И. А. Шалатонин

Микропроцессоры и аппаратные средства вычислительной техники

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ЧАСТЬ 1

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ РАДИОФИЗИКИ И ЭЛЕКТРОНИКИ
Кафедра кибернетики

И. А. Шалатонин

Методические указания к лабораторному практикуму по курсам «Архитектура микропроцессоров и вычислительных систем», «Микропроцессоры и аппаратные средства вычислительной техники»

Для студентов специальностей:
G 31 04 02 «Радиофизика»,
G 31 04 03 «Физическая электроника»,
«Компьютерная безопасность»

МИНСК
БГУ
2006

УДК 536.3(075.83)
ББК 22.34р30-252.43
И85

Р е ц е н з е н т ы
доцент кафедры интеллектуальных систем *К. М. Шестаков,*
доктор физико-математических наук, профессор *В. А. Саечников.*

Утверждено Ученым советом
факультета радиофизики и электроники
2005 г., протокол № 2

Шалатонин И. А.

Методические указания к лабораторному практикуму по курсам «Архитектура микропроцессоров и вычислительных систем», «Микропроцессоры и аппаратные средства ВТ», предназначены для студентов 2–4-го курсов факультета радиофизики и электроники.

Мн.: БГУ, 2006. – с.

И85

УДК 536.3(075.83)
ББК 22.34р30-252.43

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ ИЗ ТЕОРИИ

1.1. Понятие о реальном, защищенном и режиме системного управления процессоров фирмы Intel

Тысячи прикладных программ, написанных в 1980-е гг., предназначались для DOS и для процессора 8086. Именно поэтому все процессоры Intel поддерживали режим работы этого процессора. Процессоры Intel до 286 могли работать только в этом однозадачном реальном режиме. Начиная с процессора 286, введена возможность многозадачного режима работы, основная проблема которого – защита данных каждой задачи от остальных данных. Такой режим будем называть защищенным. Следует отметить, что процессор 286 с этой задачей не справился и являлся, по существу, более быстрой версией 8086. Только на процессорах Intel 80386 и выше, с введением новых регистров и повышением разрядности старых регистров удалось достигнуть уверенной работы в защищенном режиме.

В табл. 1 перечислены режимы работы процессоров, совместимых с Intel 80386.

таблица 1

Режимы работы процессоров

Режимы процессора Intel 80386 и выше
R-mode. Реальный (Real) режим. Точная копия единственного режима, в котором работали процессоры 8086 и 8088, адресуя 1 Мбайт оперативной памяти
P-mode. Защищенный (protected) режим. «Родной» режим процессоров 80386 и выше, наиболее полно использующий преимущества его 32-разрядной архитектуры и отличающийся передовыми методами защиты программ, 32-разрядной («плоской», в отличие от l6-разрядной, сегментной) адресацией до 4 Гбайт памяти и возможностью подкачки страниц памяти по требованию V-mode. Режим виртуального процессора 8086, или V86. Это режим является особым состоянием задачи защищенного режима, в котором процессор функционирует как 8086. При этом процессор 80×86 полностью эмулирует работу процессора 8086, не переходя в реальный режим
SMM –mode. Режим системного управления (System Management Mode) Современные модели 32-разрядных процессоров (начиная с некоторых модификаций 386-го и 486-го), кроме обычных режимов – реального, защищенного и режима V86, - имеют дополнительный режим системного управления SMM (System Management Mode). Этот режим предназначен для выполнения некоторых действий с возможностью их полной изоляции от прикладного программного обеспечения и даже операционный системы. Главным образом, этот режим предназначен для реализации системы управления энергопотреблением.

Мы, в основном, рассмотрим практическое программирование для реального режима и лишь наметим основные идеи, заложенные в осуществлении защищенного режима. Важно отметить, что реальный режим работы сохраняется для всех высших моделей процессоров.

Процессоры, совместимые с Intel 80386, поддерживают четыре уровня, или кольца, привилегированности, обеспечивающие различные степени защиты и привилегий для исполняемых программ. В каждый момент времени процессор может работать только на одном уровне привилегий.

При любой передаче управления с одного уровня на другой процессор выполняет десятки операций. Например, смена уровня привилегий на 486-м процессоре занимала 69 тактов. Однако применение различных уровней привилегий необходимо современной операционной системе, чтобы обеспечить защиту жизненно важных участков кода системы.

Для выполнения кода системного уровня операционные системы Windows 95/98 (и Windows NT) используют нулевое кольцо защиты процессоров Intel. Программы, выполняемые в нулевом кольце, могут все. Они работают с аппаратурой напрямую, ничто в системе от них не укроется, и ни одно постороннее приложение не может помешать выполнению кода с «нулевым допуском». Сервисом нулевого уровня защиты процессора 80386 пользуются

файловая система защищенного режима,

диспетчер виртуальных машин

и аппаратно-ориентированные драйверы, в том числе написанные независимыми разработчиками и производителями аппаратного обеспечения.

Разумеется, компоненты операционной системы, предназначенные для работы в нулевом кольце, должны быть тщательно проверены и отлажены, поскольку ошибка в таком драйвере может запросто вызвать крах всей системы.

Прикладные программы и многие части операционной системы выполняются только в третьем кольце процессора 80386. Соответственно они не имеют таких прав, как программы нулевого уровня, и не могут напрямую работать с устройствами компьютера, им приходится обращаться к драйверам устройств, выполняющимся в нулевом кольце. Зато они относительно безопасны для системы в целом.

Таким образом, в Windows 95/98 и Windows NT реализована двухуровневая модель защиты (рис. 1), которую многие называют моделью «ядро-пользователь», а некоторые — даже моделью «клиент-сервер». Оптимальное разделение всех выполняемых программ на системные с высоким уровнем

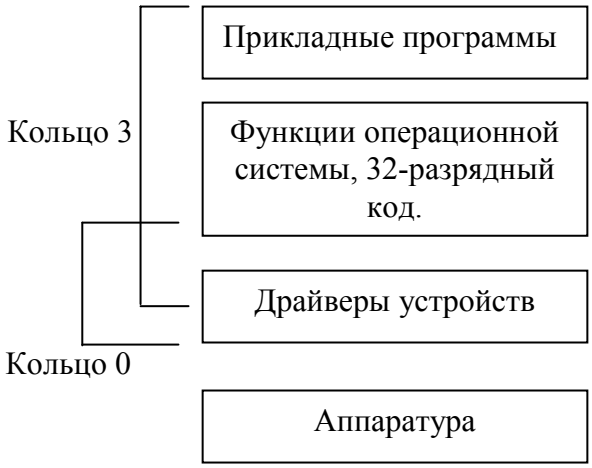


Рис. 1. Двухуровневая модель защиты (модель «ядро-пользователь», модель «клиент-сервер»)

привилегий и прикладные с низким уровнем привилегий обеспечивает достаточно высокую степень защиты без заметного ущерба для общей производительности системы. Именно поэтому Microsoft отказалась от использования первого и второго колец защиты в своих операционных системах, во всяком случае — пока. Microsoft считает, что применение других уровней привилегий заметно снизит производительность системы.

1.2. Реальный режим.

1.2.1. Параметры базового микропроцессора семейства Intel 80×86

Структурная схема микропроцессора **Intel 8086** приведена на рис. 2. Назначение выводов МП приведено на рис. 3.

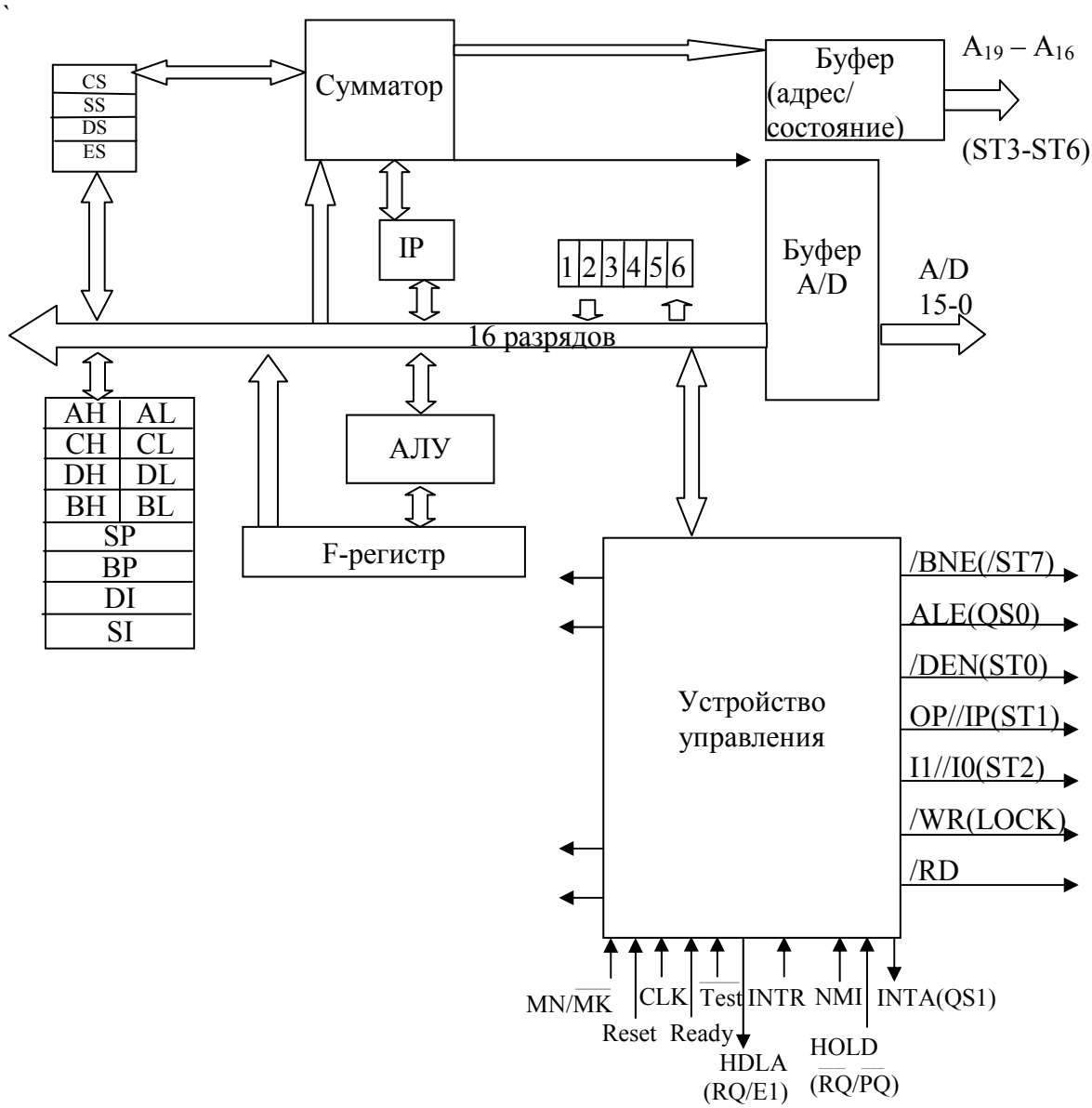


Рис. 2. Структурная схема микропроцессора I8086

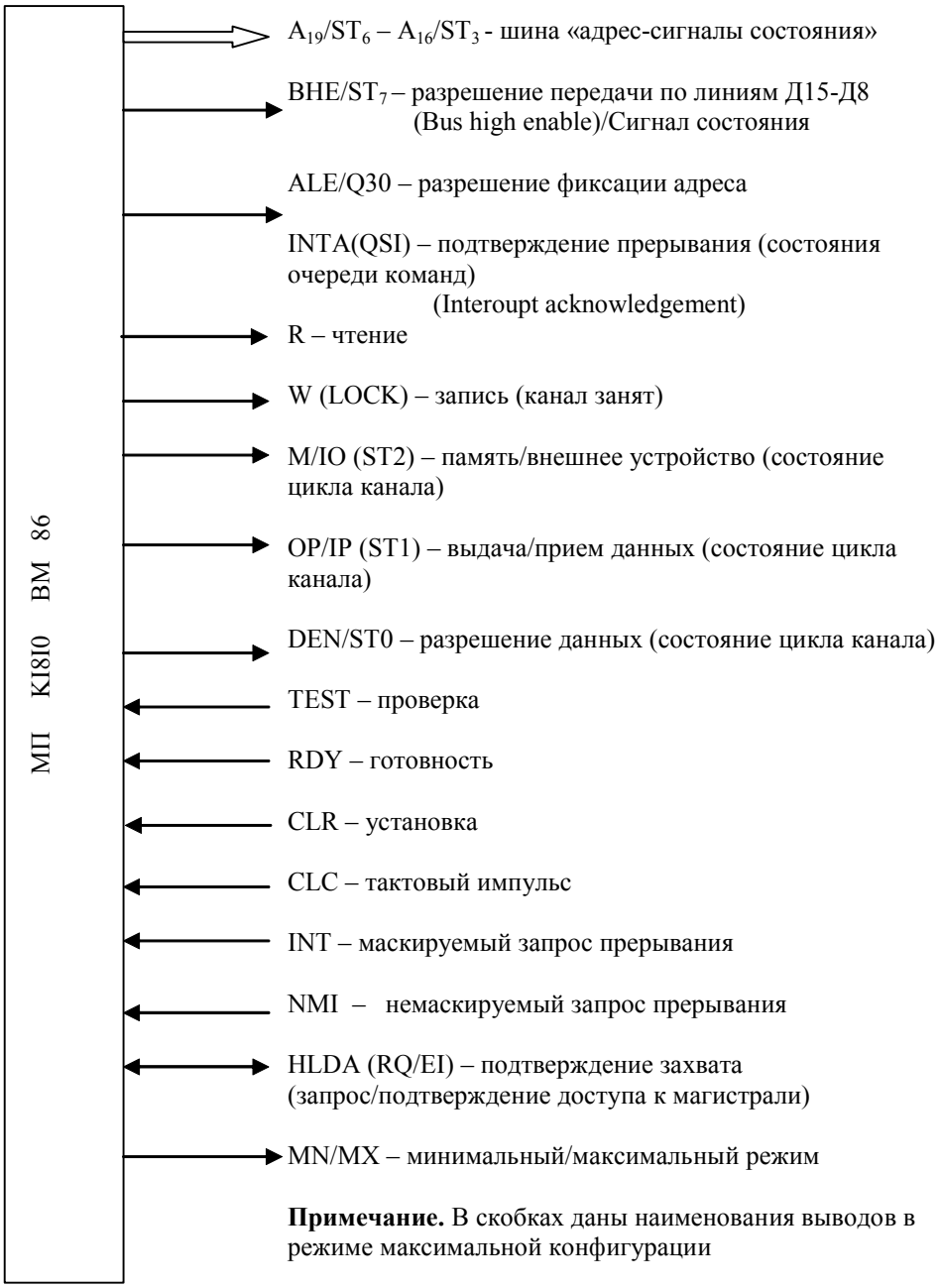


Рис. 3 . Назначение внешних выводов микропроцессора

1.2.2. Функционирование микропроцессора I8086

Временные диаграммы, поясняющие функционирование МП, приведены на рис. 4.

Каждый МЦ состоит из четырех ТИ. В режиме аппаратного ожидания между тактами Т3, Т4 выполняются такты ожидания TW, число которых определяется длительностью внешнего сигнала на входе READY.

В такт T1 на выходы BHE/ST7 A19 – A16, A15 – A0 выдается адрес ЗУ или ВУ и сигнал разделения ЗУ/ВУ на выход M/IO. Адрес сопровождается стробом на выход ALE(STB).

В такте T2 на линии BHE/ST7, A19 – A19/ST6 – ST3 выдается состояние МП, а шина данных переходит при чтении в состояние высокого входного сопротивления, а при записи выдает данные.

В такте T3 данные принимаются по сигналу чтения на выходе RD или выдаются на запись, сопровождаемые сигналом WR.

В такте T4 обмен заканчивается.

В режим ожидания (такты TW) МП переходит, если RDY = 0 (оно должно появиться до начала T3). Состояние выводов шин в этом случае сохраняется.

В режиме программного ожидания МП входит по команде WAIT при значении TEST = 1, выполняя при этом холостые такты. Выходит из этого состояния при появлении на входе TEST = 0. Он может быть не менее пяти ТИ.

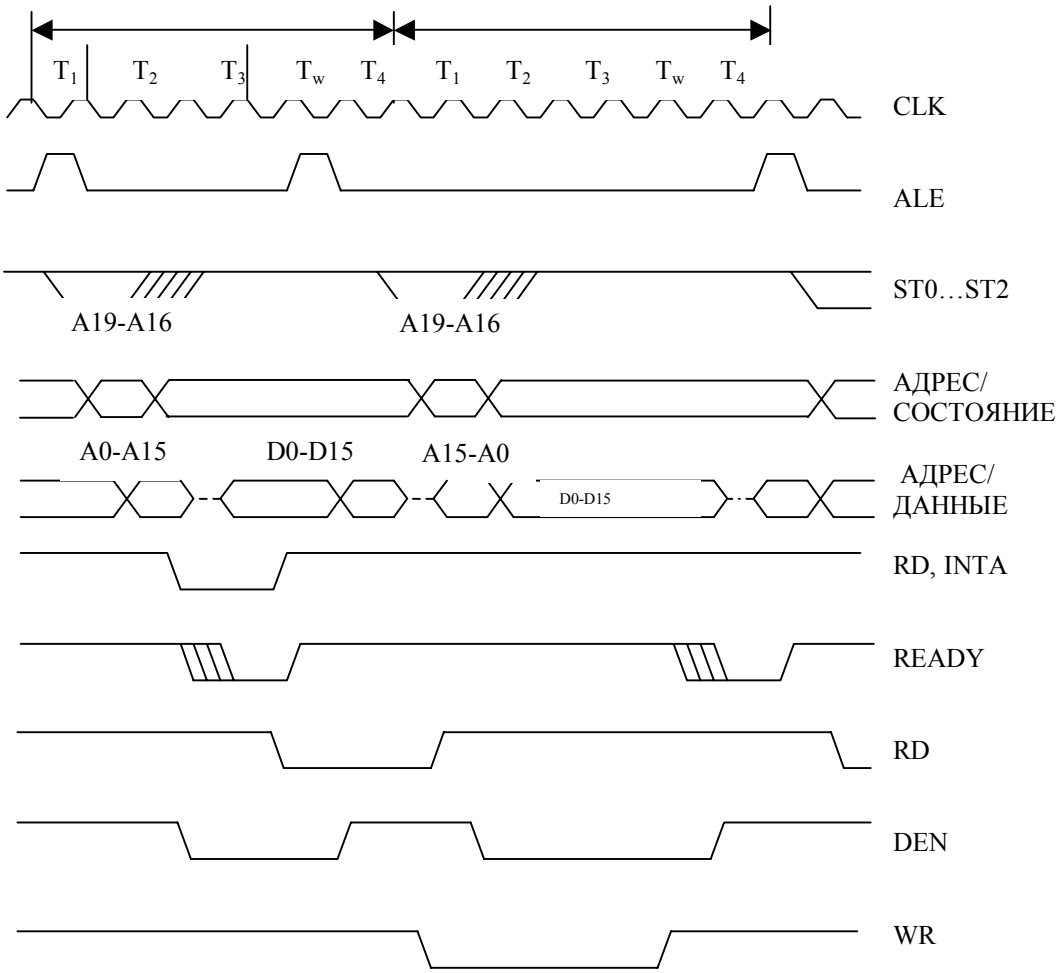


Рис. 4. Временные диаграммы основных сигналов микропроцессора КМ1

1.2.3. Минимальный и максимальный режимы работы

В минимальном режиме МП сам вырабатывает все сигналы управления микропроцессорной системой. Максимальный режим рассчитан на более сложные системы. В этом случае МП вырабатывает сигналы для управления специальным контроллером, который в свою очередь вырабатывает сигналы для управления системой. Выбор режима осуществляется подачей.

1.2.4. Программная модель микропроцессора I8086

Микропроцессор для работы в реальном режиме имеет четырнадцать 16-разрядных регистров (табл. 2). Они делятся на две группы и два отдельных регистра: **группа регистров общего назначения (РОН)** – восемь регистров, **группа сегментных регистров** – четыре регистра, регистр флагов и регистр указателя инструкций. Название группы «регистры общего назначения» не должно вводить в заблуждение – регистры в ней не равноправны, каждый имеет свое специфическое назначение и может использоваться в посторонних целях только тогда, когда он свободен от выполнения своих «прямых обязанностей». Иногда группу регистров общего назначения подразделяют на регистры данных (AX, BX, CX, DX) и регистры указателей (SI, DI, BP, SP). Иногда регистры BX, SI, DI и BP называют регистрами-модификаторами.

таблица 2

Программная модель МП Intel 8086 (КМ1810ВМ86)

Имя	English	Основное назначение	Прим.
регистры общего назначения			
AX	Accumulator	Основной сумматор	AH, AL
BX	Base	Адресация по базе	BH, BL
CX	Counter	Счетчик циклов	CH, CL
DX	Data	Для «длинных» данных	DH, DL
SI	Source Index	Индексирование источника	
DI	Destination Index	Индексирование приемника	
BP	Base Pointer	База стека	
SP	Stack Pointer	Вершина стека	
сегментные регистры			
CS	Code Segment	Сегмент команд не может быть изменен напрямую	
SS	Stack Segment	Сегмент стека	
DS	Data Segment	Сегмент данных	
ES	Extension Segment	Дополнительный сегмент	
регистр флагов			
FLAGS	Flags	Информация о текущем состоянии процессора	
регистр указателя команд			
IP	Instruction Pointer	Задает смещение команды (программно недоступен)	

К первым четырем регистрам общего назначения можно обращаться как целиком (например, AX), так и к их старшим (AH) или младшим (AL) байтам. Регистр SP используется исключительно для указания на вершину стека. Значения регистров CS и IP не могут быть изменены непосредственно в программе (они изменяются либо автоматически, либо косвенно за счет команд переходов). Имя регистра IP вообще не является зарезервированным и не может встречаться в программе (в смысле имени регистра).

Кроме того, РОНы в ряде команд используются для выполнения определенных функций (табл. 3). Такая специализация РОН усложняет программирование, зато позволяет кодировать команды с применением меньшего числа разрядов, чем это было бы необходимо при адресации действительно регистров общего назначения.

1.2.5. Типичная схема адресного пространства I8086

В табл. 5 показана типичная схема адресного пространства 8086 при работе под управлением MS DOS (адреса возрастают сверху вниз).

Биты регистра флагов перечислены в таблице 4.

Первые 640 К памяти с адресами с 00000h по 9FFFFh называются обычной памятью (conventional memory). В ней последовательно располагаются векторы прерываний (256 векторов по 4 байта), содержащие ссылки на подпрограммы обработки прерываний, область данных BIOS (базовой системы ввода/вывода), включающую в себя, в частности, буфер клавиатуры и адреса портов. Далее располагается сама операционная система MS DOS, занимающая около 80 Кбайт. От конца MS DOS до адреса 0A0000h располагается транзитная область памяти, в которой пользователь размещает необходимые ему резидентные программы и запускает свои программы. Размер транзитной области равен 640 Кбайт минус все перечисленное выше,

таблица 3

Функции регистров общего назначения микропроцессора К1810

Регистр	Назначение	Выполняемая функция
AX	Аккумулятор	Ввод-вывод, умножение и деление слов
AL	Аккумулятор (младший байт)	Умножение, деление и ввод-вывод байтов; преобразование; десятичная арифметика
AH	Аккумулятор (старший байт)	Умножение и деление байтов
BX	Базовый регистр	Базовый регистр; преобразование адресов
CX	Счетчик	Обработка строк; подсчет циклов
CL	Счетчик (младший байт)	Динамические и циклические сдвиги
DX	Регистр данных	Умножение и деление слов; косвенный ввод-вывод

SP	Указатель стека	Операции с использованием стека
BP	Указатель базы	Базовый регистр
SI	Указатель источника	Указатель исходной строки; индексный регистр
DI	Указатель приемника	Указатель строки результата; индексный регистр

таблица 4

Биты регистра флагов

Бит	Имя	Название	Тип	Назначение
0	CF	Carry Flag	состояние	Перенос или заем
2	PF	Parity Flag	состояние	Четность
4	AF	Auxiliary Flag	состояние	Перенос или заем BDS
6	ZF	Zero Flag	состояние	Ноль результата
7	SF	Sign Flag	состояние	Знак результата
8	TF	Trace Flag	управление	Трассировка
9	IF	Interrupt Flag	управление	Разрешение прерываний
10	DF	Direction Flag	управление	Направление обработки цепочек
11	OF	Overflow Flag	состояние	Переполнение

таблица 5

Типичная схема адресного пространства I80x86 при работе под управление MS DOS

Адрес	Область памяти	Объём	Класс памяти
00000	Векторы прерываний	1 К	Обычная память (640 К) Conventional memory
00400h	Данные BIOS	256 б	
00500h	MS DOS	ок. 80 К	
	Транзитная область	ок. 550 К	
0A0000	Графический видеобуфер	64 К	Верхняя память (384 К) Upper memory (UM)
0B0000	Транзитная область	32 К	
B8000	Текстовый видеобуфер	32 К	
0C0000	Транзитная область	192 К	
0F0000	BIOS	64 К	
100000	Верхняя память High Memory Area	64 К	HMA
10FFF0	Расширенная память Extended/Expanded Memory Specification	до 4 Г	XMS/EMS

т. е. около 500-600 Кбайт. Это вся память, которой может распоряжаться пользователь по своему усмотрению, не принимая специальных мер по расширению памяти даже в том случае, когда в компьютере есть свободные физические блоки памяти.

