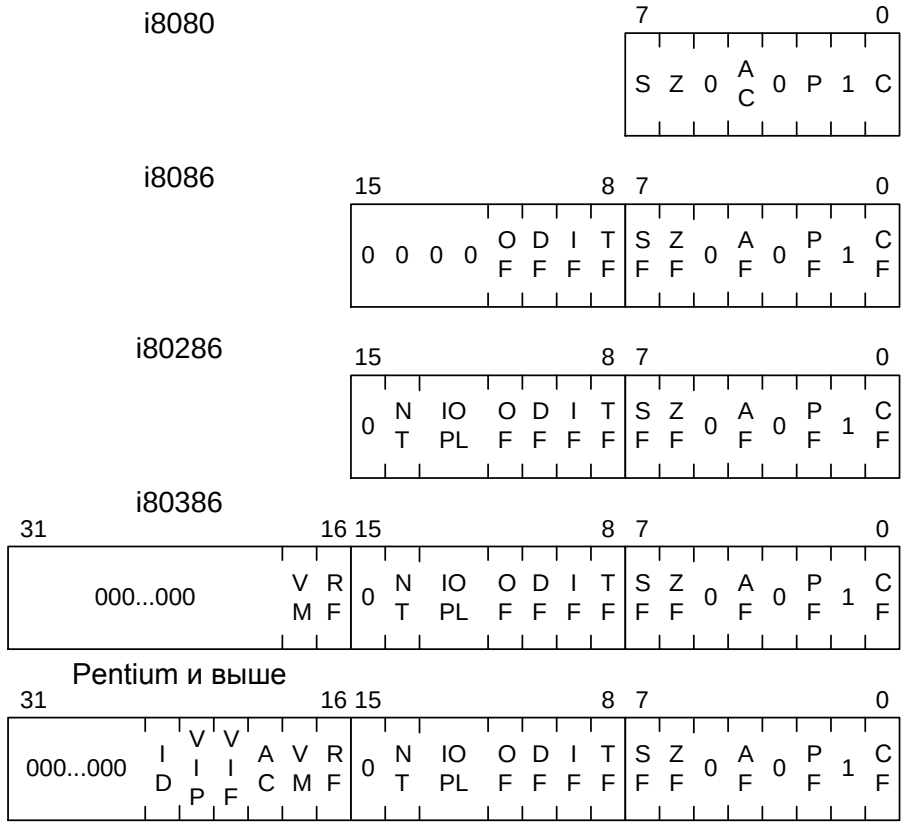


Рис. 7. Эволюция регистра флагов в процессорах фирмы Intel.

Опишем флаги, используемые в микропроцессорах IA-32.

CF (Carry Flag) – флаг переноса. Устанавливается, если результат не помещается в разрядную сетку и выходит в следующий за старшим битом разряд, например, переполнение при сложении или заем при вычитании. В зависимости от размера операндов старшим битом может быть 7-й, 15-й или 31-й.

PF (Parity Flag) – флаг четности (паритета). Устанавливается, если младшие восемь бит результата содержат четное число единичных бит. То есть на этот флаг оказывает влияние только младший байт результата.

Таким образом, если MOD=11, то данное находится в регистре, размер которого определяется битом W кода операции, иначе данное находится в памяти. MOD=01 указывает на то, что в команде присутствует 8-битное смещение, добавляемое к базовому или индексному регистру, а MOD=10 – 16-битное. При MOD=00 смещение отсутствует, за исключением случая, когда R/M=110, то есть используется прямая адресация.

Пример. Задано машинное представление команды (в шестнадцатеричной системе счисления) **836F1E19**, определить ее мнемоническое обозначение.

Первый байт $83_{16}=10000011_2$ является либо префиксом, либо кодом операции. В соответствии с Приложением А определяем, что команд с кодом 100000SW достаточно много, и необходимо учитывать поле КОП второго байта $6F_{16}=01101111_2$. Получаем, что поле КОП=101, это команда вычитания непосредственного операнда из регистра или памяти, то есть мнемоническое обозначение действия – SUB.

По первому байту определяем, что используется короткий операнд (S=1), размер операндов – слово (W=1).

По полям MOD=01 и R/M=111 определяем режим адресации первого операнда (второй – это константа, данное последнего байта – 19): (BX)+D8. То есть используется базовая адресация с использованием регистра BX и 8-разрядным смещением, равным 1E, обозначаемая как [BX+1E]. Но, если ячейка памяти указывается косвенно (имя регистра, по которому можно определить размеры операнда, в команде отсутствует), то размер операндов должен быть задан явным указателем размера – word ptr.

Обратите внимание, что при двухбайтных смещении или данных в коде сначала указывается младший байт, затем старший.