Отсюда следует, что каждый килобайт обыкновенной памяти был крайне ценен, и существуют специальные методы экономии этой памяти. В частности, в верхней памяти (UM), простирающейся от адреса 0A0000h до 100000h также имеются транзитные области – так называемые блоки верхней памяти (UMB). Для работы с этими свободными адресами необходимо, во-первых, иметь физические блоки свободной памяти, и, во-вторых, загрузить менеджер верхней памяти EMM386 или его аналоги (QEMM). Для работы с блоком HMA необходимо загрузить менеджер верхней памяти HIMEM. В этих блоках можно располагать необходимые пользователю резидентные программы (например, русификатор, драйвер мыши, кэш-буфер дисков SMARTDRV). Для их загрузки в UMB в операционной системе для файлов CONFIG.SYS и AUTOEXEC.BAT предусмотрены специальные директивы DEVICEHIGH и LOADHIGH. Можно также загрузить в верхнюю память значительную часть MS DOS, доведя размер транзитной области обыкновенной памяти до 620 Кбайт.

1.2.6. Организация ввода-вывода

Допустим непосредственный доступ к первым 0-FF (256) портам. При адресации портов с диапазоном адресов выше FF (256) используется косвенная адресация с помощью занесения адреса порта (0-FFFF) в регистр DX.

Например, in 20h

```
mov    dx,0ffffh
mov    al,9bh
out    dx,al
```

Подобно ячейкам памяти любой порт может быть 8- и 16-битовым.

Как и 8086, 32-разрядные процессоры позволяют адресовать до 64К однобайтных или 32К двухбайтных регистров в пространстве, отдельном от памяти. Дополнительно имеется возможность обращения к 32-битным портам. При операциях ввода-вывода линии A[16:31] не используются. Адрес устройства задерживается либо в команде (только младший байт, старший – нулевой), либо берется из регистра DX (полный 16-битытный адрес). Команды ввода-вывода вызывают шинные циклы с активными сигналами IORD#, IOWR#. Строковые команды обеспечивают блочный ввод-вывод со скоростью, превышающей аналогичные операции со стандартным контроллером DMA. В адресном пространстве ввода-вывода область 0F8-0FF зарезервирована для использования сопроцессором (при обращении к сопроцессору 386 выставляет единицу на линии A31 шины адреса, что используется для упрощения дешифрации адресов).

В защищенном режиме инструкции ввода-вывода являются привилегированными. Это означает, что они могут исполнятся задачами только с определенным уровнем привилегий, определяемым полем IOPL

регистра флагов или битовой картой разрешения ввода-вывода (I/O Remission Bitmap), хранящийся в сегменте состояния задачи. Несанкционированная попытка выполнения этих инструкций вызовет исключение 13 – нарушение защиты (знаменитое сообщение «General Protection Error»).

1.3. Описание режимов адресации в R-mode

1.3.1. Режимы адресации данных

Непосредственный режим. Характеризуется тем, что данное длиной 8 или 16 бит является частью команды.

Пример: MOV dest, 20h.

Прямому режиму свойственно то, что 16-битовый эффективный адрес является частью команды. Эффективный адрес EA суммируется с умножением на 16 с содержимым соответствующего сегментного регистра.

Пример: MOV dest, [100h].

Регистровый режим обладает тем свойством, что данное содержится в определяемом командой регистре; 16-битовый операнд может находится в регистрах AX, BX, CX, DX, SI, DI, SP, BP, а 8-битовый – в регистрах AL, AH, BL, BH, CL, CH, DL, DH.

Пример: MOV dest, AX.

В регистровом косвенном режиме эффективный адрес данного находится в базовом регистре BX, BP или индексном регистре (SI, DI).

$$EA = \begin{Bmatrix} (BP) \\ (BX) \\ (SI) \\ (DI) \end{Bmatrix}.$$

Пример: MOV dest, [SI].

В *регистровом относительном (т. е. со смещением) режиме* эффективный адрес равен сумме 8- или 16-битового смещения и содержимого базового или индексного регистров. Этот тип адресации удобен для доступа к элементам таблицы, когда сдвиг указывает на начало таблицы, а регистр – на ее элемент.

$$EA = \begin{Bmatrix} (BP) \\ (BX) \\ (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} 8\text{-}битовое\,смещение \\ 16\text{-}битовое\,смещение \end{Bmatrix}.$$

Пример: MOV dest, [BX + 2]

Базовый индексный режим характеризуется тем, что эффективный адрес равен сумме содержимого базового и индексного регистров.

$$EA = \begin{Bmatrix}(BP)\\(BX)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}.$$

Пример: MOV dest, [BX + SI].

Относительный (т. е. со смещением) базовый индексный режим. Свойственно то, что эффективный адрес равен сумме 8- или 16-битового смещения и базового-индексного адреса. Этот режим удобен при адресации двумерных массивов, когда базовый регистр содержит начальный адрес массива, а значение смещения и индексного регистра задают сдвиг по строке и столбцу.

$$EA = \begin{Bmatrix}(BP)\\(BX)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix} + \begin{Bmatrix}8\text{-}битовое\,смещение\\16\text{-}битовое\,смещение\end{Bmatrix}.$$

Пример: MOV dest, [BX + SI + 4]

1.3.2. Режимы адресации переходов

Внутрисегментный прямой режим. Эффективный адрес равен сумме 8- или 16-битового смещения и текущего содержимого IP (относительная адресация)

Внутрисегментный косвенный режим. Эффективный адрес перехода есть содержимое регистра или ячейки памяти, которые указываются в любом, кроме косвенного, режиме адресации данных. Допустим только в командах безусловного перехода.

Межсегментный прямой режим. Непосредственно указываются значения CS и IP.

Межсегментный косвенный режим. Заменяет содержимое CS:IP содержимым двух смежных слов из памяти.

1.3.3. Организация системы команд

Система команд МП Intel 8086 содержит 135 базовых команд, которые мы разделили на шесть функциональных групп:

1 – пересылка данных;
2 – арифметика;
3 – логика;
4 – операции со строками;
5 – передача управления;
6 – управление МП.

Система команд микропроцессора Intel 8086 приведены в прил. 1.

Подробное описание групп команд смотри в книге [4].

Структура машинных инструкций микропроцессов серии Intel 80×86 в общем случае выглядит как показано на рис. 5.

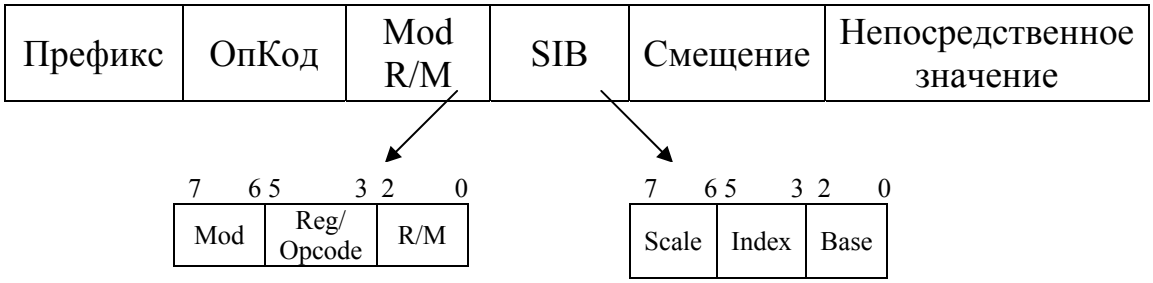


Рис. 5. Структура машинных инструкций микропроцессов серии Intel 80×86

Префикс – инструкция может содержать до четырех префиксов (или не содержать ни одного), размер каждого из которых – 1 байт

ОпКод – поле «ОпКод» содержит одно- или двухбайтный код инструкции

Mod R/M – однобайтное поле Mod R/M задает требуемый способ адресации и специфицирует используемые регистры

SIB – однобайтовое поле SIB расшифровывается как Scale-Index-Base и уточняет метод адресации

Смещение – в зависимости от метода адресации это поле занимает один, два или четыре байта и содержит смещение операнда

Непосредственное значение – непосредственный операнд занимает 1, 2 или 4 байт

В начале каждой команды может располагаться до четырех *префиксов* (от лат. *pre* – впереди и *fix* – прикреплений) - особых однобайтовых кодов, уточняющих поведение команды, - а может не быть ни одного - тогда команда ведет себя "по умолчанию".

Условно префиксы делятся на четыре группы:

1. Префиксы блокировки и повторения:
   0xF0 LOCK
   0xF2 REPNZ (только для строковых инструкций).
   0xF3 REP (только для строковых инструкций).
2. Префиксы переопределения сегмента:
   0x2E CS:
   0x36 SS:
   0x3E DS:
   0x26 ES:
   0x64 FS:
   0x65 GS:
3. Префикс переопределения размеров операндов:
   0x66
4. Префикс переопределения размеров адреса:
   0x67

Если используется более одного префикса из той же самой группы, то действие команды не определено и по-своему реализовано на разных типах процессоров.

Поле **ОпКод** занимает от одного до двух байт, а при необходимости еще и три бита поля «Reg/Opcode». Коды машинных инструкций имеют вполне определенную, регулярную структуру, со сложным сводом правил и исключений.

Назначение других полей машинных инструкций приведено на рис. 5. Дополнительную информацию можно найти в книге [1].

1.4. Основные понятия защищенного режима. Архитектура 32-разрядных процессоров

Разработка защищенного режима преследовала следующие основные цели:

- обеспечение многозадачности,
- защита кодов и данных одновременно выполняемых программ друг от друга,
- расширение адресного пространства,
- возможность работы с непрерывными массивами данных длины более 64 К.

С точки зрения программиста, фундаментальное различие реального и защищенного режимов состоит в следующем.

В реальном режиме операционная система только загружает программу в память и передает ей управление, а далее компьютер работает полностью под управлением программы пользователя (за исключением прерываний, когда операционная система временно берет управление на себя). Например, нетрудно написать программу без прерываний, в выполнение которой операционная система совершенно не будет вмешиваться.

В защищенном режиме программа пользователя, наоборот, все время работает совместно с операционной системой. В частности, ресурсов процессора уже не хватает на обеспечение механизма адресации, и в нем участвует операционная система.

Если бы мы захотели написать программу, которая монопольно управляет работой компьютера в защищенном режиме, нам это скорее всего не удалось бы, поскольку операционная система «не согласится» с передачей своих функций управления в программу. Конечно, гипотетической альтернативой здесь могла бы быть программа, частью которой является некоторая «кустарная» операционная система, поддерживающая защищенный режим.

Описанная в предыдущем разделе архитектура процессора характерна для моделей до 80286 включительно. В компьютер мог устанавливаться так называемый арифметический сопроцессор 80x87 для аппаратной

поддержки арифметики чисел с плавающей точкой. Работа с двойными словами аппаратно не поддерживалась.

Начиная с модели 80386, структура процессора Intel была существенно модернизирована (см. рис. 6). Основная цель этой – модернизации поддержка надежной работы защищенного режима, однако возможности реального режима были также существенно расширены (в частности, введена аппаратная поддержка двойных слов).

Программная модель. Программная модель 32-разрядного процессора изображена на рис. 7. Она содержит следующие группы регистров: регистры общего назначения; указатель команд; регистр флагов; сегментные регистры; регистры управления; системные адресные регистры; регистры отладки; регистры тестирования.

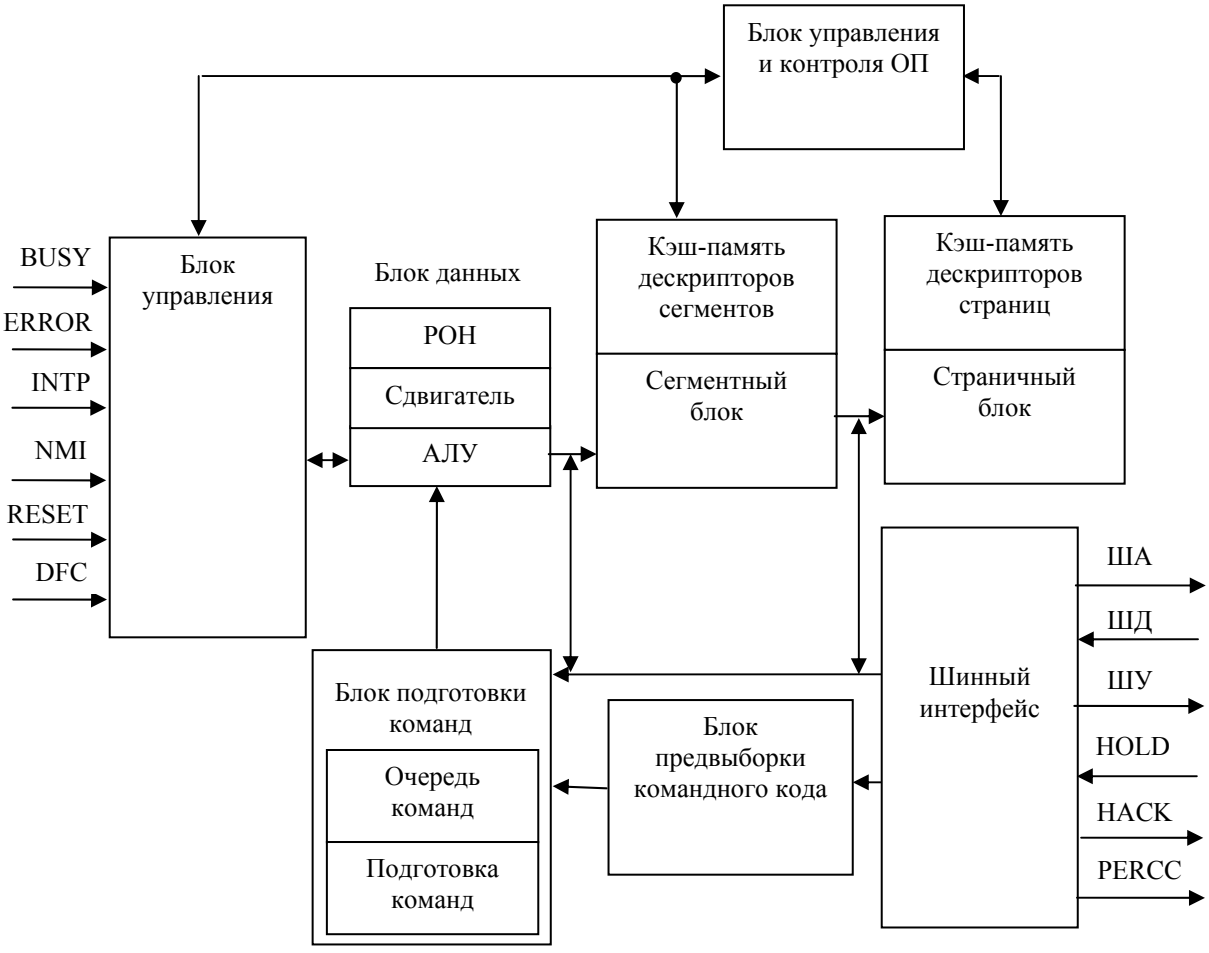


Рис.6.Структура микропроцессора I80386

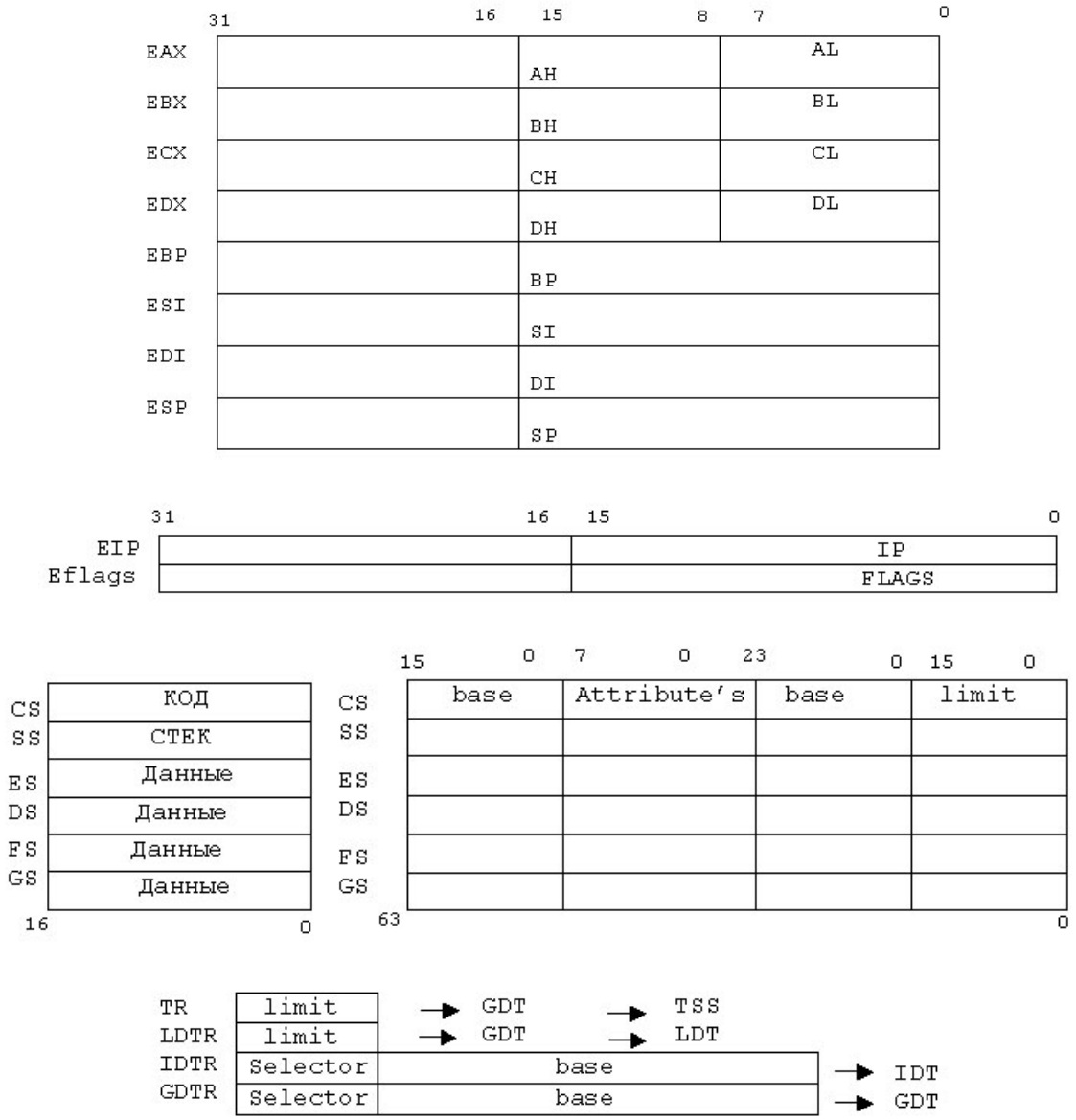


Рис. 7. Программная модель 32-разрядного процессора

Регистры общего назначения содержат все регистры данных и регистры-указатели МП i8086 и i80286 и столько же дополнительных 32-разрядных регистров. В обозначении 32-разрядных регистров используется начальная буква E (Expanded — расширенный).

Указатель команд EIP содержит смещение следующей выполняемой команды в сегменте кодов. При 16-разрядных адресах используются младшие 16 разрядов (IP).

Регистр флагов EF расширен до 32 разрядов. Младшие 16 разрядов регистра EF составляют регистр флагов F 16-разрядного процессора. В регистр EF добавлены новые флаги:

- флаг разрешения команды идентификации (ID, Identification Flag) процессора (CPUID, Central Processor Unit Identification) — для Pentium+ и некоторых процессоров типа 486;
- виртуальный запрос прерывания (VIP, Virtual Interrupt Pending) — для Pentium+;
- виртуальная версия флага разрешения (VIF, Virtual Interrupt Flag) прерывания IF для многозадачных систем (Pentium+);
- флаг контроля выравнивания (AC, Alignment Check). Используется только на уровне привилегий 3. Если ACC = 1 и AM = 1 (AM — бит в регистре управления CR0), то в случае обращения к операнду, не выровненному по соответствующему пределу (2, 4, 8)2 байт, будет вызвано исключение 17 (для1486+);
- флаг (VM, Virtual 8086 Mode) в защищенном режиме включает режим виртуального процессора 8086. Попытка использования привилегированных команд в этом режиме приведет к исключению 13;
- флаг возобновления (RF, Resume Flag). В режиме отладки единичное значение RF позволяет осуществить рестарт команды после прерываний. Используется совместно с регистрами точек отладочного останова.

Сегментные регистры. Кроме сегментных регистров МП i8086 и i80286 (DS, CS, SS, ES), программная модель содержит два дополнительных сегментных регистра данных: FS и GS. С каждым из шести сегментных регистров связаны теневые регистры дескрипторов. В теневые регистры в защищенном режиме переписываются 32-разрядный базовый адрес сегмента, 20-разрядный предел и атрибуты (права доступа) из дескрипторных таблиц.

Управляющие регистры. Для обеспечения страничной адресации и защиты, а также для управления процессом кэширования введены пять 32-разрядных регистров CR0-CR4 (Control Register).

В регистрах CR0—CR3 хранятся признаки состояния процессора, общие для всех задач. Младшие четыре разряда регистра CR0 содержат биты регистра MSW МП i80286 и некоторые другие биты управления. Регистр CR1 зарезервирован; в CR2 хранится 32-разрядный линейный адрес, по которому получен отказ страницы памяти; в 20 старших разрядах регистра CR3 хранится физический базовый адрес таблицы каталога страниц и биты управления кэш-памятью. Регистр CR4 (Pentium+) содержит биты разрешения архитектурных расширений МП.

Системные адресные регистры. Для обеспечения сегментной адресации (т. е. для работы с таблицами GDT и LDT) и для обработки прерываний введены регистры GDTR-6 байт, IDTR-6 байт, LDTR - 10

байт, TR - 10 байт. Восемь старших байт последних двух регистров также предназначены для копирования дескрипторов и программисту недоступны.

В системных регистрах-указателях глобальной дескрипторной таблицы GDTR и таблицы прерываний IDTR — хранятся 32-разрядные базовые адреса и 16-разрядные пределы таблиц соответственно. Системные сегментные регистры задач TR и локальной дескрипторной таблицы LDTR являются 16-разрядными селекторами. Им соответствуют теневые регистры дескрипторов, которые содержат 32-разрядный базовый адрес сегмента, 20-разрядный предел и байт права доступа.

Регистры отладки DR0—DR7 (Debug Register). Для целей отладки введены восемь 32-разрядных регистров DRO-DR7.

В регистрах DR0—DR3 (Debug Register) содержатся 32-разрядные адреса точек останова в режиме отладки; DR4—-DR5 зарезервированы и не используются; DR6 отображает состояние контрольной точки; DR7 — руководит размещением в программе контрольных точек.

Регистры тестирования TR (Test Register) входят в группу модельно-специфических регистров, их состав и количество зависят от типа процессора: в МП 386 используются два регистра TR6 и TR7, в Pentium II — TR1— TR12. Эта группа регистров содержит результаты тестирования МП и кэш-памяти.

Выше было упомянуто о расширении регистров общего назначения до 32 разрядов и о введении двух новых дополнительных сегментных регистров FS и *GS.* Расширен также регистр флагов *FLAGS* до 32-разрядного регистра EFLAGS, причем его новые биты 21-16 играют большую роль в обеспечении многозадачности и защиты.

На самом деле сегментные регистры *CS, DS, SS, ES, FS* и *GS* также расширены до 10 байт, но старшие 8 байт остаются недоступными для программиста. Иногда эти 8-байтные ячейки считаются отдельными регистрами и *называются теневыми регистрами* соответствующих сегментных регистров. При загрузке селектора в сегментный регистр процессор загружает в старшие 8 байт весь дескриптор, соответствующий этому селектору и далее уже не обращается к таблице дескрипторов. Кстати, в старших моделях процессора и при работе в реальном режиме в этих 8 байтах формируется структура, аналогичная дескриптору, хотя никакой таблицы дескрипторов нет.

Кроме того, в различные модели процессора Intel вводятся различные регистры для оптимизации процессов кэширования, сбора статистической информации о работе процессора и обращениях к памяти. Таких регистров могут быть десятки.

Для управления новыми регистрами введено более 30 новых команд, которые могут выполняться только в защищенном режиме.

Начиная с процессора 80486, арифметический сопроцессор был интегрирован с основным процессором.

Упрощенная структурная схема процессора Pentium приведена на рис. 8.

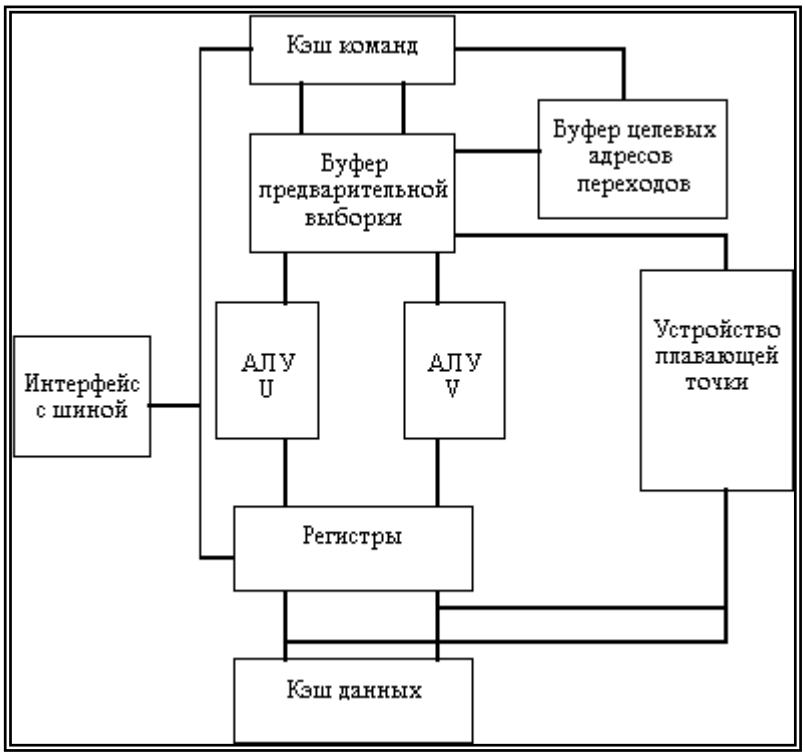


Рис. 8. Упрощенная структурная схема процессора Pentium (P5)

Процессоры Pentium первых версий содержали 32-разрядный регистровый файл РОН и 32-разрядную шину адреса. Внешняя шина данных 64-разрядна и образуется внутренним мультеплексированием шин данных процессора. Процессор поддерживает обращение до 4 Гбайт физического адресного пространства. Он содержит встроенный блок сопроцессора с плавающей точкой (FPU), блок управления и стабилизации напряжения питания (SMM, system management mode). Блок FPU доработан для выполнения операций с плавающей точкой за один такт синхронизации и располагает умножителем, делителем и сумматором. Блок SMM позволил CPU работать на пониженном напряжении питания 2,9 и 2,5В.

В кристалл процессора была встроена кэш-память L1, построенная по разделенной гарвардской архитектуре. Такая организация разделенной кэш памяти позволяет избежать конфликтов, связанных с доступом к командам и данным на различных стадиях конвейерной обработки. Организация этой памяти 2-х портовая, множественно-ассоциативная, с обратной записью. Процессор содержит логику синхронизации кэш-памяти для работы в многопроцессорной конфигурации.

Процессоры четвертого поколения (I80486) располагали единственным конвейером и назывались скалярными. Одновременно на ступень декодирования этого процессора может поступить только одна команда.

В базовой модели Pentium задействована иная методика обработки команд. Он содержит два конвейера, называемых U-pipe и V-pipe. В первом из них (первичном) выполняются целочисленные операции и команды с плавающей точкой, а во вторичном — простые целочисленные операции и определенный тип команд с плавающей точкой. Процессы в этих двух конвейерах протекают независимо друг от друга одновременно с двумя инструкциями. Такие процессоры, содержащие несколько независимо работающих конвейеров, называются суперскалярными (superscalar).

Процессор Pentium является двухпотоковым по числу конвейеров, но существуют и четырех и более потоковые микропроцессоры.

На одновременное выполнение двух команд наложены некоторые ограничения, связанные с комбинациями команд и взаимозависимостью результатов. Все команды, поступающие в конвейеры на обработку, должны покидать их точно в таком же порядке, в каком они на них и поступили. Такой метод обработки называется упорядоченным поступлением (in-orderissue), обработкой и упорядоченным завершением (in-order completion). Если команда не может быть завершена в одном конвейере, то останавливается и другой, что несколько снижало эффективность работы процессора с точки зрения производительности.

Существует и другой подход — неупорядоченное завершение (out-of-order completion), позволяющий одному из конвейеров завершать операцию даже при «заторе» в другом. В этом случае процессор может изменять очередность поступления на исполнение команд, обрабатывая их не в том порядке, в котором они следуют в программе, т.е. обрабатывая готовые к исполнению и откладывая на более поздний срок те из команд, которые не могут быть выполнены немедленно, что называется неупорядоченной обработкой (out-of order issue). Средства неупорядоченного завершения называются также средствами неупорядоченного исполнения (out-of-order execution). Для реализации подобного процесса в процессоре организуются дополнительные аппаратные узлы: буферы, окна команд, накопители команд.

Для повышения производительности процессора широко применяются методы обхода (data bypassing) и продвижения данных (data forwarding). При обходах результаты выполнения одной команды сразу пересылаются следующей, так что исключаются задержки на модификацию и повторное чтение из регистра процессора или ОЗУ. Продвижение данных позволяет процессору выполнять некоторые команды

параллельно, немедленно передавая результаты одной из них в другую, которой они не потребуются до более поздней ступени конвейерной обработки.

Как отмечалось, внутри микропрограмм выполняемых CISC-команд существует множество переходов и ветвлений. Переход — это изменение последовательности выполнения команды, которое может быть связано с дополнительной информацией (признаком состояния или условием). О существовании таковой команда «узнает» при промежуточном анализе состояний в ходе выполнения микропрограммы. Такой переход называется условным. Существуют и безусловные переходы, заведомо оговоренные логикой микропрограмм. Как те, так и другие переходы занимают при выполнении команд определенное время.

В процессоре Pentium применяется специальный буфер прогнозирования ветвлений {BTB, branch target buffer) на 256 позиций, отслеживающий и хранящий данные о результатах 256 последних ветвлений. Буфер — это место для хранения какой-либо информации, чаще всего буферы организуются для выравнивания скоростей между двумя, различными по быстродействию устройствами.

Опираясь на эту информацию, процессор пытается предсказать произойдет или не произойдет переход. Встретив команду условного перехода (по коду операции поля команды) процессор делает предположение о пути ветвления, которое может быть истинным (true) или ложным (false). Процессор начинает выполнение команды не сначала, а с предсказанного адреса микрокоманды перехода в соответствии со своим предположением. Может обеспечиваться несколько уровней прогнозирования. До окончательного утвердительного ответа на вопрос о переходе процессор не осуществляет никаких модификаций своих регистров и ОЗУ. В случае неправильного предположения все установки команды отменяются, а буфер очищается, что заметно снижает производительность процессора.

Дальнейшим развитием технологии P5 является доработка в 1996 г. процессора P55C, в результате чего стандартный Pentium был дополнен спецификацией расширения мультимедиа MMX {multimedia extantive). Массовое внедрение новой технологии Pentium MMX и Pentium MMX OverDrive позволило корпорации Intel уже в 1997 г. прекратить выпуск классических микропроцессоров Pentium.

Процессор Pentium MMX может обрабатывать дополнительно 57 MMX инструкций для поддержки режима мультимедиа. Процессор имеет параллельные очереди команд и выполняет за такт синхронизации, как и обычный P5, две инструкции.

Расширение MMX (MultiMedia eXtension) упрощает обработку больших потоков данных, связанных в основном со звуком и

изображением. Идея MMX состоит в использовании регистров арифметики с плавающей точкой для аппаратной поддержки учетверенных (64-битных) слов. Для MMX введена также новая арифметика «с насыщением». Например, если результат байтовой операции превышает 255, в режиме насыщения он полагается равным 255.

Далее в семействе P6 вводятся расширения уже к базовому набору команд MMX, например SSE.

Основная отличительная черта микроархитектуры процессоров семейства P6 – использование алгоритмики динамического выполнения команд (dynamic execution), которая построена на основе трех базовых концепций:

- предсказании переходов (branch prediction),
- динамическом анализе потока данных (dynamic data flow analysis)
- спекулятивном выполнении инструкций (speculative execution).

Микропроцессор Pentium Pro (professional) был анонсирован в 1995. Это процессор и положил начало микропроцессорам шестого поколения P6 (Pentium Pro, Pentium 2, Pentium 3,Celeron, Xeon) семейства Intel 8x86, которые имеют ряд отличий.

В процессорах семейства P6 предпринята попытка приблизиться к RISC-технологии. Сложная CISC-команда разбивается в процессоре на множество команд коротких форматов, напоминающих RISC-команды. Эти команды отправляются на выполнение каждая в своей очереди в конвейерах, которых в P6 три. Таким образом, одновременно за такт синхронизации может обрабатываться не две команды (инструкции), как в классическом Pentium, а три. Следует напомнить, что, в среднем, за один такт синхронизации в процессоре i386 выполняется пол-инструкции, а в i486 одна инструкция. После стадии обработки все составные части команды объединяются воедино и следуют на дальнейшее выполнение. Такие действия над командами называются динамическим выполнением dynamic execution. В отличие от классического Pentium, P6 выполняет инструкции с неупорядоченным завершением (out of order), глубина конвейера команд 12 ступеней. В P6 имеется динамический буфер предсказаний ветвлений и задействована система предположений возможных переходов между микропрограммами.

Логика, интегрированная в процессор поддерживает многопроцессорную симметричную конфигурацию вычислительных систем (SMP, symmetric multiprocessing). В многопроцессорных системах, построенных по симметричной схеме все процессоры, установленные на главной (хост) шине функционально равноправны и каждый из них может обмениваться данными с другими процессорами. Все эти процессоры имеют доступ к общему пространству памяти, если она построена по симметричной схеме. В этом случае, работа всех исполняемых программ и работающих

операционных систем не зависит от количества процессоров в вычислительной системе. Аналогично, SMP-системы с симметричным вводом-выводом позволяют всем процессорам иметь доступ к одним и тем же устройствам ввода-вывода (УВВ).

В отличие от симметричных многопроцессорных систем асимметричные системы (AMP) располагают узкоспециализированными процессорами для выполнения конкретных заданий. Каждый из процессоров такой системы располагает собственной областью ОЗУ и выделенным УВВ. В P6 поддерживается 4-процессорная конфигурация SMP, называемая MPS 1.1 (multi-processor specification).

Pentium 2 объединял в себе черты Pentium Pro и Pentium MMX, включая некоторые усовершенствования. Процессор оснащен кэш-памятью L1 16+16 Кбайт. Модуль процессора, т.е. сам процессор и кэш-память второго уровня L2 заключены во внутрь специального корпуса — картриджа SEC (single edge contact), который может быть установлен на системную плату в 242-контактный разъем Slot 1. Для повышения производительности процессора в нем использована концепция двух независимых шин (dual independent bus-DIP). Как отмечалось, в такой системе одновременно могут работать две шины; процессор — кэш-память L2 и процессор — ОЗУ.

Для повышения быстродействия подсистемы графической обработки предусмотрена поддержка средств работы с графическими приложениями (AGP, accelerated graphics port).Программная модель Pentium III представленная рис. 9

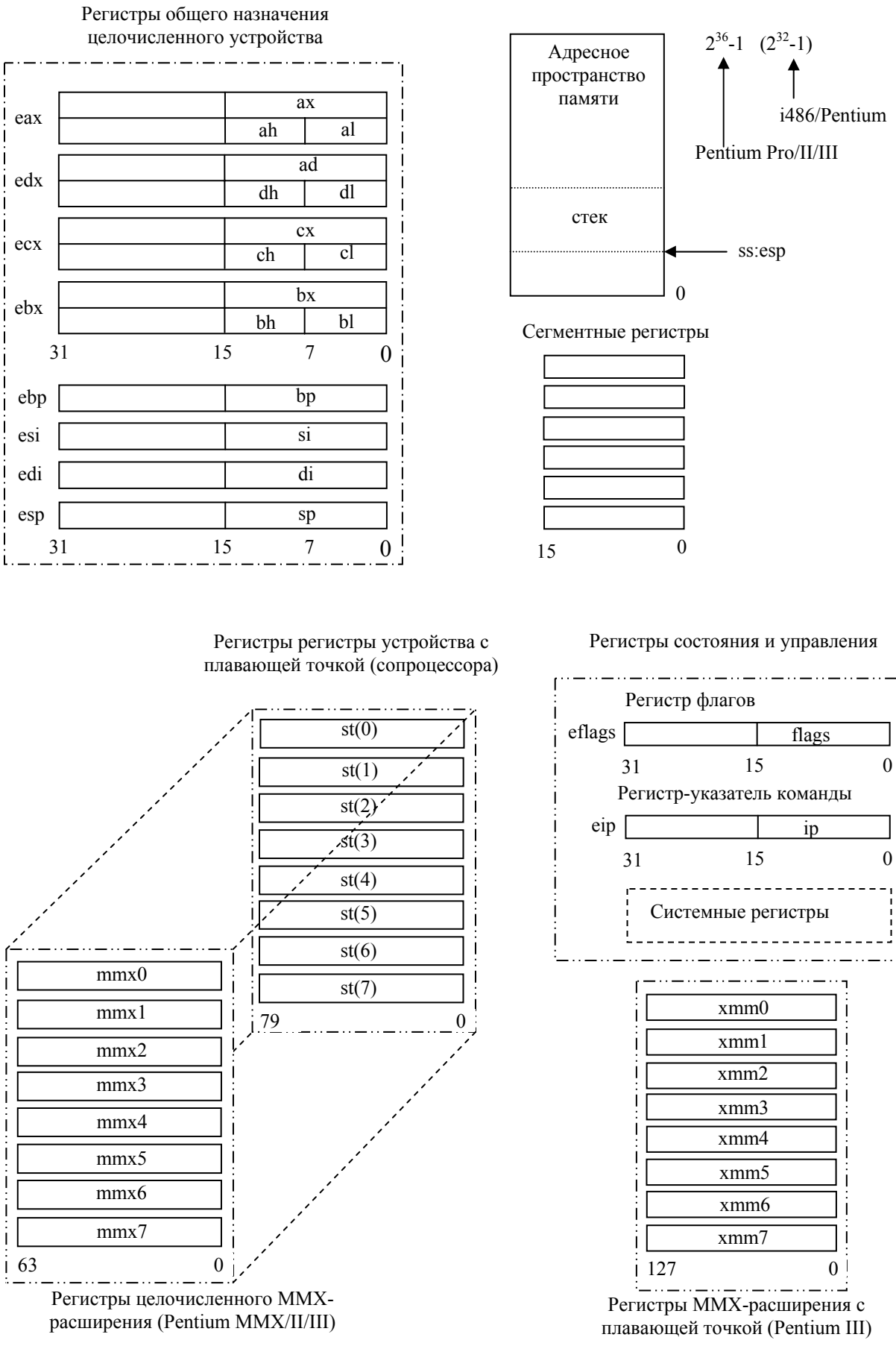


Рис.9.Программная модель микропроцессора Intel (Pentium 3)

Программную модель микропроцессора Intel с архитектурой P6 составляют:

- пространство адресуемой памяти (для Pentium III — до 2^{36}- 1 байт);
- набор регистров для хранения данных общего назначения;
- набор сегментных регистров;
- набор регистров состояния и управления;
- набор регистров устройства вычислений с плавающей точкой (сопроцессора);
- набор регистров целочисленного MMX-расширения, отображенных на регистры сопроцессора (впервые появились в архитектуре микропроцессора Pentium MMX);
- набор регистров XMM-расширения с плавающей точкой (впервые появились в архитектуре микропроцессора Pentium III);
- программный стек. Это специальная информационная структура, работа с которой предусмотрена на уровне машинных команд.

Регистры с точки зрения программиста можно разделить на две части: **пользовательские регистры** и системные регистры.

Системные регистры — это регистры для поддержания различных режимов работы, сервисных функций, а также регистры, специфичные для определенной модели микропроцессора. Функции этих регистров нами частично уже рассмотрены выше.

Пользовательские регистры

- регистры общего назначения eax/ax/ah/alt ebx/bx/bh/bl, edx/dx/dh/dl, ecx/cx/ch/cl, ebp/bp, esi/si, edi/di, esp/sp. Регистры этой группы используются для хранения данных и адресов;
- сегментные регистры cs, ds, ss, es, fs, gs. Регистры этой группы используются для хранения адресов сегментов в памяти;
- регистры сопроцессора st(0), st(l), st(2), st(3), st(4), st(5), st(6), st(7). Регистры этой группы предназначены для написания программ, использующих тип данных с плавающей точкой;
- целочисленные регистры MMX-расширения mmx0, mmxl, mmx2, mmx3, mmx4, mmx5, mmx6, imnx7. Регистры MMX-расширения с плавающей точкой xmm0, xmml, xmm2, xmm2, xmm3, xmm4, xmm6, xmm7.
- регистр указателя команды eip/ip.
- регистр флагов eflags/flags;

Микроархитектура процессора Pentium 4, получившая название NetBurst (пакетно-сетевая), приведена на рис.22 в книге [9]

1.5. Особенности архитектуры 64-разрядных МП

В 1997 году компании Intel и Hewlett-Packard разработали новую микропроцессорную архитектуру EPIC (Explicitly Parallel Instruction Computing — явного параллельного вычисления инструкций), которая была положена в основу 64-разрядных микропроцессоров IA-64, McKinley, Itanium, Itanium 2.

Особенностями архитектуры EPIC являются:

- большое количество регистров общего назначения. МП IA-64 содержит 128 64-разрядных регистров для операций с целыми числами и 12 880 — с дробными;
- поиск зависимостей между командами, причем поиск выполняется не процессором, а компилятором. Команды МП IA-64 группируются компилятором в «связку» длиной в 128 бит. Связка содержит 3 команды и шаблон, в котором указаны зависимости между командами (т. е. определяется, можно ли команду к2 запустить параллельно команде к1, или же к2 должна выполняться только после к1), а также между другими связками (можно ли с командой к3 из связки с1 запустить параллельно команду к4 из связки с2);
- масштабируемость архитектуры, т. е. приспособление набора команд к большому количеству функциональных устройств. Например, одна связка из трех команд соответствует набору из трех функциональных устройств процессора. Процессоры IA-64 могут содержать разное количество таких функциональных устройств, оставаясь при этом совместимыми по программному коду, поскольку вследствие того, что в шаблоне указана зависимость и между связками, процессору с N одинаковыми блоками из трех функциональных устройств будет соответствовать командное слово из 3N команд (N связок);
- предикация (Predication). Предикацией называется способ обработки условных разветвлений. Команды из разных ветвей условного ветвления обозначаются предикатными полями (полями условий) и выполняются параллельно, но их результаты не записываются, пока значения предикатных регистров не определены. Если в конце цикла определяется условие ветвления, один из предикатных регистров, соответствующий «правильной» ветви, устанавливается в единицу, а второй — в ноль. Перед записью результатов процессор проверяет предикатное поле и записывает результаты лишь тех команд, предикатное поле которых содержит единицу;
- загрузка по предположению (Speculative loading). Этот механизм предназначен для снижения простоев процессора, связанных с ожиданием

выполнения команд загрузки из относительно медленной основной памяти. Компилятор перемещает команды загрузки данных из памяти так, чтобы они выполнились как можно раньше. Тогда в случае, если для выполнения какой-либо команды МП понадобятся данные из памяти, процессор не будет простаивать.

Процессор Itanium 2 способен выполнять шесть команд за один машинный цикл. В совокупности с повышением тактовой частоты и пропускной способности системной шины (6,4 Гб/с, частота шины — 400 МГц, разрядность шины — 128), этот фактор обеспечивает в 1,5—2 раза большую производительность, чем в первом Itanium. Процессор имеет большой объем кэш-памяти третьего уровня (до 3 Мбайт), расположенной на кристалле и работающей на частоте ядра.

64-разрядные МП разработанные фирмой AMD, базируются на архитектуре ×86-64, которая является расширением архитектуры 32-разрядных процессоров ×86-32 (рис. 10-а и рис. 10-б).

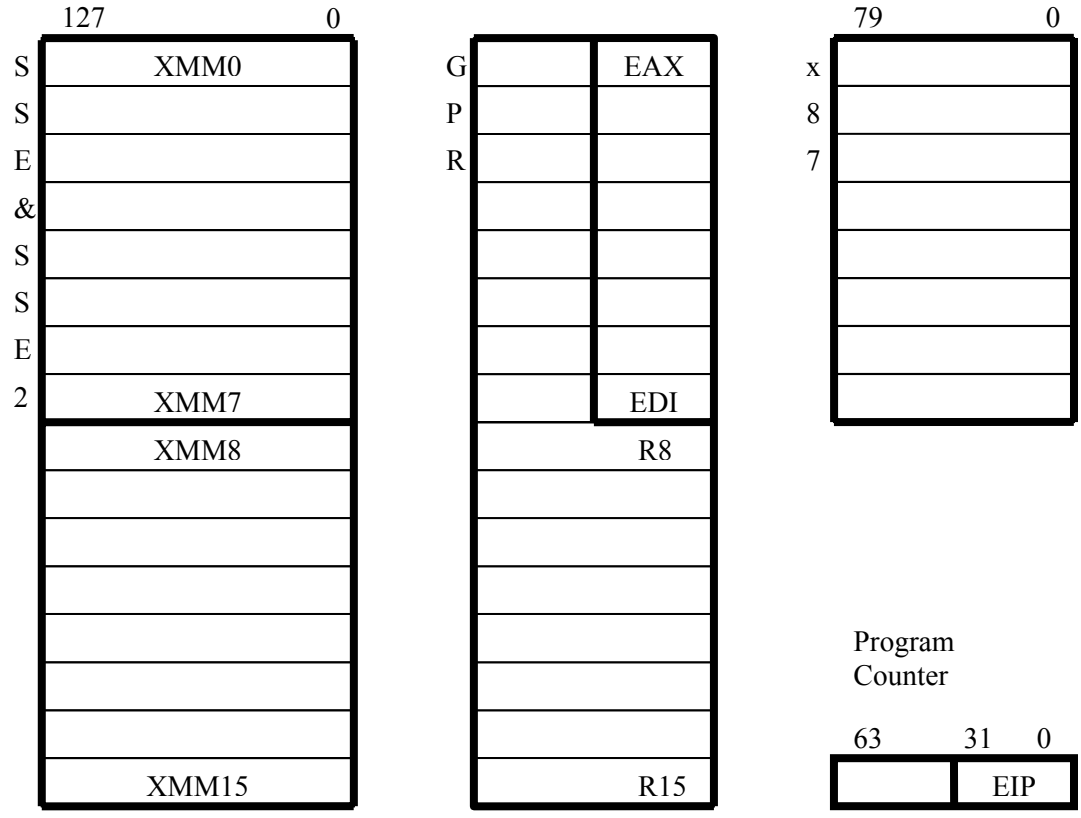


Рис. 10-а. Архитектура процессоров ×86-64

Регистры общего назначения (GPR) дополнены еще 8 регистрами R8—R15, используемыми в 64-разрядном режиме, а существующие EAX, EBX и другие расширены с 32 до 64 разрядов. В блок SSE (Streaming SIMD Extensions) в дополнение к восьми 128-разрядных регистрам

XMM0—XMM7, введены восемь новых регистров XMM8—XMM15, что обеспечит поддержку SSE2 путем увеличения SIMD-команд.