В итоге получаем полное мнемоническое обозначение команды:

```
SUB  word ptr [bx+1E],0019
```

Данные рассуждения можно свести в табл. 6.

ЛАБОРАТОРНАЯ РАБОТА № 6

Регистр флагов процессора

При выполнении различных команд в микропроцессорах вырабатываются определенные признаки, указывающие на результат выполнения. Эти признаки, такие, как равенство результата нулю, переполнение и другие, хранятся в специальном регистре и используются для анализа результатов, - в основном для условных переходов. Так как эти признаки, хранящиеся в отдельных битах этого регистра, не связаны между собой, то их называют флагами или флажками.

Иногда этот регистр называют PSW (Processor Status Word), отражающий его назначение. Первоначально в i8080 существовали две команды работы со стеком POP PSW и PUSH PSW, объединяющие два регистра - регистр флагов и аккумулятор. Но в дальнейшем, начиная с i8086, произошло объединение регистровых пар в отдельные регистры, и полное разделение аккумулятора и флагов. Поэтому, начиная с i80286, данный регистр называют регистром флагов процессора или регистром системных флагов.

Переходя от одного поколения к следующему поколению процессоров, этот регистр хоть и претерпевает значительные изменения, но сохраняет полную преемственность, что видно из рис. 7.

Все флаги подразделяются на условные (флаги состояния) и управляющие (флаги управления). Последние появились, начиная с модели i8086. Флаги состояния сообщают об особенностях результата команды обработки данных. Для их изменения прямых команд нет, кроме работы с флагом переноса. Флаги управления воздействуют на те или иные аспекты работы процессора и устанавливаются или сбрасываются программно.

Таблица 6. Структура команды на машинном уровне.

Код	Мнемоника	Бит W	Бит D/S	MOD R/M	REG/ КОП	Смещение, данные
836F1E19	SUB word ptr [bx+1E],0019	1 Слово	S=1 Короткий операнд	01 111 (bx)+D8	КОП= 101	1E-см. 19-данные

Задание.

1). Для программы, составленной в предыдущей работе, используя код и мнемоническое обозначение команды из отладчика, составить таблицу для команд с третьей по седьмую по образцу табл. 6.

2). В соответствии с вариантом задания записать по приведенным кодам мнемонические обозначения двух бинарных и одной унарной команд. Пояснить в побитном представлении поля этих команд по образцу табл. 6.

3). Проверить правильность составленных ассемблерных команд, включив их в программу и используя отладчик.

Таблица 7. Варианты заданий к лабораторной работе №5.

№ вар.	Название команды	Код команды
1.	Сложение с переносом	81 57 05 FF 01
	Вычитание с заемом	19 50 05
	Декремент	FE 0D
2.	Логическое объединяющее ИЛИ	08 A6 00 02
	Логическое исключающее ИЛИ	32 A0 50 02
	Изменение знака	F6 5F 0A
3.	Вычитание с заемом	83 1C 64
	Сложение	00 88 00 01
	Инверсия	F6 95 00 10
4.	Логическое И	83 67 0A 14
	Вычитание	29 48 04
	Инкремент	FE C4

Продолжение табл. 7.

№ вар.	Название команды	Код команды
5.	Проверка	85 4F 02
	Логическое объединяющее ИЛИ	81 8C 00 10 88 22
	Декремент	FE 4C 01
6.	Сравнение	81 7E 05 00 20
	Сложение с переносом	10 BC 00 01
	Инверсия	F6 D2
7.	Вычитание с заемом	1A 47 50
	Сложение по модулю 2	81 31 34 12
	Умножение без знака	F7 60 10
8.	Обмен	87 80 8A 00
	Сложение	81 44 02 00 10
	Изменение знака	F6 1C
9.	Логическое объединение по И	81 25 99 66
	Сложение с переносом	13 97 00 04
	Извлечение из стека	8F 01
10.	Сложение по модулю 2	81 70 55 33 22
	Вычитание	2A 65 F0
	Инверсия	F6 15
11.	Вычитание с заемом	81 AC 00 04 34 12
	Логическое объединяющее ИЛИ	09 17
	Инкремент	FE 85 00 80
12.	Сложение с переносом	81 11 60 20
	Сложение по модулю 2	32 8F 00 24
	Декремент	FF 0C
13.	Логическое объединение по И	80 61 04 44
	Сложение	03 47 02
	Включение в стек	FF 30
14.	Логическое объединяющее ИЛИ	81 8F 00 02 00 04
	Вычитание	29 55 08
	Умножение со знаком	F6 28
15.	Сложение	80 80 00 12 88
	Логическое объединение по И	09 07
	Извлечение из стека	8F 44 FC