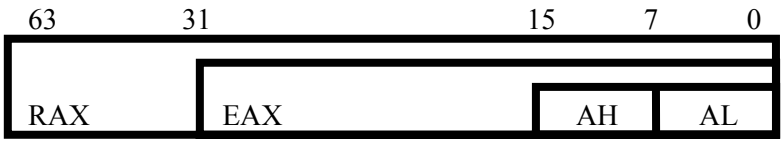


Рис. 10 б. Расширение регистров общего назначения

Процессор, построенный на основе x86-64, может выполнять существующие 32-разрядные приложения в отличие от, например, Intel Itanium, в котором систему команд x86-32 необходимо эмулировать.

1.6. Адресное пространство 32-разрядных процессоров. Принципы адресации в защищенном режиме

Память для процессоров 80×86 разделяются на байты, слова (16 бит), двойные слова (32 бит). Слова записываются в двух смежных байтах, начиная с младшего. Адресом слова является адрес его младшего байта. Двойные слова записываются в четырех смежных байтах.

Более крупными единицами являются страницы и сегменты. Память может логически организовываться в виде одного или множества сегментов переменной длины (**в реальном режиме – фиксированной**). Сегменты могут выгружаться на диске и по мере необходимости с него подкачиваться в физическую память. Кроме сегментации, в **защищенном режиме** возможно разбиение логической памяти на страницы, каждая из которых может отображаться на любую область физической памяти. Сегментация и разбиение на страницы могут применяться в любых сочетаниях. **Сегментация** является средством организации логической памяти, используемым на прикладном уровне. **Разбиение на страницы применяются на системном уровне для управлении физической памятью**.

Применительно к памяти различают три адресных пространства: логическое, линейное и физическое. Основным режимом работы 32-разрядных процессоров считается защищенный режим, в котором работают все механизмы преобразования адресных пространств.

Логический адрес, также называется виртуальным, состоит из **селектора** (в реальном режиме – просто сегмента) и **смещение**. Смещение формируется суммированием компонентов (base, index, disp) в эффективный адрес. Поскольку каждая задача может иметь до 16К

селекторов, а смещение, ограниченное размером сегмента, может достигать 4 Гб, логическое адресное пространство для каждой задачи может достигать 64 Тб. Все это пространство виртуальной памяти в принципе доступно программисту (этот «принцип» должна реализовывать операционная система).

Блок сегментации транслирует логическое адресное пространство в **32-битное пространство линейных адресов**. Линейный адрес образуется сложением базового адресного сегмента с эффективным адресом. Базовый адрес сегмента в реальном режиме образуется умножением содержимого используемого сегментного регистра на 16 (как и в 8086). В защищенном режиме базовый адрес загружается из дескриптора, хранящегося в таблице, по селектору, загруженному в используемый сегментный регистр.

Физический 32-битный адрес памяти образуется после преобразования линейного адреса блоком страничной переадресации. В простейшем случае (при отключенном блоке страничной переадресации) линейный адрес совпадает с физическим – присутствующим на внешней шине адреса процессора. Включенный блок страничной переадресации осуществляет трансляцию линейного адреса в физический блоками (страницами) размером 4 Гб. Этот блок может включаться только в защищенном режиме.

Как и у процессоров 8086/8088, для обращения к памяти процессор 80x86 (совместно с внешней схемой) формирует шинные сигналы MEMWR# (Memory Write) и MEMRD (Memory Read) для операции записи и считывания соответственно. Шина адреса разрядностью 32 бита позволяет адресовать 4 Гб физической памяти, но в реальном режиме доступен только 1 Мб, начинающийся с младших адресов.

В реальном режиме размер сегмента фиксирован – как и в 8086, он составляет 64 Кб (FFFFh). Попытка использования эффективного адреса, выходящего за границы сегмента, при 32-битной адресации вызывает исключение типа 13. При 16-битной адресации при вычислении эффективного адреса возможный перенос в разряд A16 игнорируется, и сегмент «сворачивается кольцом» (как и в 8086). Средства контроля следят и за переходом через границу сегмента во время обращения по «приграничному» адресу. При попытке адресации к слову, имеющему смещение FFFFh, или двойному слову со смещением FFFDh-FFFh (их старшие байты выходят за границу сегмента), или выполнения инструкции, все байты которой не умещаются в данном сегменте, процессор вырабатывает прерывание – исключение типа 13 (0Dh) – Segment Overrun Exception. При попытке выполнения инструкции ESCAPE с операндом памяти, не умещающимся в сегменте,

вырабатывается исключение типа 9 – Processor Extension Segment Overrun Interrupt (только для 386).

Система команд 32-разрядных процессоров предусматривает 11 режимов адресации операндов. Из них только два не имеют отношение к памяти:

- непосредственный операнд (8, 16 или 32-бит), который может содержаться в самой команде (непосредственная адресация).
- операнд-регистр, который может находится в любом 8, 16 или 32-битном регистре процессора(регистровая адресация).

Остальные девять режимов (табл. 8.) используются при формировании эффективного адреса операнда из памяти.

Эффективный адрес вычисляется с использованием комбинации следующих компонентов:

- Смещение (Displacement или Disp) – 8-, 16- или 32-битное число, включенное в команду.
- База (Base) – содержимое базового 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 инструкции. Этот элемент используется для указания размера элемента массива. Доступен только в 32-битном режиме адресации.

Эффективный адрес вычисляется по формуле EA=Base+Index*Scale+Disp.

Отдельные слагаемые в этой формуле могут и отсутствовать. Возможные режимы адресации приведены в табл. 6.

таблица 6

Режимы адресации памяти 32-битных процессоров

Прямая адресация	EA=Disp
Косвенная регистровая адресная Register Index Mode	EA=Base
Базовая адресация Based Mode	EA=Base+Disp
Индексная адресация Index Mode	EA=Index+Disp
Масштабированная индексная адресации Scaled Index Mode	EA=Scalex*Index+Disp
Базово-индексная адресация Based Index Mode	EA=Base+Index
Масштабированная базово-индексная адресация Based Scaled Index Mode	EA=Base+Scale* Index
Масштабированная базово-индексная адресация Based Index Mode with Displacement	EA=Base+Index+Disp

Масштабированная базово-индексная адресации со смещение Based Scaled Index with Displacement	EA=Base+Scale*Index+ Disp*

Процессор может использовать режимы 32-битной или 16-битной адресации. Режим 16-битной адресации соответствует режимам процессоров 8086 и 80286, при этом в качестве компонентов адреса используются младшие 16 бит соответствующих регистров.

Режим 32-битной адресации использует расширенные 32-разрядные регистры и имеет дополнительные режимы, использующие масштабирование индекса. Различия 16- и 32-битных режимов адресации приведены в табл. 7.

В реальном режиме по умолчанию используется 16-битная адресация, но с помощью префикса изменение разрядности адреса (Address Length Prefix) для текущей инструкции можно переключиться в 32-битный режим. При этом появляются дополнительные возможности адресации (масштабирования), но вычисленное значение эффективного адреса все равно не может преодолеть 64-килобайтный барьер – при такой попытке генерируется исключение 13 – General Protection Fault.

В защищенном режиме адресация по умолчанию определяется битом D дескриптора используемого кодового сегмента: при D=0 – 16 бит, при D=1 – 32 бита. Префикс разрядности адреса переключает разрядность для текущей инструкции на противоположную.

таблица 7

Различия режимов адресации

Компоненты	16-битная адресации	32-битная адресации
Базовый регистр	BX или BP	Любой 32-битный общего назначения
Индексный регистр	SI или DI	Любой 32-битный общего назначение, кроме ESP
Масштаб	Нет (всегда 1)	1, 2, 4 или 8
Смещение	0, 8 или 16 бит	0, 8 или 32 бит

При обращениях к памяти использование сегментных регистров по умолчанию определяется типом обращения (табл. 8). На время текущей инструкции при необходимости для большинства типов обращения возможно использование альтернативного сегментного регистра, на что указывает префикс замены сегмента (CS:; DS:; ES:; SS:; FS: или GS) перед кодом инструкции, например

ADD FS:[ESI],EAX ; [FS:ESI] <-[FS:ESI]+EAX.

таблица 8

Использование сегментных регистров при адресации памяти

Тип обращения к памяти	Сегментный регистр	
	по умолчанию	альтернативный
Выборка команд	CS	Нет
Стековые операции	SS	Нет
Строка-приемник	ES	Нет
Любые ссылки к памяти, кроме использующих в качестве базового регистры BP, EBP или ESP	DS	CS,ES,SS FS,GS
Ссылки к памяти, использующие в качестве базового регистры BP, EBP или ESP	SS	CS,DS,ES, FS,GS

Таким образом, при адресации в защищенном режиме адрес по-прежнему состоит из сегмента и смещения, однако:

- смещение может быть как 16-разрядным, так и 32-разрядным,
- размер сегмента ограничивается максимальной величиной смещения, т. е. величиной FFFF FFFF + I = 232 = 4 Гбайт,
- адрес сегмента недоступен для программиста и не указывается в программе.

В сегментных регистрах (которые по-прежнему 16-разрядные) содержатся так называемые селекторы. Биты 0 и 1 селектора определяют уровень привилегий при доступе к запрашиваемому сегменту, бит 2 определяет, по какой из двух таблиц (LDT или GDT– см. ниже) будет устанавливаться соответствие между селектором и сегментом. Остальные биты селектора 3-15 рассматриваются как номер дескриптора в одной из двух таблиц дескрипторов LDT/GDT(Local/GlobalDescriptor Table).

Таблицы дескрипторов создаются операционной системой и программа не имеет к ним прямого доступа (за исключением указания селекторов).

Каждый элемент таблицы дескрипторов, т. е. каждый дескриптор, имеет объем 8 байт и содержит информацию по отдельному сегменту, прежде всего его адрес и длину, которые в этом случае носят название базы и границы. База - - 32-разрядный адрес начала сегмента, граница – 20-разрядный размер сегмента (единица измерения может быть либо байт, либо 4096 байт в зависимости от значения других полей дескриптора).

Кроме того, в дескрипторе содержится много другой информации о сегменте: различные правила умолчания при работе с ним, 32/16-разрядный тип смещений, тип и принадлежность сегмента, уровень привилегий при доступе, режим доступа (чтение/запись/выполнение) и проч.

Для каждой задачи имеются две таблицы дескрипторов – глобальная таблица GDT (одна) и локальная таблица LDT (по одной на каждую задачу).

Итак, если мы указываем адрес в виде СЕЛЕКТОР. СМЕЩЕНИЕ, по значению селектора в соответствующей таблице дескрипторов отыскивается сегмент, и физический адрес получается как сумма базы и смещения (в том случае, когда не действует более сложный режим страничной организации памяти).

Подчеркнем, что установление связи между селекторами и сегментами выполняет операционная система и программист не может знать место расположения сегментов в физической памяти.

Подсчитаем максимальный размер адресного пространства при таком механизме адресации. Поскольку на индекс дескриптора в селекторе отводится 13 бит, в двух таблицах (GDT и LDT) может быть максимально $2\times2^{13} = 2^{14}$ дескрипторов. Размер каждого сегмента ограничен максимальной величиной 32-разрядного смещения, т. е. 2^{32}. Отсюда максимальный размер адресного пространства составит фантастическую величину $2^{14}\times2^{32} = 2^{46} = 64$ терабайт (1 терабайт = 1024 гигабайт).

Однако современный компьютер IBM с 32-разрядной адресной шиной максимально может адресовать только $2^{32} = 4$ Гбайт памяти. Адресацию большего объема памяти можно имитировать подкачкой необходимых сегментов с жесткого диска. Кстати, многие операционные системы так и поступают по другой причине – при недостаточном объеме установленной на компьютере физической памяти.

Конечно, сложный механизм адресации защищенного режима обеспечивает большую гибкость в распределении сегментов. Однако в одном случае можно получить адресацию даже более простую, чем в реальном режиме. Допустим, что базы всех сегментов установлены в ноль, разрядность всех смещений равна 32, а границы всех сегментов установлены в 4 Гбайт. Тогда получим так называемую модель памяти FLAT, в которой доступ к памяти осуществляется только указанием 32-разрядных смещений. Можно считать, что в этом случае сегментов вообще нет (в этом смысле модель FLAT часто называют бессегментной).

Однако модель FLAT обладает тем недостатком, что в ней невозможно защитить отдельные участки памяти. В целях такой защиты может применяться режим страничной организации памяти (pagination).

При такой организации вся память разбивается на страницы, которые представляют собой непрерывные участки памяти длиной 1000h - 4096 = 4 Кб (Pentium II может поддерживать и страницы по 4 Мб).

Адресация при страничном режиме двухуровневая. Данные по 1024 страницам сведены в таблицу с 4-байтными элементами (кстати, такая таблица также является также страницей). Число таких таблиц может

доходить до 1024. Данные по всем таблицам также сведены в одну таблицу с 4-байтными элементами, которую можно назвать главным каталогом таблиц страниц. Таким образом, страничная организация охватывает максимально 1024×1024×4096 = 4 Гб.

При страничной организации 32-разрядное смещение рассматривается следующим образом. Биты 31-22 -- номер таблицы страниц в каталоге, биты 21-12 – номер страницы в таблице страниц, биты 11-0 – смещение от начала страницы. Для ускорения выбора страниц предусмотрено кэширование, т. е. хранение в буфере адресов недавно использованных страниц.

Страничная организация памяти позволяет решить две следующие основные задачи:

- Защита отдельных страниц (или даже целых групп страниц) путем указания в элементах таблицы страниц (или в элементах каталога таблиц) приблизительно той же информации, что и в дескрипторе сегмента (см. выше).
- Организация расширенной виртуальной памяти путем подкачки с жесткого диска нужных страниц.

Защита как сегментов, так и страниц осуществляется следующим образом. Каждому сегменту или странице присваивается уровень привилегий, который указывается в дескрипторах сегментов и в элементах страничных таблиц. Каждой программе также присваивается некоторый уровень привилегий. При каждом обращении к памяти процессор проверяет соответствие привилегий и при его нарушении генерирует исключение.

Кроме того, при обращении к памяти производятся другие многочисленные проверки, например, запрещена запись в сегмент, помеченный как сегмент кода команд. Аналогичная защита предусмотрена и у страниц.

Таким образом, организация памяти зависит от режима работы МП. В реальном и виртуальном режимах i8086 адресация памяти такая же, как в МП i8086. В защищенном режиме используется сегментная и страничная организация памяти. Сегментная организация используется на прикладном уровне, а страничная — на системном. Формирование адреса ячейки памяти в защищенном режиме представлено на рис. 11.

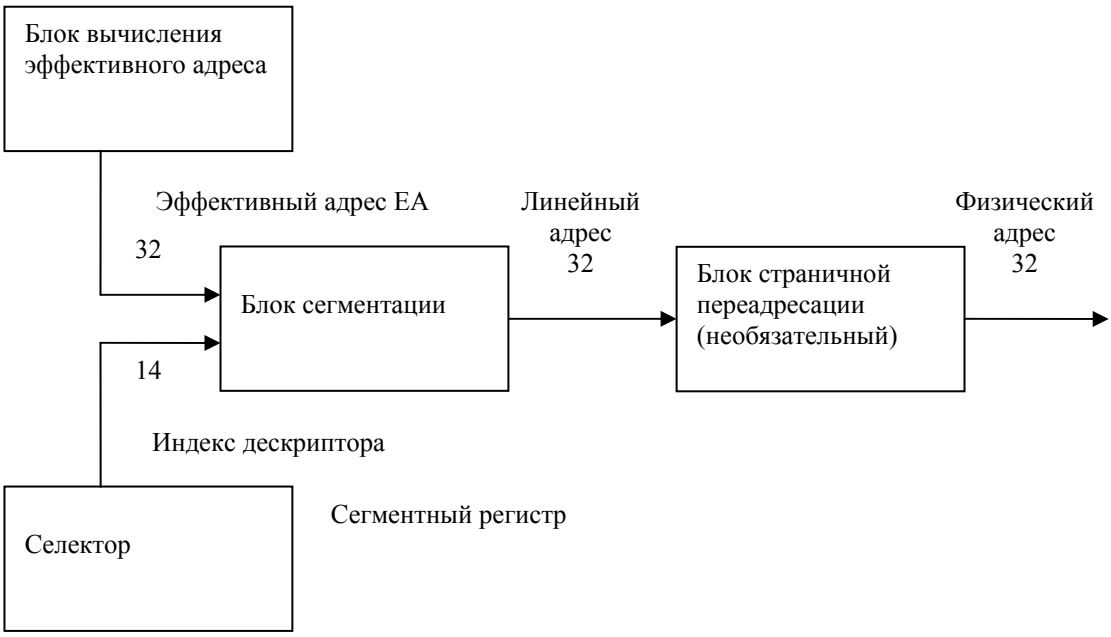


Рис. 11. Формирование адреса ячейки памяти в защищенном режиме.

Блок сегментации преобразует пространство логических адресов в пространство линейных адресов. Исходными данными для блока сегментации являются смещение EA в сегменте и сегментный регистр, которые задаются в команде. Содержимое сегментного регистра в защищенном режиме является селектором. Он содержит информацию о типе дескрипторной таблицы (глобальной или локальной) и индекс дескриптора. Индекс дескриптора является номером дескриптора в таблице. Дескриптор содержит базовый адрес сегмента. Линейный адрес определяется сложением базового и эффективного адресов . Блок страничной переадресации формирует физический адрес памяти. При отключенном блоке линейный адрес совпадает с физическим. Блок вычисления эффективного адреса вычисляет смещение операнда в сегменте согласно типу адресации, приведенному в табл. 6.

1.7. Развитие системы команд микропроцессоров 80×86

Расширение MMX ориентированно на мультимедийное, 2D и 3D-графическое и коммуникационное применение. Основная идея расширения MMX заключается в одновременной обработки нескольких элементов данных за одну инструкцию – так называемая технология SIMD (Single Instruction – Multiple Data).

Расширение MMX использует новые типы упакованных 64-битных данных:

- упакованные байты (Packed byte) – восемь байт;
- упакованные слова (Packed word) – четыре слова;

- упакованные двойные слова (Packed doubleword) – два двойных слова;
- учетверенное слово (Quadword) – одно слово.

Эти типы данных могут специальным образом обрабатываться в регистрах MMX0-MMX7, представляющих собой младшие 64 биты стека 80-битных регистров FPU. Как и регистры FPU, эти регистры не могут использоваться для адресации памяти, совпадение регистров FPU и MMX накладывает ограничения на чередование кодов FPU и MMX – забота об их независимости лежит на программисте приложений MMX.

Еще одна особенность технологии MMX – поддержка арифметики с насыщением (saturating arithmetic). Ее отличие от обычной арифметики с циклическим переполнением (wraparound mode) заключается в том, что при возникновении переполнения в результате фиксируется максимальное возможное значение для используемого типа данных, а перенос игнорируется. В случае антипереполнения в результате фиксируется минимальное возможно значение. Граничные значения определяются типом (знаковые или беззнаковые) и разрядностью переменных. Такой режим вычислений актуален, например, для вычисления цветов в графике.

Одновременно обрабатываемое 64-битное слово может содержать как одну единицу обработки, так и 8 однобайтных, 4 двухбайтных или 2 четырехбайтных операнда. Новые инструкции включают следующие группы:

- арифметические (Arithmetic Instructions), включающие сложение и вычитание в разных режимах, умножение и комбинацию умножения и сложения;
- сравнение (Comparison Instructions) элементов данных на равенство или по величине;
- преобразование форматов (Conversion Instructions);
- логические (Logical Instructions) – И, И-НЕ, ИЛИ и Исключающее ИЛИ, выполняемые над 64-битными операндами;
- сдвиги (Shift Instructions) – логические и арифметические;
- пересылки данных (Data Transfer Instructions) между регистрами MMX и целочисленными регистрами или памятью;
- очистка MMX (Empty MMX State) – установка признаков пустых регистров в слове тегов.

Инструкции MMX не влияют на флаги условий.

Регистры MMX, в отличии от регистров FPU, адресуются физически а не относительно значения TOS. Более того, любая инструкция MMX обнуляет поле TOS регистра состояния FPU. В слове тегов свободному регистру соответствует комбинация “11”, остальные комбинации

указывают только на занятость регистра. После каждой операции MMX биты тегов используемого регистра назначения обнуляются. Неиспользуемые в MMX биты [79:64] регистров FPU заполняются единицами, так что ошибочное использование данных MMX инструкций FPU приведет к исключению.

Инструкции MMX не порождают новых исключений. Исключения при их выполнении могут возникать только при нарушении границ при обращениях к памяти (данные и инструкции). Однако если предшествующая инструкция FPU породила условие исключения, то оно произойдет при выполнении инструкции MMX. После его обработки инструкция MMX может быть благополучна исполнена.

Инструкции MMX доступны из любого режима процессора. При переключении задач необходимо следить за корректностью сохранения контекста, как и при работе с FPU.

Часто чередование FPU и MMX может снизить производительность за счет необходимости сохранения и восстановления весьма объемного контекста FPU.

Команды SSE (начиная с Pentium III) делятся на 4 категории:

- SIMD-команды для данных одинарной точности с плавающей запятой (SPFP-команды);
- Дополнительные SIMD-команды для целочисленных данных;
- Команды управления кэшированием;
- Команды сохранения и восстановления компонент состояния процессора.

Одна SIMD-команда с плавающей запятой может обрабатывать одновременно четыре 32-разрядных числа одинарной точности с плавающей запятой (называемых SPFP-элементами данных).

SIMD-команды для работы с SPFP-данными используют восемь новых 128-разрядных регистров - XMM-регистров. Команды обращаются к XMM-регистрам по именам: XMM0, XMM1, :, XMM7.

Каждое 32-разрядное число с плавающей запятой имеет 1 знаковый бит, 8 битов порядка и 23 бита мантиссы, что соответствует стандарту IEEE-754 на формат представления чисел одинарной точности с плавающей запятой.

SIMD-команды поддерживают два типа операций над упакованными данными с плавающей запятой - параллельные и скалярные.

Параллельные операции, как правило, действуют одновременно на все четыре 32-разрядных элемента данных в каждом из 128-разрядных операндов. В именах команд, выполняющих параллельные операции, присутствует суффикс PS.

Скалярные операции действуют на младшие (занимающие разряды 0-31) элементы данных двух операндов. Остальные три элемента данных в

выходном операнде не изменяются (исключение составляет команда скалярного копирования MOVSS). В имени команд, выполняющих скалярные операции, присутствует суффикс SS.

SSE-команды имеют следующий синтаксис: instruction [dest, src]

Здесь instruction - имя команды, dest обозначает выходной операнд, src - входной операнд.

1.8. Прерывания и исключения

Прерывания и исключения нарушают нормальный ход выполнения программы для обработки внешних событий или сообщения о возникновении особых условий или ошибок.

Прерывания подразделяются на аппаратные (маскируемые и немаскируемые), вызываемые электрическими сигналами на выходах процессора, и программные, вызываемые по команде INT xx. Программные прерывания процессором обрабатываются как разновидность исключений.

Аппаратные прерывания подразделяются на маскируемые и немаскируемые. Процессор может воспринимать прерывания после выполнения каждой команды, длинные строковые команды имеют для восприятия прерываний специальные окна.

Маскируемые прерывания вызывает сигнал на входе INTR (Interrupt Request) при установленном флаге разрешения (IF=1). В этом случае процессор сохраняет с стеке регистр флагов, сбрасывает флаг IF и вырабатывает два следующих друг за другом (back to back) цикла подтверждения прерывания, в которых генерируются управляющие сигналы INTA# (Interrupt Acknowledge). Высокий уровень сигнала INTR должен сохраняться по крайней мере до подтверждения прерывания. Первый цикл подтверждения холостой, по второму импульсу внешний контроллер прерываний передает по шине номер вектора, обслуживающего данный тип аппаратного прерывания. Прерывание с полученным номером вектора выполняется процессором также, как и программное. Обработка текущего прерывания может быть в свою очередь прервана немаскируемым прерыванием, а если обработчик установит флаг IF, то и другим маскируемым аппаратным прерыванием.

Немаскируемые прерывания выполняются не зависимо от состояния флага IF по сигналу NMI (Non Mascable Interrupt). Высокий уровень на этом входе вызовет прерывание с типом (вектором) 2, которое выполняется также, как и маскируемое. Его обработка не может прерваться под действием сигнала на входе NMI до выполнения команды IRET.

Исключения (Exceptions) подразделяются на отказы, ловушки и аварийные завершения.

Отказ (fault) – это исключение, которое обнаруживается и обслуживается до выполнения инструкции, вызывающей ошибку. После обнаружения этого исключения выполнение возвращается снова на туже инструкцию (включая все префиксы), которая вызвала отказ. Отказы, использующиеся в системе виртуальной памяти, позволяют, например, подкачать с диска в оперативную память затребованную страницу или сегмент.

Ловушка (trap) – это исключение, которое обнаруживается и обслуживается после выполнения инструкции, его вызывающей. После обслуживания этого исключения управление возвращается на инструкцию, следующей за вызывающей ловушку. К классу ловушек относятся и программные прерывания.

Аварийное завершение (abort) – это исключение, которое не позволяет точно установить инструкцию, его вызвавшую. Оно используется для сообщения о серьезной ошибке, такой как аппаратное ошибка или повреждение системных таблиц.

Под исключения Intel резервирует векторы 0-31 в таблице прерываний, однако в PC часть из них перекрывается системными прерываниями BIOS и DOS.

Трактовка исключений и прерываний в защищенном режиме похожа на их трактовку в реальном режиме с тем отличием, что вместо таблицы векторов прерываний, находящейся на фиксированном месте (в начале) памяти, таблица дескрипторов прерываний IDT может быть расположена в произвольном месте в памяти, а ее адрес хранится в специально предназначенном для этого регистре процессора IDTR.

Напомним, что исключения могут быть внутренние, программные и внешние. Внутренние прерывания или исключения (exceptions) возникают в процессоре при возникновении в нем конфликтной ситуации (например, деления на ноль). Внешние прерывания поступают от внешних устройств. Программные прерывания инициируются самой программой (по команде int). В защищенном режиме в режиме многозадачности возникает четвертая разновидность прерываний -- переключение с одной задачи на другую.

Несмотря на разную природу прерываний, методы их обработки и в реальном, и в защищенном режимах совпадают – происходит передача управления на соответствующую процедуру обработки прерывания.

Общий сценарий обработки прерывания таков. В микропроцессоре возникает исключение, или исполняемая программа генерирует прерывание, или прерывание поступает от внешней аппаратуры на вход прерываний микропроцессора INT. Далее все происходит одинаково. Процессор определяет номер прерывания и дескриптор прерывания, соответствующий этому номеру. Дескриптор прерывания содержит,

помимо всего прочего, адрес обработчика прерывания. Далее в стеке сохраняется адрес возврата и управление передается на обработчик, который должен заканчиваться командой IRET.

По аналогии с реальным режимом все прерывания защищенного режима занумерованы (от 0 до 255), причем первые 32 номера закреплены именно за исключениями (в настоящее время используется только несколько первых номеров). Вся информация о прерываниях собрана в таблицу IDT, элементы которой называются дескрипторами прерываний, или шлюзами.

Дескрипторы прерываний, как и дескрипторы сегментов, имеют длину 8 байт, но структура их различна. Точнее можно сказать, что шлюзы являются стандартными, зарезервированными системой дескрипторами сегментов.

Соответственно указанным типам прерываний шлюзы, входящие в таблицу, могут быть трех типов: шлюзы задач – для переключения задач, шлюзы прерываний – для аппаратных прерываний и исключений, и шлюзы ловушек (trap) для программных прерываний. Тип данного шлюза указывается двумя байтами внутри кода этого шлюза.

В шлюзах, соответствующих программным прерываниям, указывается уровень привилегий программы, которая может пользоваться данным прерыванием.

2. УЧЕБНАЯ МИКРОПРОЦЕССОРНАЯ СИСТЕМА МИКРОЛАБ КМ 1810 ВМ 86

2.1. Краткое описание Микролаб

Микролаб КМ1810ВМ86 910 (дальнейшем просто Микролаб) предназначен для изучения элементов аппаратного и программного обеспечения устройств, построенных на базе микропроцессора КМ1810ВМ86.

В Микролабе имеется возможность архитектурного расширения КМ1810ВМ86 математическим процессором К1810ВМ87, который добавляет свои восемь 80-разрядных регистров, а также регистры состояния и управления к четырнадцати программно доступным регистрам КМ1810ВМ86.

Программа-монитор, хранящаяся в ПЗУ, обеспечивает возможность автономной работы Микролаба с просмотром занесенной в ОЗУ программы пользователя, ее редактирование и исполнение с использованием или без использования точки прерывания, а также в шаговом режиме.

При исполнении программы по командам (шаговый режим) имеется возможность просматривать и изменять содержимое всех регистров микропроцессора и ячеек памяти.

Для связи с пользователем микролаб КМ1810ВМ86 910 оснащен клавиатурой и встроенным индикатором. Предусмотрены разъемы для подключения периферийного устройства и связь с ПЭВМ по последовательному порту.

Память Микролаба включает:

- постоянное запоминающее устройство на 4 БИС К573РФ2 суммарной емкостью 8 Кбайт, расширено на 4 Кбайт установкой двух дополнительных БИС К573РФ2 в специальные панели, имеющиеся на плате Микролаба, адреса основного ПЗУ FE000 - FFFFF, дополнительного FD000 - FDFFF;
- оперативное запоминающее устройство на 8-ми микросхемах КР541РУ2 общей емкостью 4 Кбайт, адреса ОЗУ 00000 - 00FFF.

Ввод-вывод информации от внешних устройств осуществляется через два программируемых периферийных интерфейса КР 580ВВ55А. В Микролабе установлены устройства, имитирующие внешние порты ввода/вывода. В качестве датчиков параллельного кода используются восемь переключателей, а приемниками параллельной информации являются восемь светодиодов. Приемником последовательной информации является динамик.

Для связи с пользователем Микролаб имеет клавиатуру (24 клавиши)

и встроенный индикатор, вводимая и выводимая информация представлена в шестнадцатеричной системе счисления.

Микролаб обеспечивает следующие функции:

- опрос и изменение содержимого-памяти;
- опрос и изменение содержимого регистров МП;
- ввод и исполнение программ и подпрограмм пользователя;
- отладка программ пользователя с помощью шагового режима и точек прерывания;
- перемещение выбранного блока памяти с одного места на другое;
- считывание/запись данных из/в портов ввода/вывода.

Упрощенная структурная схема Микролаба представлена на рис. 12.

Плата микропроцессора строится на основе микросхем микропроцессорных наборов К1810 и КР580. В ее состав входят: блок центрального процессора, блок памяти, включающий ОЗУ и ПЗУ, клавиатуру, индикацию и устройства ввода/вывода.

Подробная структурная схема учебной микро-ЭВМ приведена на рис. 13. Расположение органов управления на плате показано на рис. 14.

Карта памяти учебной микро-ЭВМ приведена в табл. 9. На лицевой панели УМК расположены: клавиатура, дисплей, световая индикация.

С клавиатуры пульта осуществляется вызов следующих директив, Ÿ

Просмотр/изменение байта

БТ **<адрес>** **,** **[[<данные>]** **,** **]** **.**

Просмотр/изменение слова

С **<адрес>** **,** **[[<данные>]** **,** **]** **.**

Просмотр/изменение регистра

РГ **<имя регистра>** **,** **[[<данные>]** **,** **][** **.** **]**

Отображение байта данных из порта ввода

ВВБ **<адрес порта>** **,** **[** **,** **]** **.**

Отображение слова из порта ввода

ВВС **<адрес порта>** **,** **[** **,** **]** **.**

Вывод байта

ВЫВ.Б **<адрес порта>** **,** **<данные >** **[** **,** **<данные >]** **.**

Вывод слова

ВЫВ.С **<адрес порта>** **,** **<данные >** **[** **,** **<данные >]** **.**

Запуск программы пользователя

ПУСК **[<адрес порта>]** **[** **,** **]** **<адрес точки прерывания>** **.**

Перемещение блока данных

ПРС **<начальный адрес>** **,** **<конечный адрес>** **,** **<адрес назначения>** **.**

Использование отдельной команды

ШАГ **[<начальный адрес>]** **,** **.**

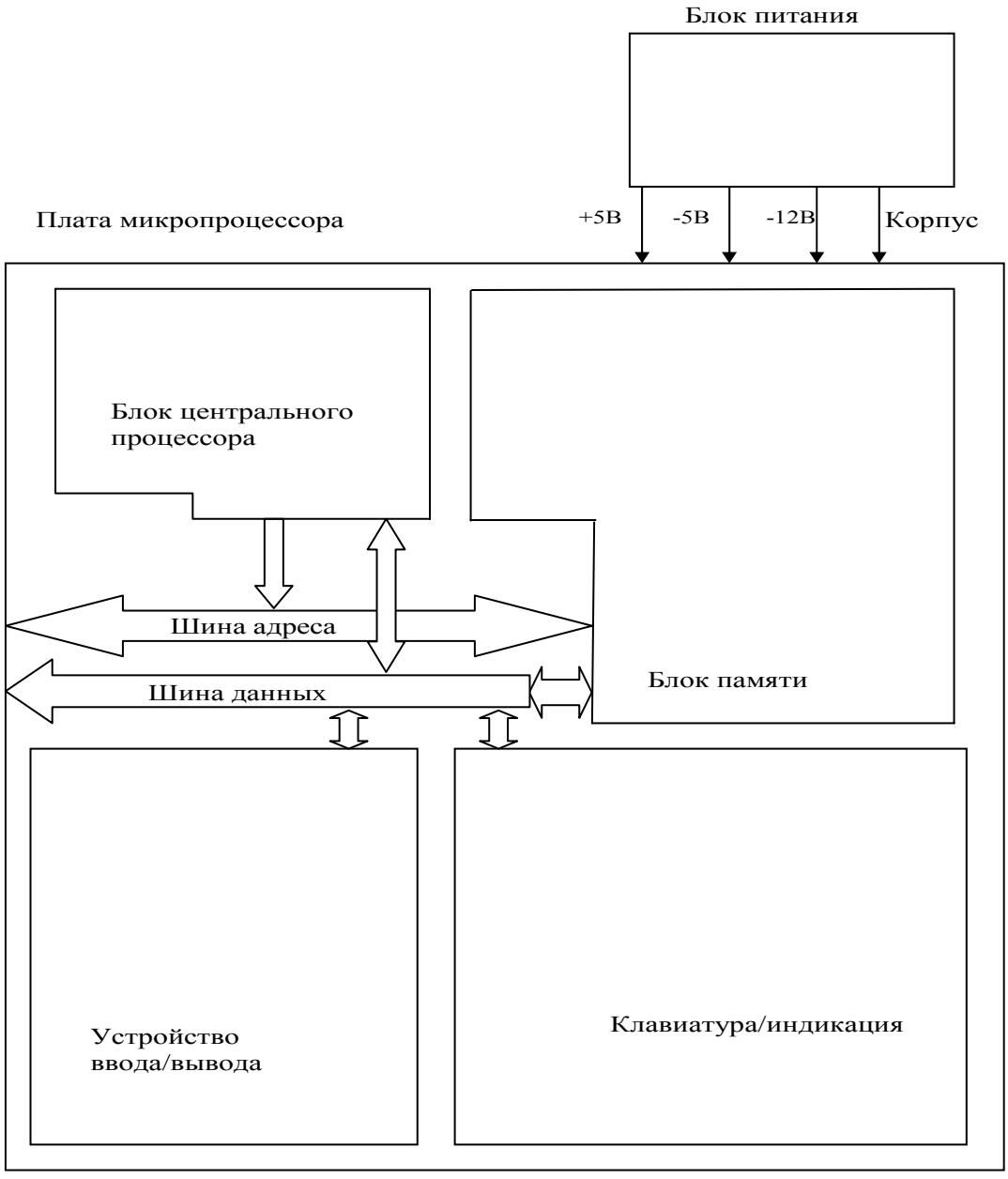


Рис. 12. Упрощенная структурная схема Микролаб

Ÿ

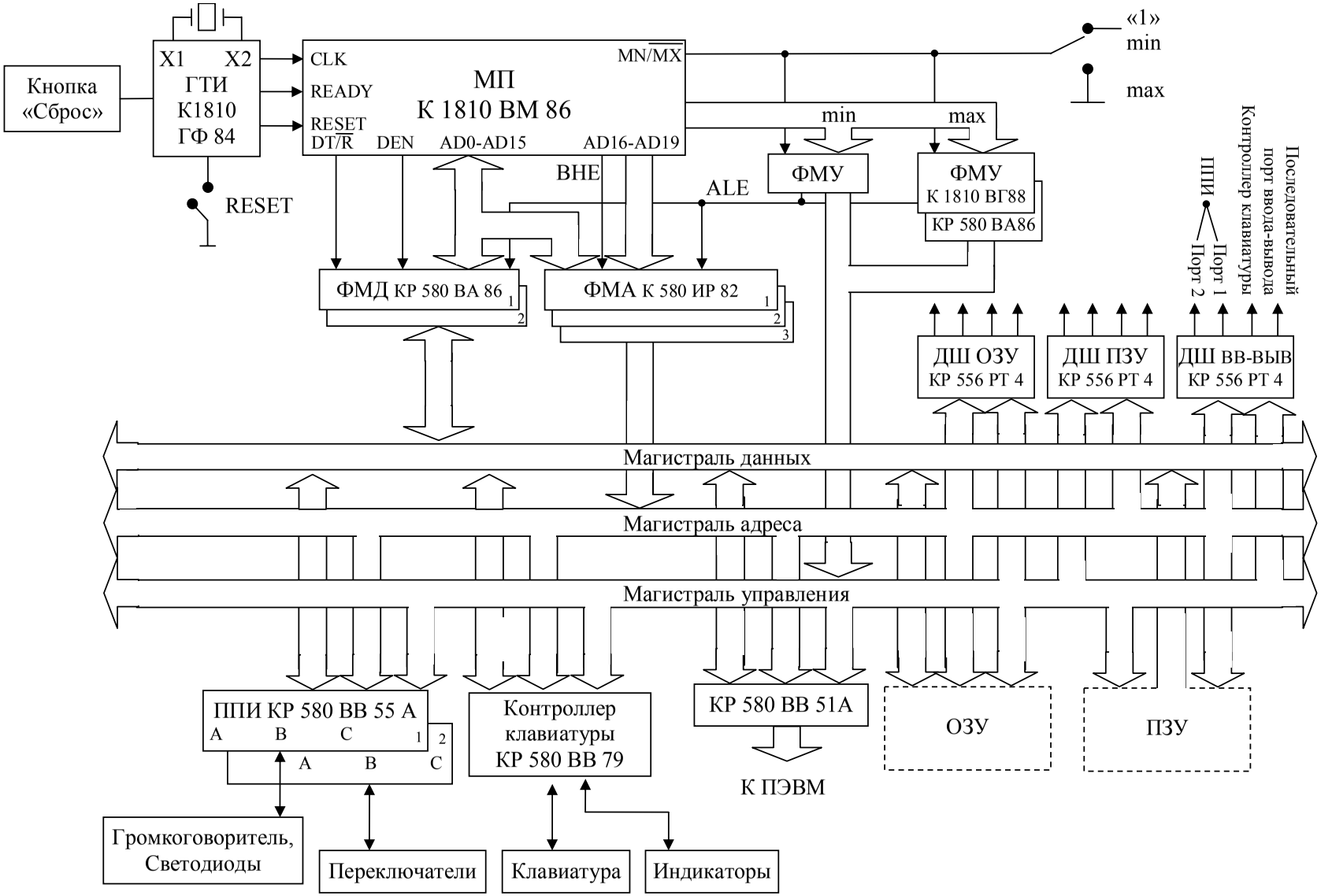


Рис. 13. Подробная структурная схема Микролаб

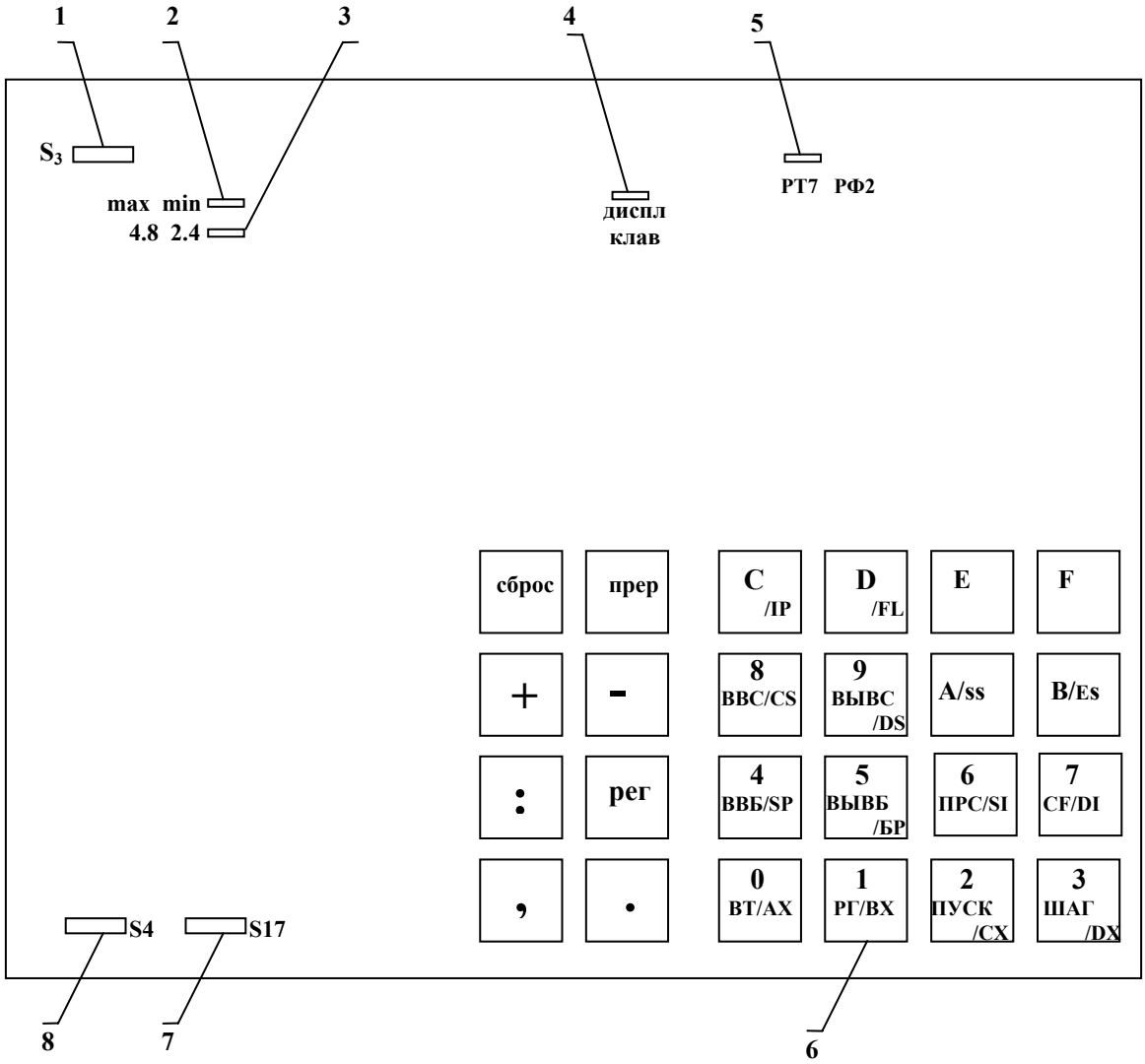


Рис. 14. Расположение органов управления на плате.

1 – переключатель выбора количества состояний ожидания

2 – перемычка выбора режима работы МП

3 – перемычка выбора рабочей частоты МП

4 – перемычка выбора программы монитора

5 – перемычка выбора типа дополнительных ПЗУ

6 – клавиатура

7 – переключатель выбора скорости работы последовательного интерфейса

8 – переключатели устройства ввода

таблица 9

Карта памяти

Адрес	Область памяти ЗУ	
00000 00013	Векторы прерываний 0--4	4К
00014 000CF	Область данных монитора	
000D0 000FF	Стек пользователя	
00100 00FFF	Область ОЗУ для хранения изменяющихся программ и данных пользователя	
01000 FCFFF	Неиспользованные адреса	
FD000 FDFFF	Область дополнительного ПЗУ	4К
FE000 FFFFF	Область основного ПЗУ	8К

таблица 10

Адрес	Назначение
FFE8	Чт/зп ОЗУ дисплея или чт очереди БИС КР 580 ВВ79
FFEA	Чт состояния или эп команды в БИС КР 580 ВВ79
FFF0	ЧТ/зп данных БИС КР 580 ВВ 51 А
FFF2	Чт состояния или Зп команды в БИС КР 580 ВВ 51 А
FFF8	Чт/Зп БИС КР 580 ВВ 55 А порт Р1А
FFF9	Порт Р2А
FFFA	Порт Р1В (светодиоды, громкоговоритель)
FFFB	Порт Р2В (переключатели)
FFFC	Порт Р1С
FFFD	Порт Р2С (магнитофон)
FFFE	Р1РУС -- Зп команды в БИС КР 580 ВВ 55 А порт Р1
FFFF	Р2РУС -- Зп команды в БИС КР 580 ВВ 55 А порт Р2

2.2. Адресация памяти и внешних устройств в Микролаб

2.2.1. Адресация внешней памяти в Микролаб

Адресуемая область памяти (в общем случае 1 Мбайт) поделена физически на две части, так называемые банки, размером до 512 Кбайт. Один банк связан с младшей половиной 16-разрядного канала данных (D7-Do), а другой – со старшей (D15-D8). Адрес (A19-A1) используется для того, чтобы адресовать ячейки памяти как в младшем, так и в старшем банках. Младший разряд адреса A_0 не используется при адресации ячеек памяти, но используется для адресации банка памяти. Младший банк, содержащий четно-адресуемые байты, выбирается, когда $A_0 = 0$. Старший банк, содержащий нечетно-адресуемые байты ($A_0 = 1$), выбирается по отдельному сигналу разрешения старшей половины канала (BHE/ = 0).

Механизм выбора банка по сигналам BHE/ и A_0 расшифровывается в табл. 11.

таблица 11

BHE	A_0	Пересылаемый байт
0	0	Оба байта
0	1	Старший байт в(из) нечетный адрес
1	0	Младший байт в(из) четный адрес
1	1	Ничего

Таким образом микропроцессор может оперировать как с одно-байтными, так и с двухбайтными данными. Все адресное пространство памяти, подключаемое к 16-разрядной МД, может быть представлено состоящим из двух банков по 512 Кбайт каждый (рис. 15). Банк нечетных адресов подключен к старшим (D15-D8), а банк четных адресов – к младшим (D7-D0) разрядам МД. Для доступа как к отдельным байтам, так и к двухбайтным данным памяти МП БИС имеет выход BHE/ (Bus High Enable), сигнал на котором разрешает выдачу старшего байта данных на МД.

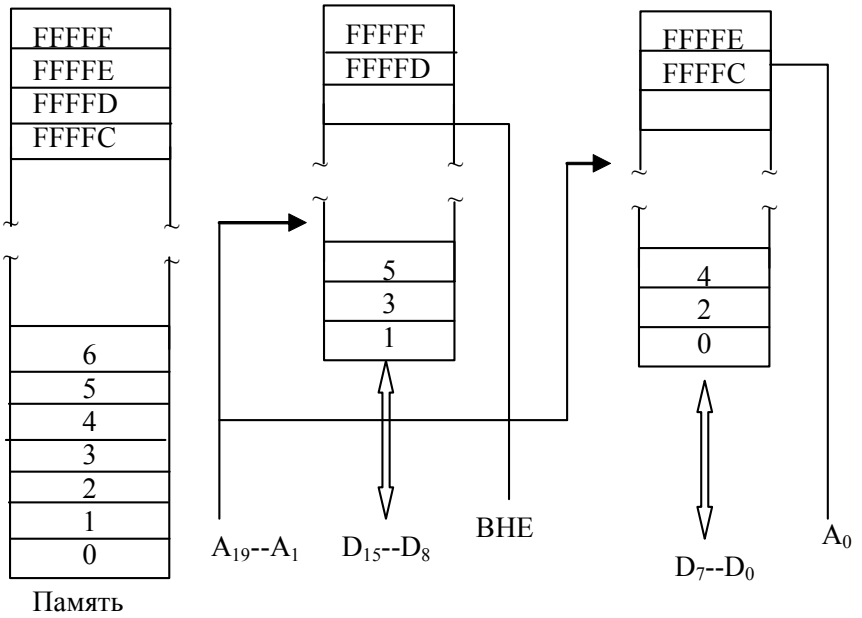


Рис. 15. Обращение МП БИС КР1810ВМ86 к памяти

Использование сигнала A_0 для выборки ряда с четными адресами позволяет выдавать на МД младший байт. Очевидно, что используя различные сочетания выборок рядов памяти (сигналов BHE/ и A_0), можно управлять одно- или двухбайтной передачей данных между памятью и микропроцессором.

2.2.2. Адресация внешних устройств в Микролаб

Подключение внешних устройств в составе Микролаб реализовано посредством четырех БИС: две микросхемы программируемого параллельного интерфейса КР580ВВ55А (микросхемы D38 и D39 в соответствии с обозначениями на принципиальной электрической схеме Мик-

ролаба), последовательный порт ввода/вывода КР580ВВ51 (микросхема D43), программируемый контроллер дисплея и клавиатуры КР580ВВ79 (микросхема D42).

Каждая из КР580ВВ55А содержит три 8-разрядных порта данных (обозначаемых буквами A, B, C), которые могут быть как входными, так и выходными, и один порт управления (рис. 16).

Микросхема D38 соединена с младшим байтом данных (D0-D7), а D39 обеспечивает связь со старшим байтом данных (D8-D15). Все порты могут быть адресованы индивидуально (например, P1A, P2C), или соответствующая пара портов (P1A и P2A, P1B и P2B) может быть адресована одновременно для образования 16-разрядного порта данных.

Адреса портов ввода/вывода для двух БИС параллельных интерфейсов приведены в табл. 12 и 13.

Управление выводом необходимых портов обеспечивает дешифратор ввода-вывода построенный на двух микросхемах: К155ЛА2 (микросхема D35) и К556РТ4 (микросхема D35). Во время операций с байтами дешифратор вырабатывает сигнал выбора соответствующего порта (выводы 11 и 12 микросхемы D34). При работе со словами для адресации желаемой пары портов задается только адрес порта P1, а дешифратор ввода/вывода генерирует сигналы выбора обоих портов одновременно.

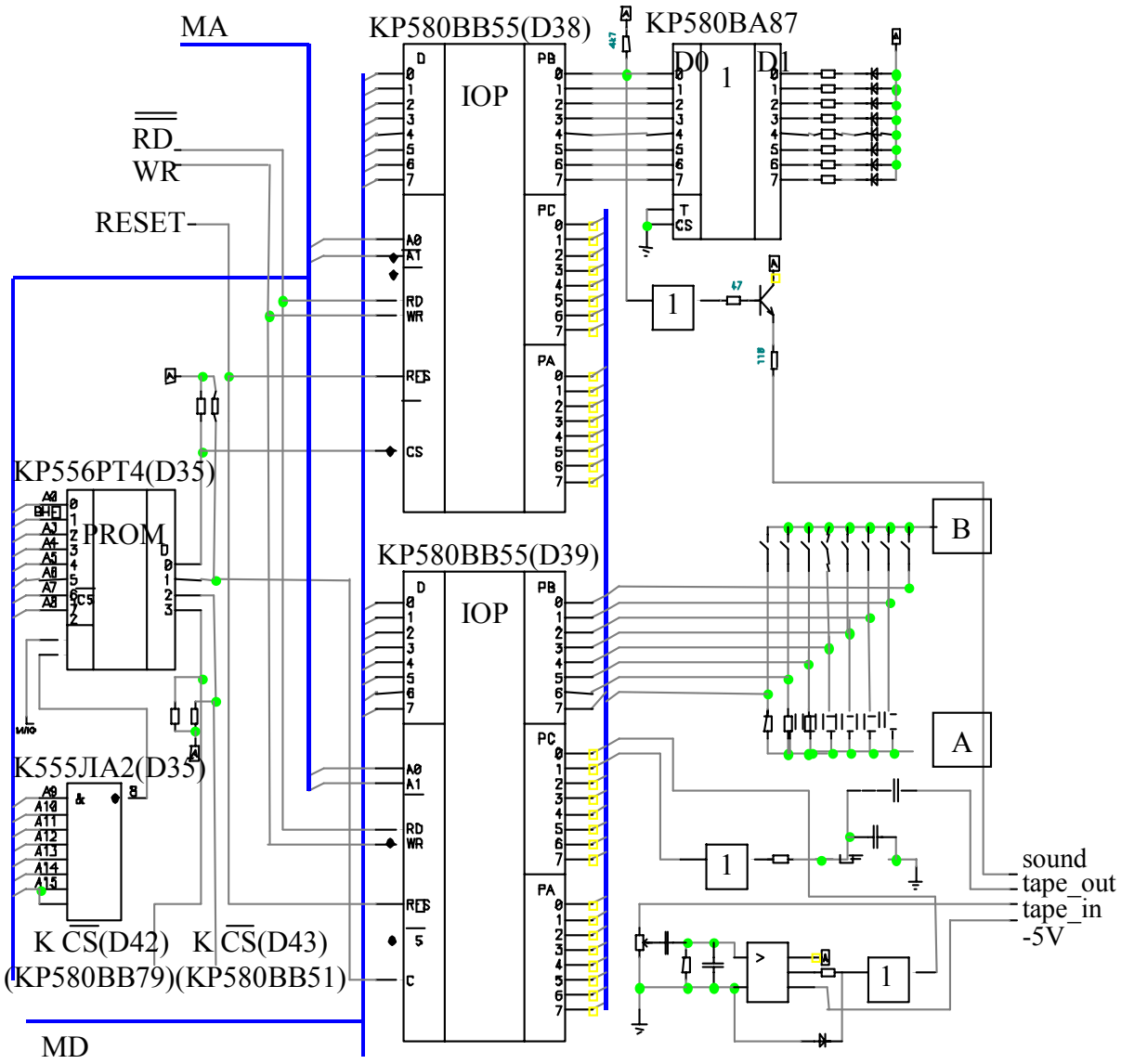


Рис. 16. Схема подключения параллельного программируемого интерфейса (ППИ) в комплекте Микролаб

Работа дешифратора ввода-вывода разрешается уровнем лог. 0 на входе CS1 микросхемы D34 (сигнал, указывающий на то, что МП выполняет операцию ввода/вывода) и уровнем лог. 1 адресных битов A9-A15 (поступающих на микросхему D35 и указывающих на то, что адрес ввода-вывода находится в пределах FE00H-FFFFH).

Микросхема D34 декодирует адресные биты A_0, A_3--A_8 и сигнал BHE/ для генерации сигнала выбора соответствующего порта ввода-вывода. Логический 0 на выходе дешифратора разрешает выбор соответствующего порта ввода/вывода.

На плате комплекта Микролаб установлены устройства, имитирующие внешние порты ввода-вывода. Это 8 светодиодов VD3-VD10, которые подключены к порту P1B и служат приемниками информации (порт вывода с адресом FFFAH) и 8 переключателей S4, соединенных с портом P2B, информация о состоянии которых может быть считана МП

(порт с адресом FFFB). Разряд B_0 порта P1B соединен с громкоговорителем, используемым в качестве приемника информации.

Дешифратор портов ввода-вывода обеспечивает выбор еще двух БИС КР580ВВ51А и КР580ВВ79. На рис. 16 приведен пример подключения БИС КР580ВВ51 к шине данных и периферийному устройству.

В Микролаб интерфейс последовательного ввода-вывода выполнен на БИС программируемого интерфейса связи в последовательном коде КР580ВВ51А (микросхема D43), которая работает в асинхронном режиме.

БИС КР580ВВ51А занимает в адресном пространстве ввода-вывода МП два порта ввода-вывода. Так как она соединена с младшим байтом данных (D0-D7), то оба порта имеют четные адреса FFF0H и FFF2H. Функционирование микросхемы определяется адресным битом А0 и сигналами IORC и IOWC (см. табл. 12).

Генератор скорости передачи, выполненный на двух двоично-десятичных счетчиках (микросхемы D52, D53), использует сигнал PCLK для формирования различных скоростей передачи данных. Для выбора скорости передачи необходимо установить соответствующий контакт переключателя S17 в положение «замкнуто», а остальные контакты должны быть при этом разомкнуты. Соответствие контактов переключателя S17 скоростям передачи представлено в табл. 13.

В Микролаб 1810 БИС КР580ВВ79 отвечает за сканирование клавиш, кодирование матрицы клавиатуры и обновление элементов индикации.

Контроллер клавиатуры и индикации занимает два порта ввода/вывода в адресном пространстве ввода-вывода МП. БИС КР580ВВ79 сопрягаются с младшим байтом шины данных (D0-D7), поэтому оба порта имеют четные адреса FFE8 и FFEA. Функционирование БИС определяется адресным битом А1 и сигналами IORC, IOWC (табл. 14).

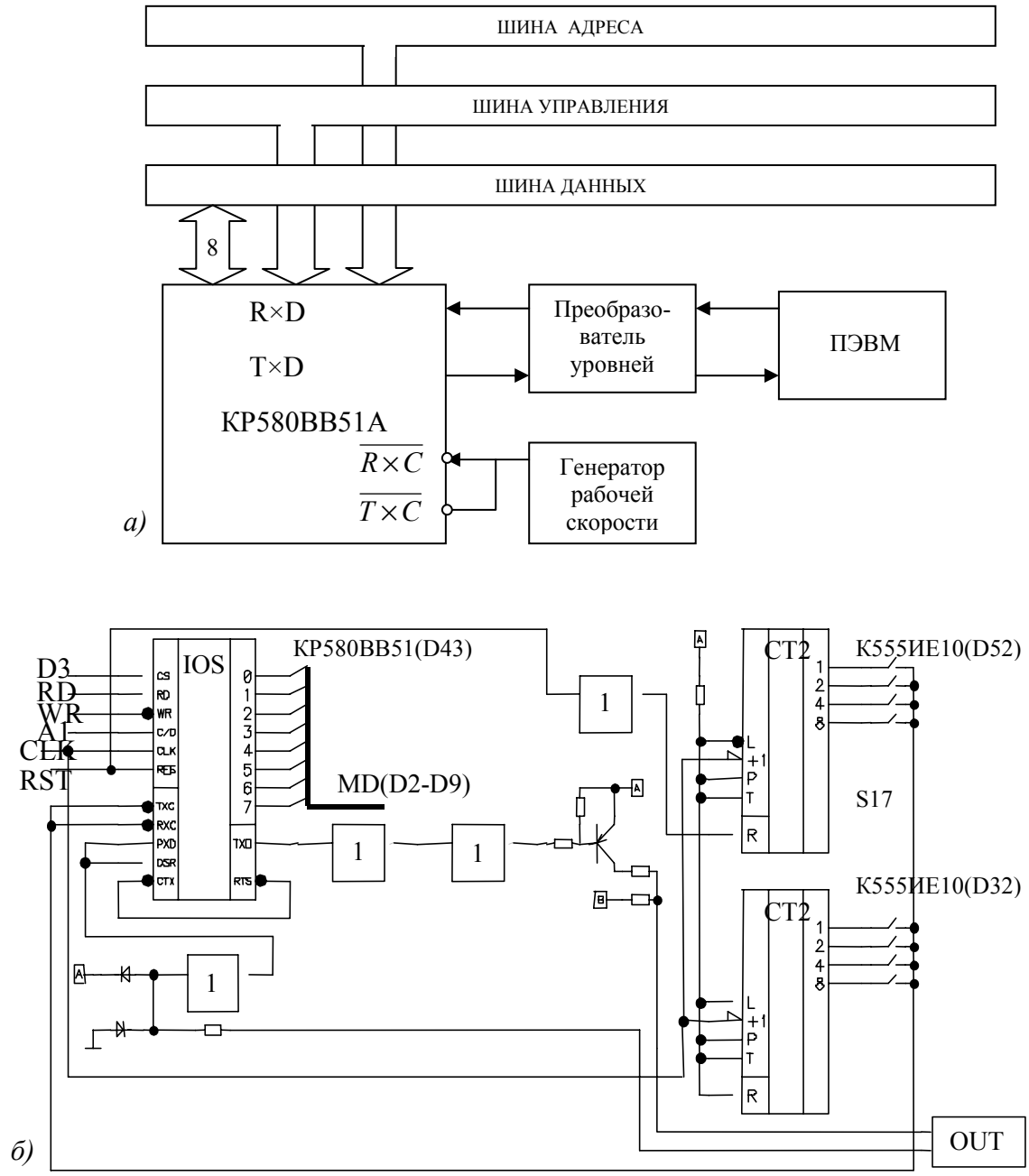


Рис. 17. Подключения последовательного программируемого интерфейса в комплекте Микролаб: *а)* --структурная схема, *б)* --принципиальная электрическая схема

таблица 12

Входы БИС КР580ВВ51А			Адрес порта	Действие
C/D	RD	WR		
0	0	1	FFFOH	Чтение данных
0	1	0	FFFOH	Запись данных
1	0	1	FFF2H	Чтение состояния
1	1	0	FFF2H	Запись команды

таблица 13

Скорость передачи, бод	Контакт переключателя

19 200	SA8
9600	SA7
4800	SA6
2400	SA5
1200	SA4
600	SA3
300	SA2
150	SA1

таблица 14

Входы БИС КР580ВВ79			Адрес порта	Действие
A_0	RD/	WR/		
0	0	1	FFE8H	Чтение ОЗУ дисплея или клавиатуры
0	1	0	FFE8H	Запись в ОЗУ дисплея
1	0	1	FFEAH	Чтение состояния
1	1	0	FFEAH	Запись команды

Программа монитора программирует БИС КР580ВВ79 следующим образом:

- восемь 8-разрядных индикаторов с вводом слева и кодированное сканирование клавиатуры с обнаружением 2-клавишных сцеплений (команда установки режимов работы клавиатуры-индикатора, код ООН);
- коэффициент деления К равен 25 для обеспечения времени сканирования клавиатуры индикаторов 5 мс и времени устранения дребезга контактов 10 мс (команда программирования синхронизации, код 39Н).

2.3. Структура лабораторного комплекса

Микролаб КМ1810ВМ86 910 может работать как автономно так и в составе лабораторного комплекса, приведенного на рис. 18.

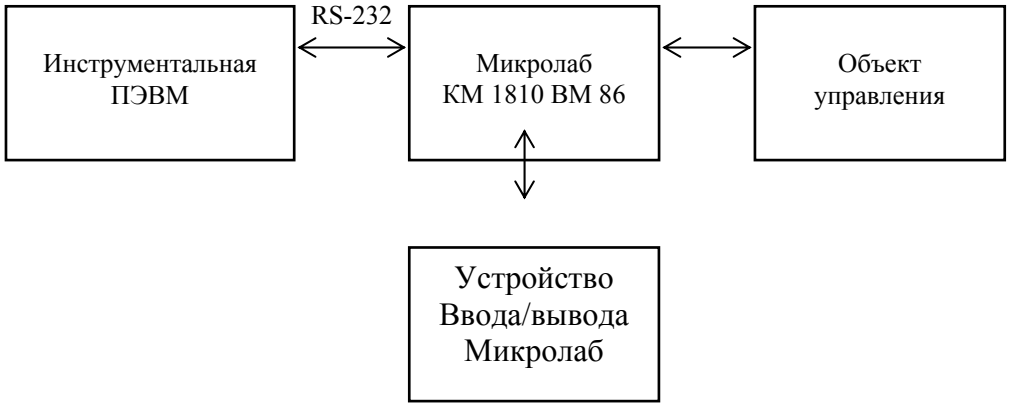


Рис. 18. Лабораторный комплекс на базе Микролаб КМ 1810 ВМ 86 910

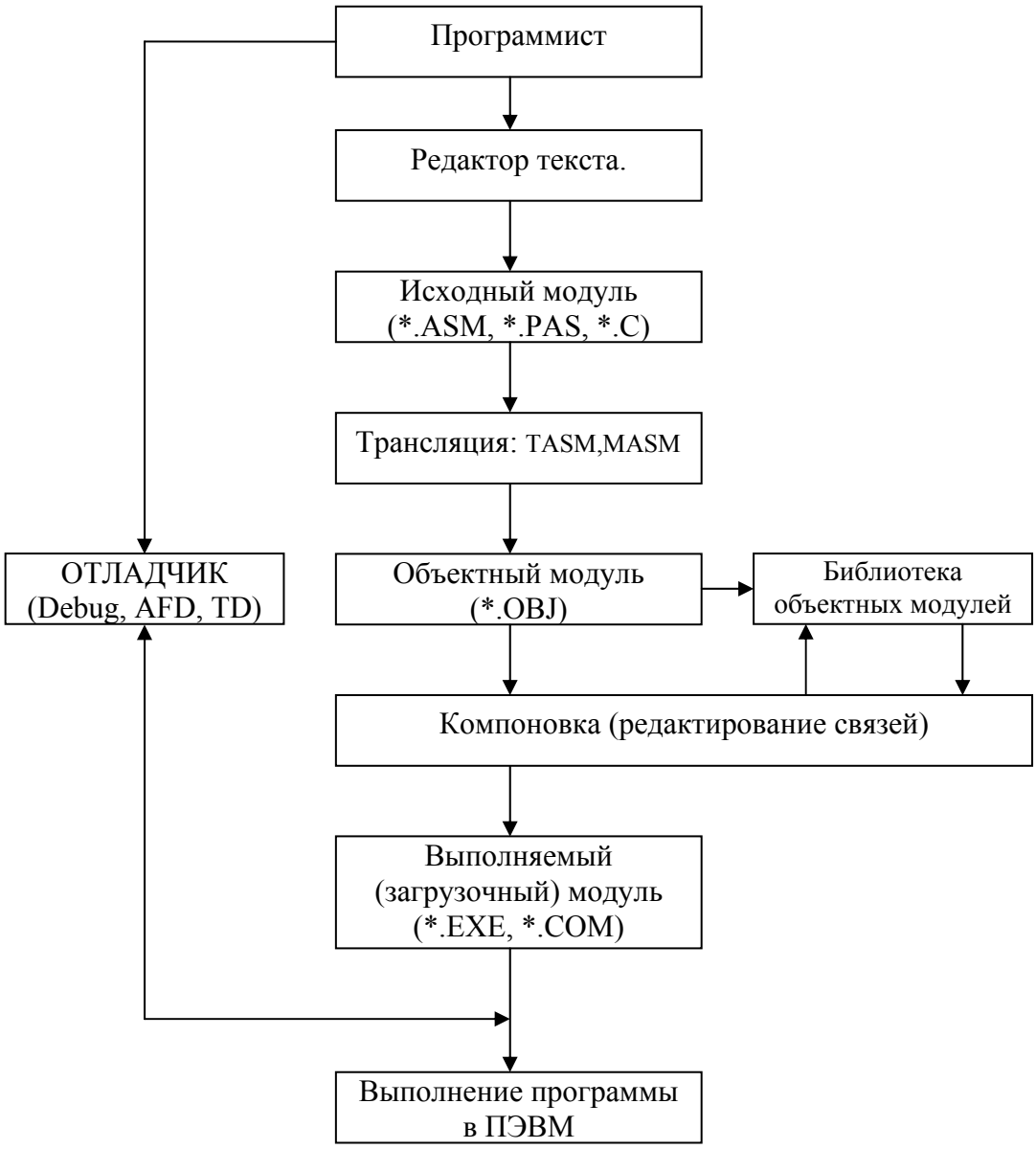
3. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЛАБОРАТОРНОГО ПРАКТИКУМА

Прохождение задач пользователя на инструментальной ПЭВМ поясняет рис. 19.

3.1. Трансляция и компоновка программ

Текст программы создается с помощью одного из простых текстовых редакторов, например БЛОКНОТ или встроенного редактора в NORTON COMANDER и сохраняется с расширение ***.asm**.

Далее выполняется трансляция программы с помощью одного из трансляторов **masm.exe** или **tasm.exe**. На этом этапе формируется объектный модуль, который представляет собой программу в машинных кодах с относительной адресацией памяти. Например, если наша



программа называется **prg.asm**, тогда командная строка будет иметь следующий вид

Рис. 19. Прохождение задач пользователя на ПЭВМ

C:>tasm prg.asm

Если не обнаружено ошибок, на диске создастся новый объектный файл с расширением *.OBJ, например **prg.obj.** Или же выводится сообщения об ошибке, причем указывается номер строки, где она произошла. Такой способ локализации ошибок не очень удобен, поэтому рекомендуется осуществлять трансляцию с ключем L создавая файл листинга **prg. lst**. Файл листинга необходимо создавать и том случае, если мы хотим получить информации о машинных кодах программы для последующего автономного программирования Микролаба .

В общем случае полный формат командной строки следующий:

TASM [опции] <имя *.asm файла> [, имя *.obj файла] [, файл листинга *.lst] [, файл перекрестных ссылок *.crf].

Можно задавать только имя исходного файла. В этом случае имена объектного файла, файла листинга и файла перекрестных ссылок будут совпадать с именем исходного файла (наиболее типичный случай), и достаточно поставить запятые вместо имен этих файлов:

C:>tasm.exe prg.asm , , ,

Создадутся файлы: **prg.obj , prg.lst , prg.crf .**

Файл ***.lst** позволяет более наглядно анализировать ошибки в программе. К тому же он также содержит полезную отладочную информацию и машинные код инструкций.

Файл ***.crf** целесообразно формировать в только в больших многомодульных программах.

Необязательный аргумент «опции» позволяет задать режим работы транслятора. Назначение опций можно посмотреть, запустив просто **tasm.exe**

Таким образом ,запуск программы на трансляцию может выглядеть так

C:>tasm.exe /zi prg.asm , , ,

При компоновке программы компоновщиком **tlink.exe**. из объектного модуля (модулей) получается загрузочный модуль. Для этого используется следующий формат командной строки:

tlink.exe prg.obj

В результате получим файл **prg.exe**.

Полный формат команды запуска транслятора следующий:

Tlink.exe [опции] <список объектных файлов>.

3.2 Запись программы пользователя в ОЗУ Микролаб

Структура карты памяти Микролаб описана в разделе 2.

Задачи пользователя разрабатываются с помощью инструментальной ПЭВМ и выполняться на Микролаб (см. рис.18).

Для пересылки программ пользователя в ОЗУ Микролаб и чтения ОЗУ Микролаба, используется программа SLINK.EXE, которая имеет дружественен и понятный интерфейс. Пересылаемый файл должен быть приведен к виду *.bin . Если после процедуры прохождения задачи на ПЭВМ у вас имеется файл вида *.com , то для преобразования к формату *.bin его достаточно переименовать. Если после процедуры прохождения задачи на ПЭВМ у вас имеется файл вида *.exe , то для преобразования к формату *.bin его достаточно обработать процедурой exe2bin и на выходе будет сформирован нужный файл файл *.bin

3.3. Описание особенностей программирования COM-порта под ОС Windows NT, 2000, XP

Программа SLink.EXE обеспечивает взаимодействие Микролаб инструментальной ПЭВМ посредством интерфейса RS-232. Версии операционной системы Windows NT, 2000, XP не допускают прямого обращения к портам ПК, поэтому данная версия Slink при работе с COM портом использует API-функции Windows.

Наиболее просто к API-функциям можно обращаться при использовании языка C/C++. Предпочтение было отдано последнему, как объектно-ориентированному языку. При написании данной версии Slink использовалась среда Microsoft Visual Studio 6.0 Enterprise Edition. Графически интерфейс приложения разрабатывался при помощи библиотеки MFC 6.0.

В ОС Windows работа с COM-портом ведется как с обычным файлом. Для открытия порта используется функция CreateFile. В нашем случае эта функция используется следующим образом:

```
m_hPort = CreateFile(strPort, GENERIC_READ|GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
```

Из всех параметров, передаваемых функции отметим первых три, так как использование остальных для работы с портами является стандартным. Первый параметр представляем собой имя открываемого порта (в Windows определены стандартные имена COM и LPT-портов), второй – режим открытия (здесь – для чтения/записи). Функция возвращает дескриптор порта, который в дальнейшем используемся другими API-функциями для работы с портом. В случае неудачи функция возвращает предопределенное значение с именем INVALID_HANDLE_VALUE.

Для настройки адаптера используется функция SetCommState(m_hPort, &dcb), где m_hPort – дескриптор порта, &dcb – адрес структуры, в которую заносятся настройки порта.
Использование данной структуры представлено ниже:

```
// Настройка параметров адаптера
dcb.DCBlength = sizeof (DCB);     // размер структуры
```

```
dcb.BaudRate = static_cast <DWORD> (uSpeed);     // установка скорости обмена
dcb.fBinary = TRUE;
dcb.fParity = FALSE;     // запрет проверки контроля четности
dcb.Parity   = NOPARITY;  // запрет контроля четности
dcb.fOutxCtsFlow = FALSE;  // запрещаем использование сигнала CTS
dcb.fOutxDsrFlow = FALSE;  // запрешаем использование сигнала DSR
dcb.fDtrControl = DTR_CONTROL_DISABLE;     // запрещаем использование сигнала DTR
dcb.fDsrSensitivity = FALSE;
dcb.fTXContinueOnXoff = FALSE;
dcb.fOutX = FALSE;
dcb.fInX = FALSE;
dcb.fErrorChar = FALSE;
dcb.fNull = FALSE;
dcb.fRtsControl = RTS_CONTROL_DISABLE;
dcb.fAbortOnError = FALSE;
dcb.wReserved = 0;
dcb.ByteSize = 8;     // длина слова
dcb.StopBits = TWOSTOPBITS;     // два стоп-бита
```

После проделанных действий для записи/чтения данных из порта и в порт используются функции WriteFile/ReadFile:

```
BOOL bFlag = WriteFile(m_hPort, &cByte, 1, &dwWritten, NULL);
BOOL bFlag = ReadFile(m_hPort, &cByte, 1, &dwRead, NULL);
```

Где &cByte – адрес памяти в которую/из которой производится запись/чтение. Следующий параметр – кол-во передаваемых байт. Далее – число реально переданных/считанных байт. В случае ошибки возвращается нулевое значение.

Закрывается порт при помощи функции CloseHandle(m_hPort), единственным параметром которой является дескриптор порта.

Программный протокол передачи данных

Для обеспечения надежности процесс обмена информацией с Микролаб строится по сценарию, в котором каждый шаг выполняется лишь после наступления определенных условий, то есть, в данном случае, передача (прием) очередного байта возможна лишь тогда, когда абонент на другом конце линии уведомит передатчик (приемник) о своей готовности. Это, во-первых, гарантирует, что каждый байт данных будет обработан соответствующим образом даже машинами и программами с разным быстродействием, а во-вторых, позволяет обмениваться не только данными, но и некими стандартными сообщениями, определенными в протоколе, например: «Байт принят и обработан, ожидаю следующего» или «Есть ли кто-нибудь на линии? Прошу ответить». Такой способ связи

называется запрос-ответным режимом или handshaking' ом (рукопожатие - англ.). Сам процесс обмена подобными сигналами и называется подтверждением, а сами сигналы - сигналами подтверждения или также подтверждениями. Аппаратным подтверждение является, когда сигналы подтверждения передаются по отдельным линиям, распознаются и обрабатываются адаптером. К слову, большинство линий стандарта RS-232 как раз и предназначены для аппаратного подтверждения. В случае программного подтверждения сигналы подтверждения передаются в общем потоке данных, распознаются и обрабатываются программой связи. Программное подтверждение, естественно, уменьшает (и существенно) скорость обмена информацией, но зато оно выручает в тех случаях, когда применение аппаратного подтверждения по каким-то причинам невозможно или нежелательно, к тому же программное подтверждение более гибкое. В рассматриваемом случае из-за особенностей Микролаба будет использоваться именно программное подтверждение.

Рассмотрим более подробно логику использованного в Slink протокола передачи данных. Логику передачи данных реализует функция Transfer, которая запускается при получении от пользователя подтверждения о передаче данных. Листинг функции приведен ниже:

```
BOOL CConnection::Transfer(BYTE *pData, USHORT uDataSize, USHORT uSegment, USHORT uOffset)
    {
    // устанавливаем соединение
    if(!SetConnection())
        return FALSE;
    // производим инициализацию устройства
    if(!PortOut(MODE_TRANSFER))  // устанавливаем режим связи
        return FALSE;

    BYTE bByte = static_cast <BYTE> (uOffset & 0x00FF);
    if(!PortOut(bByte))
        return FALSE;

    bByte = static_cast <BYTE> (uOffset >> 8);
    if(!PortOut(bByte))
        return FALSE;


    bByte = static_cast <BYTE> (uSegment & 0x00FF);
```

```
if(!PortOut(bByte))
        return FALSE;

bByte = static_cast <BYTE> (uSegment >> 8);
if(!PortOut(bByte))
        return FALSE;

bByte = static_cast <BYTE> (uDataSize & 0x00FF);
if(!PortOut(bByte))
        return FALSE;

bByte = static_cast <BYTE> (uDataSize >> 8);
if(!PortOut(bByte))
        return FALSE;

// передаем данные
for(int i = 0; i < uDataSize; i++)
{
        if(!PortOut(pData[i]))
                return FALSE;
}

return TRUE;
}
```

Прежде всего устанавливается соединение функцией SetConnection, которая посылает в устройство символ 0x82, зарезервированный в данном протоколе как символ-запрос. При получении ответа(символ-ответ 0x8A) функция возвращает логическое значение «истина», что является индикатором того, что устройство найдено и готово к передаче данных.

На следующем этапе производится инициализация устройства. При помощи функции PortOut в устройство побайтно передается режим передачи данных (0x0 – передача, 0x1 - прием), адрес памяти Микролаб по которому будут записаны (считаны в случае приема) данные (сначала - смещение, затем - сегмент) и размер передаваемых.

Далее следует побайтная пересылка данных.

Функция PortOut также реализует часть программного протокола. Байт считается успешно переданным, если устройство пришлет тот же байт в ответ. Тот же принцип реализован и для функции чтения байта

PortIn – после чтения байта программа посылает его же в устройство в качестве подтверждения.

3.4. Описание процедур ReadWord и пакета программ «Print»

3.4.1. Процедура ReadWord

Данная процедура предназначена для ввода с клавиатуры как массивов, так и отдельных беззнаковых двухбайтных чисел в ассемблерных .EXE-программах.

- файл READWORD.OBJ -- собственно процедура;
- файл READWORD.DOC -- описание процедуры;
- файлы TRYTOUSE.ASM и TRYTOUSE.EXE -- исходный текст примера и его оттранслированный вариант.

Текст примера находится в конце данного описания.

- AL -- количество чисел, которое требуется ввести;
- ES:DI -- сегмент: смещение вашего массива/переменной;
- В AL -- сколько элементов требовалось ввести;
- в AH -- сколько элементов фактически введено
- флаг переноса (CF) сброшен -- ввод нормально завершен
- флаг переноса установлен -- ввод прерван пользователем

1. Запрашиваемое для ввода количество элементов не может быть больше 255.
2. Процедура не предназначена для использования в COM-программах.
3. Процедуру следует объявить внешней и использующей дальний тип вызовов (FAR).
4. Флаг направления (DF) рекомендуется сбрасывать для совместимости с последующими версиями.
5. Кое-что о работе с регистрами. Вообще-то считается хорошим тоном сохранять задействованные в вашей подпрограмме регистры, через которые не передаются параметры. Но есть и «неаккуратные» процедуры, так что будьте осторожны и при вызовах «подозрительных» процедур все-таки сохраняйте ваши регистры -- так надежнее. Хотя в данной версии это не обязательно, рекомендую советом не пренебрегать -- мало ли что придет мне в голову при написании следующей версии.
6. Поскольку процедура использует прерывания BIOS int 10h и int 16h, ввод-вывод не может быть перенаправлен.
7. В данной версии не предусмотрена работа в графических режимах, так что если вы вызовете ее в графическом режиме, результат будет непредсказуем. Для совместимости с последующими версиями рекомендуется обнулять регистр BX.

8. Рекомендуется использовать малую модель памяти (SMALL). Впрочем, вряд ли вам понадобится большая.
9. Процедура транслировалась с использованием Turbo Assembler 3.0, но, по-видимому, нет препятствий к тому, чтобы использовать ее с более ранними версиями TASM, кроме, быть может, 1.x.
10. Линковка: tlink .exe prg.obj readword.obj

При вызове процедуры на дисплей выводится приглашение «?_». В ответ -- либо набор числа (допустимы символы '0'--'9','A'--'F', 'H','O' и их строчные варианты, ввод преобразовывается к верхнему регистру, например 'a' - к 'A'), либо нажатие клавиши редактирования. Буква, задающая систему счисления ('H'-шестнадцатеричная, 'O'-восьмеричная), должна быть последней. При невыполнении этого, а также при наборе недопустимого символа или при переполнении внутреннего стека символов, при невыполнении правил редактирования процедура подаст звуковой сигнал.

Допустимы следующие клавиши редактирования:
[Enter] -- ввести набранное число, звуковой сигнал -- если вы нажимаете [Enter], не имея в строке никаких цифр;
[ESC] или [Ctrl]-[C] -- немедленно прервать ввод;
[BackSpace] -- затереть символ слева от курсора, звуковой сигнал -- если нечего стирать.

В случае, если набранное число превышает 65535 (FFFFh), процедура либо запрещает его ввод и подает звуковой сигнал, либо просит перевести это число.

Процедура определяет в процессе набора присутствие десятичных и шестнадцатеричных цифр и соответственно устанавливает основание системы счисления. Так, если вы набрали, например, 10A или D15, то символ 'H' можно опустить. Если вы хотите установить восьмеричную СС, но в строке присутствует хотя бы одна цифра старшей СС, ввод символа 'O' будет запрещен и процедура подаст звуковой сигнал. По умолчанию, если отсутствуют цифры шестнадцатеричной СС, устанавливается десятичная СС.

Пример программы:

```
.MODEL SMALL                    ;малая модель памяти

EXTRN   ReadWord:FAR            ;наша процедура внешняя и использует
                                ;дальний тип вызовов

.STACK 64                       ;под стек отводим 64 байта

.DATA                           ;определяем сегмет данных

MyData  DW 8 DUP (00h)          ;массив из 8 слов
                                ;сюда будем отправлять введенные данные

.CODE                           ;сегмент кода
```

```
TryToUse      PROC    FAR              ;головная процедура

push   DS                              ;стандартные действия для EXE-файла
xor    AX,AX                           ;просто повторяйте их и не заботьтесь
push   AX                              ;о смысле

mov    AX,SEG MyData                   ;инициализируем DS
mov    DS,AX                           ;сразу в DS и ES - нельзя
mov    ES,AX                           ;и ES - в каком сегменте MyData ?
mov    AX,OFFSET MyData                ;где в сегменте у нас MyData ?
mov    DI,AX                           ;сразу в DI - нельзя
cld                                    ;сбрасываем флаг направления
xor    AX,AX                           ;обнуляем AX
xor    BX,BX                           ;обнуляем BX
mov    AL,8                            ;вводить 8 элементов
push   DX                              ;сохраняем регистры
push   CX
push   SI
push   DS

call   ReadWord                        ;и вызываем нашу процедуру


pop    DS                              ;восстанавливаем регистры
pop    SI
pop    CX
pop    DX

jc     Bye                             ;перейти, если пользователь прервал ввод
                                       ;здесь мы что-то делаем с введенными данными и т.д.


Bye:   RETF                            ;до свидания

TryToUse      ENDP                     ;конец головной процедуры

END                                    ;конец программы
```

3.4.2. Пакет программ «Print»

Пакет программ «Print» предназначен для вывода на экран чисел и двухмерного массива. В его состав включено шесть макрокоманд:

init_ds -- инициализация сегмента данных;

exit -- стандартный выход из программы;

out_spase -- вывод пробела;

out_enter -- переход на новую строку;

resultat_per *a* -- выводит на экран значение переменной *a*;

resultat_mas *a, m, n* -- выводит на экран массив *a* размерами m×n.

Внимание! Макрокоманда resultat_mas содержит метки m_1, m_2, m_3, m_4 по этому недопустимо их использование в других частях программы и повторное использование макрокоманды resultat_mas. Для подключения библиотек макрокоманд и функций в начале программы необходимы строки:

```
include printmcr.inc
extrn      is_minus:far
```

При компоновке основной программы объектный файл printprc.obj присоединяется с помощью командной строки:

tlink.exe prg.obj printprc.obj

Сегмент кода программы должен начинаться макрокомандой init_ds, а заканчиваться exit. Макрокоманды изменяют регистры ax, bx, cx, dx, si, di, bp, ip, flags.

Пакет программ «Print» содержит файлы:

Readme.doc -- данная документация;

Test.asm -- пример использования пакета программ «Print»;

Test.bat -- приложение, которое автоматически компилирует и линкует файл test.asm;

Printmcr.inc -- библиотека макрокоманд;

Printprc.obj -- библиотека функций;

Printprc.asm -- исходный файл библиотеки функций.

Текст файла printmcr.inc

```
init_ds     macro                   ;макрокоманда инициализации сегмента данных
  mov       ax,data
  mov       ds,ax
  xor       ax,ax
endm
exit        macro                   ;макрокоманда завершения работы программы
  mov       ax,4c00h
  int       21h
endm
delenie             macro           ;макрокоманда, которая разбивает регистр al(8 разрядов)
  div       bl                      ;на два числа по четыре разряда в каждом и переводит их в
  rol       ax,8                    ;упакованный двоично-десятичный формат
  push      ax
        aam
  shl       al,4
  shr       ax,4
  add       al,dl
  adc       dh,0
        daa
  mov       dl,al
  pop       ax
  adc       dh,0
  rol       ax,8
        aam
  shl       al,4
  shr       ax,4
  add       al,dh
        daa
        mov         dh,al
endm
hex_to_bsd          macro   n       ;макрокоманда переводит двоичное число в
  xor       dx,dx                   ;упакованный двоично-десятичный формат
  mov       bl,64h
  mov       ax,n
  and       ax,000fh
        daa
  mov       dl,al
  mov       ax,n
  and       ax,00f0h
  div       bl
  rol       ax,8
  push      ax
        aam
```

```
  shl       al,4
  shr       ax,4
  add       al,dl
        daa
  mov       dl,al
  pop       ax
  adc       dh,ah
  mov       ax,n
  and       ax,0f00h
        delenie
  mov       ax,n
  and       ax,0f000h
        delenie
        mov         n,dx
endm
out_one             macro           ;макрокоманда выводит на экран одну цифру
  shr       dx,12
  add       dx,30h
  int       21h
endm
out_bsd             macro   n       ;макрокоманда выводит на экран
  mov       ax,0200h                ;упакованное двоично-десятичное число
  mov       dx,n
        out_one
  mov       dx,n
  shl       dx,4
        out_one
  mov       dx,n
  shl       dx,8
        out_one
  mov       dx,n
  shl       dx,12
out_one
endm
out_enter   macro                   ;макрокоманда переносит курсор в начало следующей строки
  mov       dx,0ah
  mov       ax,0200h
  int       21h
  mov       dx,0dh
  int       21h
endm
out_spase   macro                   ;макрокоманда выводит на экран пробел
  mov       dx,0h
  mov       ax,0200h
  int       21h
endm
resultat_mas        macro   num,m,n     ;макрокоманда выводит на экран массив чисел
  xor       si,si                       ;размерностью m×n
  xor       cx,cx
  mov       cl,m
m1:         push    cx
  mov       cl,n
m2:         mov     bx,num[si]
  call      is_minus
  mov       bp,bx
  hex_to_bsd        bp
  out_bsd   bp
        out_spase
  inc       si
  inc       si
  dec       cx
  jz        m3
  jmp       m2
```

```
m3:     out_enter
  pop   cx
  dec   cx
  jz    m4
  jmp   m1
m4:
endm
resultat_per    macro   n       ;макрокоманда выводит на экран число
  mov   bx,n
  call  is_minus
  mov   bp,bx
  hex_to_bsd    bp
  out_bsd       bp
  out_enter
endm
```

Текст файла printprc.inc

```
    public is_minus
stk     segment stack
  db    1h dup("?")
stk     ends
code    segment para    public "code"
  assume cs:code,ss:stk
is_minus proc   far             ;процедура определяет знак числа в регистре bx
  test  bx,8000h                ;и если он отрицательный, то выводит на экран «-»,
  jz    metka                   ;отнимает единицу и инвертирует число
  mov   dx,45
  dec   bx
  not   bx
  jmp   konec
metka:  mov     dx,0
konec:  mov     ax,0200h
  int   21h
ret
is_minus endp
code    ends
end is_minus
```

Текст файла test.asm

(Пример использования макрокоманд в программах)

```
include printmcr.inc                                    ;обязательная строка
extrn   is_minus:far                                    ;обязательная строка
data    segment para    public  «data»
  num   dw      12      dup(0ffffh)     ;формирование массива чисел (--1 в дополнительном
коде)
  m     db      4                       ;размеры сформированного
  n     db      3                       ;массива
  per   dw      64              ;число для вывода на экран с помощью макрокоманды
resultat_per
data    ends
stk     segment stack
  db    100h dup(«?»)
stk     ends
code    segment para    public «code»
  assume cs:code,ds:data,ss:stk
start   proc
  init_ds                       ;инициализация  сегмента данных
                                ;некоторые действия для ввода и обработки данных
  resultat_mas  num,m,n         ;макрокоманда вывода получившегося массива на экран
  mov   bp,50h
  resultat_per  per             ;макрокоманда вывода числа на экран
```

```
    exit                        ;макрокоманда стандартного завершения работы
start       endp
    code    ends
    end     start
```

3.5. Программирование на ассемблере в среде Windows

В операционной системе Windows программировать отчасти проще, чем под MS-DOS. Достаточно освоить алгоритм создания окон и их управляющих и информационных элементов, разобраться в механизме обработки сообщений. Начнем с нескольких общих положений в программирование в Windows.

Программирование в Windows основывается на использовании API функций API (Application Program Interface).. Их количество достигает двух тысяч. Все взаимодействие с внешними устройствами и ресурсами операционной системы будет происходить посредством API функций

Windows поддерживает **два типа приложений:**

Графическое (оконное) приложение строится на базе набора API функций, составляющий графический интерфейс пользователя GUI (Graphic User Interface). Такое приложение представляет собой программу, которая весь вывод на экран производит в графическом виде. Первым результатом работы оконного приложения является отображение на экране окна.

Консольное приложение представляет собой программу, работающую в текстовом режиме. Работа такого приложения напоминает режим MS-DOS. Работа консольного приложения также обеспечивается API функциями.

Операционная система Windows использует линейную адресацию памяти. Другими словами, всю память можно рассматривать как один сегмент. Это означает, что адрес любой ячейки памяти будет определяться содержимым одного 32-битного регистра, например EBX.

Програмист пишущий программу теперь фактически не ограничены объемом данных, кода или стека (объеме локальных переменных). Сегменты в тексте программы играют теперь другую роль. Они позволяют задать отдельным фрагментам кода (секциям) определенные свойства: запрет на запись, общий доступ и т. д.

Операционная система Windows -- многозадачная среда. Каждая задача имеет свое адресное пространство и свою очередь сообщений. Более того, даже в рамках одной программы может быть осуществлена многозадачность -- любая процедура может быть запущена как самостоятельная задача.

Главным элементом графического приложения в среде Windows является окно. Для каждого окна определяется своя процедура обработки сообщений.

Окно может содержать элементы управления: кнопки, списки, окна редактирования и др. Эти элементы также являются окнами, но обладают

особыми свойствами. События, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в процедуру окна.

Начнем знакомство с программированием в Windows с описания того, как вызываются функции API. Обратимся к файлу помощи WIN32.HLP, который поставляется, например, с пакетом Borland C++, и выберем любую API функцию, например MessageBox:

```
int MessageBox(HWND hWnd, LPCTSTR IpText, LPCTSTR IpCaption, UINT uType);
```

Данная функция выводит на экран окно с сообщением и кнопкой (или кнопками) выхода. Смысл параметров: hwnd — дескриптор окна, в котором будет появляться окно-сообщение; IpText -- текст, который будет появляться в окне; IpCaption -- текст в заголовке окна; uType -- тип окна, в частности можно определить количество кнопок выхода. Теперь о типах параметров. Все они -- 32-битные целые числа: HWND -- 32-битное целое, LPCTSTR -- 32-битный указатель на строку, UINT -- 32-битное целое. По причине, о которой будет сказано ниже, к имени функций нам придется добавлять суффикс «A», кроме того, при использовании MASM необходимо также в конце имени добавить @16. Таким образом, вызов указанной функции будет выглядеть так: CALL MessageBoxA@16. А как же быть с параметрами? Их следует предварительно поместить в стек командами PUSH. Запомним правило: помещать параметры в стек следует начиная с правого параметра . Итак, пусть дескриптор окна расположен по адресу HW, строки -- по адресам STRI и STR2, а тип окна сообщения -- это константа. Самый простой тип имеет значение 0 и называется MB_OK. Тогда:

```
MB_OK           equ 0
STR1            DB "Неверный ввод! ",0
STR2            DB "Сообщение об ошибке.",0
HW              DWORD ?

PUSH            MB_OK
PUSH            OFFSET STR1
PUSH            OFFSET STR2
PUSH            HW
CALL            MessageBoxA@16
```

Как видим, все так же просто, как если бы мы вызывали эту функцию на Си или Delphi. Результат выполнения любой функции -- целое число, которое возвращается в регистре EAX.

Теперь выясним смысл дополнительных символов в имени API функции. Все функции, работающие со строками, существуют в двух вариантах. Если строка рассматривается как набор символов ASCII, то к имени функции добавляется A (MessageBoxA). Другой вариант функции, использующий строки в формате UNICODE (два байта на символ), заканчивается буквой U (MessageBoxU). В наших программах будут использоваться строки только в формате ASCII. Достаточно заменить

букву A на U, и можно эти же программы заставить работать с UNICODE. Имена всех API функций в win32 модифицируются так, что перед именем функции ставится знак подчеркивание «_», а поле «@» и количество байтов, получаемых функцией через стек (компилятор TASM использует другой формат вызова функций). Очистка стека от полученных параметров целиком ложится на вызванную функцию. При компиляции MASM32 автоматически добавляет знак подчеркивание «_» в начало вызываемых функций, поэтому в тексте программы его добавлять не требуется.

Для начала рассмотрим консольное приложение (текст программы HelloWorldConsole.asm приведен ниже). Консольные приложения несколько отличаются от классических оконных, здесь нет необходимости самостоятельно создавать окно и обрабатывать сообщения. Консоль предназначена для ввода и вывода информации в текстовом режиме. Кроме текстов самих файлов программы, нам потребуются ключи компиляции. Для консольного приложения это:

```
ml.exe /c /coff Prog.asm
link.exe Prog.obj /subsystem:console
```

Примечание: Приведенные ниже примеры написаны с ориентацией на то, что пользователи при трансляции и линковке будут использовать пакет MASM версии 6.13 и выше.

Текст программы HelloWorldConsole.asm

```
.386p                                          ;тип процессора: 80386 и выше
.model flat,stdcall                            ;плоская модель памяти
include HelloWorldConsol.inc                   ;сюда включаем текст файла HelloWorldConsol.inc
_DATA segment dword public use32 'data'        ;начало сегмента данных
  HelloMessage  db "Hello World!!!",0Ah,0Dh,0  ;текст сообщения
  Handle  dd    ?                              ;дескриптор приложения
  Len     dd    ?
  Bufer   dd    ?
_DATA ends                                     ;конец сегмента данных
_TEXT segment dword public use32 'code'        ;начало сегмента кода
start:                                         ;обязательная метка
  push    SDT_OUTPUT_HANDLE                    ;получаем в eax
  call    GetStdHandle@4                       ;HANDLE для вывода
  mov     Handle,eax                           ;и сохраняем его
;--------------------------------------------------------------------------------
  push    offset  Bufer                        ;буфер Вывода
  push    offset  Len                          ;кол-во выведенных символов
  push    10h                                  ;кол-во символов для вывода
  push    offset  HelloMessage                 ;текст сообщения
  push    Handle                               ;handle вывода
  call    WriteConsoleA@20                     ;вывод текста на экран
;--------------------------------------------------------------------------------
  mov             ecx,0ffffffh                 ;величина паузы
Pause:
  loop    Pause                                ;пустой цикл для создания паузы
;--------------------------------------------------------------------------------
  push    0                                    ;константа для стандартного выхода
  call    ExitProcess@4                        ;завершение работы программы
_TEXT ends                                     ;конец сегмента кода
end start                                      ;конец программы
```

Текст файла HelloWorldConsole.inc

```
includelib c:\masm32\include\kernel32.lib        ;библиотеки
includelib c:\masm32\include\user32.lib          ;API функций
extrn          GetStdHandle@4:near               ;получение HANDLE для вывода
extrn          WriteConsoleA@20:near             ;вывод информации на экран
extrn          ExitProcess@4:near                ;завершение работы
SDT_OUTPUT_HANDLE equ -11                        ;константа для получения HANDLE для вывода
```

Программа HelloWorldConsole выводит на экран сообщение «Hello World!!!». Для этого нам потребуются ресурсы, описанные в файле HelloWorldConsole.inc. В программе используются API функции из двух библиотек: kernel32.lib и user32.lib и константа SDT_OUTPUT_HANDLE, предназначенная для получения идентификатора выходного буфера консольного приложения.

Так как Windows многозадачная операционная система, то каждому приложению ставится в соответствие некоторое число – Handle, что переводится как идентификатор или дескриптор, – для однозначного определения процесса. Консольное приложение не способно работать с другими объектами, кроме консоли, и идентификатор самого приложения нам не потребуется, однако для вывода текста нам потребовался идентификатор выходного буфера консоли, который мы получим при помощи вызова API функции GetStdHandle@4. Тип полученного идентификатора определяется константой, записанной в стек до вызова функции.

Вся работа программы заключается в вызове соответствующих API функций и создании задержки для визуализации результата работы программы. Алгоритм работы программы понятен из комментариев в листинге программы, и дублировать его здесь нет смысла. Единственное, на что следует обратить внимание – это метка *start*, с нее начинается выполнение Windows приложений, и она обязательно должна присутствовать в каждой программе, написанной для работы под Windows (метка может быть любая, но она должна присутствовать в начале и в конце сегмента кода); и директива *.model flat,stdcall*, которая назначает использование плоской модели памяти и стандартного вызова процедур, обязательных для использования Windows приложениями.

Теперь рассмотрим графическое (оконное) приложение HelloWorldMesBox. Результатом его работы является окно-сообщение с текстом «Hello World!!!». Для компиляции оконных приложений используются несколько иные ключи:

```
ml.exe /c /coff Prog.asm
link.exe Prog.obj /subsystem:windows
```

Текст программы HelloWorldMesBox.asm.

```
.386p                                                   ;тип процессора: 80386 и выше
.model flat,stdcall                                     ;плоская модель памяти
include HelloWorldMesBox.inc                ;сюда включаем текст файла HelloWorldMesBox.inc
_DATA segment dword public use32 'data'                 ;сегмент данных
  HelloMessage      db "Hello World!!!",0               ;текст сообщения
  HelloWorldTitle db "Hello world message box",0        ;заголовок окна
_DATA ends                                              ;конец сегмента данных
_TEXT segment dword public use32 'code'                 ;сегмент кода
start:                                                  ;обязательная метка
  push      MB_ICONINFORMATION                          ;стиль окна
  push      offset    HelloWorldTitle                   ;заголовок окна
  push      offset    HelloMessage                      ;текст сообщения
  push      0                                           ;идентификатор предка
  call      MessageBoxA@16                            ;создание окна с сообщением
;------------------------------------------------------------------------------
  push      0                                           ;код выхода
  call      ExitProcess@4                             ;завершение работы
_TEXT ends                                              ;конец сегмента кода
end         start                                       ;конец программы
```

Текст включаемого файла HelloWorldMesBox.inc

```
includelib c:\masm32\include\kernel32.lib               ;библиотеки
includelib c:\masm32\include\user32.lib                 ;API функций
extrn               ExitProcess@4:near                  ;завершение работы
extrn               MessageBoxA@16:near                 ;создание окна с сообщением
MB_ICONINFORMATIONequ       40h                         ;константа для определения стиля окна
```

Эта программа даже проще, чем вывод сообщения в консоль, так как не требует определения никаких идентификаторов. Для создания окна сообщения нам потребовались API функция MessageBox, стиль информационного окна, тексты заголовка и сообщения и идентификатор предка. Особый интерес представляет параметр идентификатор предка. Дело в том, что между окнами может существовать иерархическая зависимость «родитель – потомок», и родительское окно становится не доступным, пока не будут закрыты все дочерние окна. Для определения, какое же окно является родительским по отношению к этому окну, и существует параметр идентификатор предка, куда заносится идентификатор родительского окна. В нашем случае такое окно отсутствует, по этому в стек заносится 0, обозначающий отсутствие предка.

Переходим к рассмотрению классической программы под Windows. В такой программе имеется главное окно, а следовательно, и процедура главного окна, реагирующая на сообщения. В коде программы можно выделить следующие разделы:

- регистрация класса окна;
- создание главного окна;
- цикл обработки очереди сообщений;
- процедура главного окна.

Регистрация класса окна осуществляется с помощью функции RegisterClassA, единственный параметр которой -- адрес структуры WNDCLASS, содержащий информацию об окне. После регистрации класса окна нам необходимо при помощи API функции CreateWindowA создать экземпляр окна, а затем отобразить его на экране. Обработка очереди сообщений представляет собой цикл из трех функций: GetMessageA – получение следующего сообщения из очереди сообщений; TranslateMessage – разрешить использование клавиатуры путем трансляции сообщений о виртуальных клавишах в соответствующие сообщения об алфавитно-цифровых клавишах; DispatchMessageA – передача управления на процедуру главного окна, повторяющийся до тех пор, пока не придет сообщение о завершении работы программы. Процедура главного окна должна как минимум содержать вызов API функции DefWindowProcA для тех сообщений, которые не обрабатываются в функции окна. Также она должна убрать из стека 16 байт, которые ей передаются.

Текст программы ProgWinMain.asm

```
        .386p                                           ;модель процессора
        .model      flat,stdcall                        ;модель памяти
        option      casemap:none
        include ProgWinMain.Inc                         ;добавляем включаемый файл
ProgWinMain.Inc
        .data                                           ;сегмент данных
          NewHandle       dd      0                     ;идентификатор окна
          Handle  dd      0                             ;идентификатор приложения
          Msg             MSGSTRUC    <?>               ;переменная для хранения сообщения
          WndClass        WNDCLASS    <?>               ;переменная для хранения класса окна
          TitleName       db      'Main Window',0       ;тексты заглавий
          ClassName       db      'CLASS32',0
          Cap             db      'Message!',0
          MessagePQ       db      'Нажате этой кнопки привело к выходу',0       ;тексты сообщений
          MessageLB       db      'Вы нажали левую кнопку мыши!',0
        .code                                           ;сегмент кода
        start:                                          ;обязательная метка
          push            0
          call            GetModuleHandleA@4            ; получение дескриптора приложения
          mov             Handle,eax                    ;сохранение идентификатора в переменную
        ;-------------------------------------------------------------------------
        ;заполнение структуры класса окна и его регистрация
          mov             [WndClass.CLSSTYLE],style             ;стиль окна
          mov             [WndClass.CLWNDPROC],offset  WndProc  ;адрес процедуры окна
          mov             [WndClass.CLSCEXTRA],0                ;два параметра для дополнительной
информации
          mov             [WndClass.CLWNDEXTRA],0
          mov             eax,Handle                            ;дескриптор приложения заносится
          mov             [WndClass.CLSHINSTANCE],eax ;в структуру через eax
          push            IDI_APPLICATION                       ;загружаем пиктограмму окна
          push            0
          call            LoadIconA@8
          mov             [WndClass.CLSHICON],eax               ;и заносим ее в структуру
          push            IDC_CROSS                             ;загружаем изображение курсора,
          push            0                                     ;которое будет отображаться,
          call            LoadCursorA@8                         ;когда мышка наведена на окно
          mov             [WndClass.CLSHCURSOR],eax             ;сохраняем изображение курсора
```

```
  mov     [WndClass.CLBKGROUND],17          ;цвет фона окна
  mov     dword ptr [WndClass.CLMENUNAME],0 ;имя меню (здесь меню нету)
  mov     dword ptr [WndClass.CLNAME],offset ClassName      ;имя класса окна
  push    offset  WndClass                  ;сохранение в стек адреса структуры
  call    RegisterClassA@4                  ;регистрация класса окна
;-------------------------------------------------------------------------------
;создание окна
  push    0
  push    Handle                            ;дескриптор приложения
  push    0
  push    0
  push    400                               ;высота окна
  push    400                               ;ширина окна
  push    100                               ;y координата левого верхнего угла
окна
  push    100                               ;x координата левого верхнего угла
окна
  push    WS_OVERLAPPEDWINDOW               ;свойства окна (перекрывающееся)
  push    offset  TitleName                 ;заглавие окна
  push    offset  ClassName                 ;название класса окна
  push    0                                 ;идентификатор предка
  call    CreateWindowExA@48                ;создание окна
  cmp     eax,0                             ;проверка на наличие ошибок, и если
они
  jz      Error                             ;есть, то завершить работу
  mov     NewHandle,eax                     ;сохраняем полученный дескриптор
окна
;-------------------------------------------------------------------------------
;отображение и обновление окна на экране
  push    SW_SHOWNORMAL                     ;тип отображения окна – обычное
  push    NewHandle                         ;дескриптор окна
  call    ShowWindow@8                      ;отображать окно
  push    NewHandle                         ;дескриптор окна
  call    UpdateWindow@4                    ;обновить (нарисовать) видимую
  ;часть окна (посылается сообщение WM_PAINT, которое обрабатывается функцией
  ;DefWindowProcA@16)
;-------------------------------------------------------------------------------
;цикл обработки сообщений
MsgLoop:                                    ;начало цикла
  push    0
  push    0
  push    0
  push    offset  Msg                       ;в структуру Msg
  call    GetMessageA@16                    ;получаем сообщение
  cmp     eax,0                             ;если пришло сообщение о завершении
  je      EndLoop                           ;работы программы, то выходим из цикла,
  push    offset  Msg                       ;иначе переводим сообщение нажатых и
  call    TranslateMessage@4                ;отпущенных клавиш
  push    offset  Msg                       ;и переходим на выполнение процедуры
  call    DispatchMessageA@4                ;главного окна
  jmp     MsgLoop                           ;продолжаем цикл
Error:                                      ;сюда попадаем при ошибке,
EndLoop:                                    ;а сюда при нормальном завершении работы
  push    0
  call    ExitProcess@4                     ;завершаем работу программы
;-------------------------------------------------------------------------------
;процедура главного окна
;она получает следующие параметры через стек
;[esp+14h]LPARAM дополнительная информация о сообщении
;[esp+10h]       WPARAM дополнительная информация о сообщении
;[esp+0ch]идентификатор сообщения (определяет его тип)
;[esp+08h]дескриптор окна, получившего сообщение
WndProc  proc                               ;начало процедуры главного окна
```

```
        push    ebp                             ;сохраняем используемые регистры в стек
        mov     ebp,esp                         ;сохраняем вершину стека в ebp для
                                                ;доступа к переданным параметрам
        push    ebx
        push    esi
        push    edi
        cmp     dword ptr [ebp+0ch],WM_DESTROY          ;определяем тип
пришедшего
        je      WMDestroy                               ;сообщения и переходим к
        cmp     dword ptr [ebp+0ch],WM_CREATE           ;соответствующему
обработчику
        je      WMCreate                                ;сообщения
        cmp     dword ptr [ebp+0ch],WM_LBUTTONDOWN
        je      WMLBDown
        cmp     dword ptr [ebp+0ch],WM_RBUTTONDOWN
        je      WMRBDown
        push    dword ptr [ebp+14h]             ;если сообщение не подходит ни под
        push    dword ptr [ebp+10h]             ;одно из обрабатываемых, необходимо
        push    dword ptr [ebp+0ch]             ;вызвать API функцию DefWindowProc,
        push    dword ptr [ebp+08h]             ;для чего сохраняем в стек все полученные
        call    DefWindowProcA@16               ;параметры и вызываем функцию
        jmp     Finish                          ;и завершаем работу процедуры
    WMCreate:                   ;сюда переходим при сообщении о  создании окна
        jmp     Finish          ;создается пустое окно(без элементов) и поэтому просто
                                ;завершаем работу процедуры
    WMLBDown:                   ;сюда переходим при сообщении о  нажатии левой кнопки
мыши
        push    0                                       ;выводим соответствующее
сообщение
        push    offset  Cap                             ;для чего вызываем функцию
MessageBox
        push    offset  MessageLB
        push    dword ptr       [ebp+08h]               ;обратите внимание на наличие
предка
        call    MessageBoxA@16                          ;выводим сообщение
        jmp     Finish                                  ;и завершаем работу процедуры
    WMRBDown:           ;сюда переходим при сообщении о  нажатии правой кнопки мыши
        push    0                                       ;выводим соответствующее
сообщение,
        push    offset  Cap                             ;для чего вызываем функцию
MessageBox
        push    offset  MessagePQ
        push    dword ptr       [ebp+08h]
                                                ;продолжаем выполнение процедуры
                                                ;что приведет к закрытию окна
    WMDestroy:                                  ;сюда переходим при сообщении о закрытии
окна
        push    0                               ;посылаем сообщение для
        call    PostQuitMessage@4               ;завершения работы приложения
    Finish:                             ;сюда переходим  для завершения работы процедуры
        pop     edi                             ;восстанавливаем регистры
        pop     esi
        pop     ebx
        pop     ebp
        ret     16      ;возвращаемся в основную программу с удалением 16 бит из стека
    WndProc endp                                    ;конец процедуры главного окна
    end start                                       ;конец программы
```

Текст включаемого файла ProgWinMain.inc

```
    includelib c:\masm32\include\user32.lib             ;библиотеки
    includelib c:\masm32\include\kernel32.lib           ;API функций
```

```
        ;список сообщений, которые приходят в очередь сообщений
WM_DESTROY          equ     2h              ;при закрытии окна
WM_CREATE           equ     1h              ;при создании окна
WM_LBUTTONDOWN      equ     201h            ;при нажатии левой кнопки мыши
WM_RBUTTONDOWN      equ     204h            ;при нажатии левой кнопки мыши
        ;константы для определения стиля класса окна, легко убедиться, что они представляют
CS_VREDRAW          equ     1h              ;собой флаги, то есть в двоичном
CS_HREDRAW          equ     2h              ;представлении имеют единственную
CS_GLOBALCLASS      equ     4000h           ;единицу, а остальные цифры это нули
        ;стиль класса окна есть логическая сумма соответствующих констант
style               equ     CS_VREDRAW + CS_HREDRAW + CS_GLOBALCLASS
WS_OVERLAPPEDWINDOW equ     000cf0000h
IDI_APPLICATION     equ     32512   ;идентификатор стандартной пиктограммы
IDC_CROSS           equ     32515   ;идентификатор стандартного курсора
SW_SHOWNORMAL       equ     1       ;режим показа окна – нормальный
;-------------------------------------------------------------------------------
        ;список API функций, которые будут использованы в программе
extern  MessageBoxA@16:near             ;создает окно с сообщением
extern  CreateWindowExA@48:near         ;создает окно (главное окно в моей программе)
extern  DefWindowProcA@16:near          ;стандартная процедура обработки сообщений
extern  DispatchMessageA@4:near         ;предает сообщение соответствующему окну
extern  ExitProcess@4:near              ;завершение работы программы
extern  GetMessageA@16:near             ;получить следующее сообщение из очереди
extern  GetModuleHandleA@4:near         ;получает идентификатор приложения
extern  LoadCursorA@8:near              ;загружает курсор
extern  LoadIconA@8:near                ;загружает икону
extern  PostQuitMessage@4:near          ;отправляет сообщение о завершении работы в
                                        ;начало очереди сообщений
extern  RegisterClassA@4:near           ;регистрирует класс окна
extern  ShowWindow@8:near               ;отображает окно
extern  TranslateMessage@4:near         ;переводит сообщения о нажатой и
                                        ;отпущенной клавише в сообщение ASCII символ
extern  UpdateWindow@4:near             ;обновляет окно (рабочую область)
;-------------------------------------------------------------------------------
MSGSTRUC    struc                       ;структура сообщения
  MSHWND        dd      ?               ;идентификатор окна, получившего сообщение
  MSMESSAGE     dd      ?               ;идентификатор сообщения
  MSWPARAM      dd      ?               ;дополнительная информация о сообщении
  MSLPARAM      dd      ?               ; дополнительная информация о сообщении
  MSTIME        dd      ?               ;время посылки сообщения
  MSPT          dd      ?               ;положение курсора
MSGSTRUC    ends
WNDCLASS    struc                       ;структура класса окна
  CLSSTYLE      dd      ?               ;стиль окна
  CLWNDPROC     dd      ?               ;указатель на главную процедуру окна
  CLSCEXTRA     dd      ?               ;информация о доп. байтах для структуры
  CLWNDEXTRA    dd      ?               ;информация о доп. байтах для окна
  CLSHINSTANCE  dd      ?               ;дескриптор приложения
  CLSHICON      dd      ?               ;идентификатор иконы
  CLSHCURSOR    dd      ?               ;идентификатор курсора
  CLBKGROUND    dd      ?               ;цвет фона
  CLMENUNAME    dd      ?               ;имя-идентификатор меню
  CLNAME        dd      ?               ;имя класса окна
WNDCLASS    ends
```

Работу этой программы поясняют комментариев в листинге. Программа довольно сложная, но основная ее часть типична при програмировании под Windows и будет повторяться в других программах. Единственная ее функция – реакция на кнопки мыши – введена для демонстрации правильной работы приложения, остальной код стандартный. Следует обратить внимание, что Windows использует регистры EBX, EDI и ESI для своих нужд и их изменение в программе может привести к нежелательным результатам. Кроме того, необходимо четко понимать способ функционирования процедуры окна, так как именно он определяет суть программы под Windows. Задача данной процедуры – правильная реакция на **все** приходящие сообщения, и поэтому она должна содержать вызов функции DefWindowsProc или корректно обрабатывать все возможные варианты сообщений, что зачастую нецелесообразно; также процедура окна должна удалить из стека все полученные параметры (16 байт).

4. ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа 1

Построение интерфейсов микропроцессорных систем с использованием непрограммируемых и программируемых интерфейсных компонентов. Подключение клавиатуры и памяти к микропроцессорной системе

Цель работы: изучить способы построения системной магистрали МПС, научиться подключать клавиатуру, блок памяти и другие внешние устройства к микропроцессорной системе.

Задание для домашней подготовки

1. Изучить структуру «Микролаб 1810»;
2. Ознакомиться с описанием параллельного и последовательного порта ввода-вывода в «Микролаб 1810» по методическому пособию «Аппаратно программные средства на базе микропроцессора КМ1810ВМ86»(часть 1- уроки 9, 10) [12];
3. Ознакомиться с описанием принципов построения ОЗУ и ПЗУ и организацией памяти в «Микролаб 1810» по методическому пособию «Аппаратно программные средства на базе микропроцессора КМ1810ВМ86»(часть 1- уроки 7, 11) [12].

Задания для самостоятельной работы: изучить функциональные схемы и принципы работы шинных формирователей (ШФ) КР580ВА86 (Intel 8286), буферных регистров (БР) КР580ИР82 (Intel 8282), статических микросхем памяти ОЗУ серии КР537 или К541КР537РУ13, ПЗУ (К573РФ2), программируемого параллельного интерфейса К580ВВ55, программируемого последовательного интерфейса К580ВВ55, генератора тактовых импульсов КР1810ГФ24, контроллер прерываний К1810ВН59 (или К580 ВН59А).

Примечание. Справочную информацию по используемым микросхемам рекомендуем искать в Internet (например, на сайтах http://digitchip.by.ru, http://microprocessor.by.ru) или пользуясь поисковыми сайтами (например, www.google.ru).

Порядок выполнения лабораторной работы

1. Работа с учебной микро-ЭВМ «Микролаб 1810»

Для включения микро-ЭВМ необходимо:

--подключить учебную микро-ЭВМ к сети переменного тока;

--включить кнопку «Сеть»

--нажать управляющую кнопку «Сброс». При этом на связь с Вами выйдет программа «Монитор» и на экране дисплея состоящего из восьми

семисегментных индикаторов должно появиться сообщение монитора «-86 1.1» (86-тип монитора 1.1-номер его программной версии)

Индикаторы разделены на две группы по четыре клавиши в каждой. Левая группа относится к полю адреса, правая- к полю данных.

После указанных действий микро-ЭВМ готова к работе и находится в режиме ожидания директив оператора.

Знак «-», появляющейся в старшем разряде поля адреса означает, что микро-ЭВМ «Микролаб 1810» ожидает ввода одной из десяти команд (клавиши 0-9) Когда команда введена знак «-» исчезает и появляется знак «.» в младшем индикаторе поля адреса, который означает, что последующий ввод будет направлен в поле адреса и работа монитора с этого момента определяется типом введенной команды.

2. Изучить краткое описание «Микролаб КМ 1810 ВМ 86», адресацию памяти и внешних устройств в «Микролаб 1810».

3. Изучить команды клавиатуры (10 команд) учебной микро-ЭВМ

4. Запустить на выполнение с адреса FD05:100 записанную в область дополнительного ПЗУ программу, имитирующую бросание игральной кости. Программа работает следующим образом: после того, как программа загружена в память и управление передаётся на начало программы, на индикаторе начинает катиться «кость». Нажатие любой клавиши (кроме «ПРЕР» и «СБРОС») останавливает кость и выдаёт на индикатор плавно опускающуюся строку вида «SIDE #», где # – номер выпавшей грани кости (от 1 до 6), после этого внутренний динамик произносит выпавшее число (голосом). Затем ожидается нажатие любой клавиши, после которого программа начинает работать сначала. Ознакомиться с текстом программы, приведенным в прил. 4. Составить блок-схему работы программы. Особое внимание обратить на задания режимов сканирования клавиатуры и динамического режима вывода информации на дисплей с использованием контроллера клавиатуры и индикации КР580ВВ79.

5. Ввести в учебную микро-ЭВМ «Микролаб 1810» и запустить на выполнение тестовую программу обеспечивающую получение числа из порта ввода (FFFB) и визуальное отображение числа в шестнадцатеричном коде на индикаторах дисплея. Обратить особое внимание на то, как в тестовой программе осуществляется предварительная настройка микросхем КР580ВВ55 и КР580ВВ79 на работу в соответствующих режимах (см. прил. 5). Составить таблицу сотвествия между десятичными, двоичными и шестнадцалиричными значениями величен (от $15_{(10)}$ до $33_{(10)}$)

6. Проанализировать структурную схему построения однопроцессорной системы обмена информацией с внешними устройствами и памятью (см. рис. 20) и нарисовать (поэтапно) принципиальную электрическую схему однопроцессорной системы обмена информацией с внешним устройствами, содержащую :

6.1. Принципиальную электрическую схему модуля центрального процессора, обеспечивающую формирование магистралей микропроцессорной системы в минимальном режиме на основании структурной схемы на рис. 20 . Схема должна выполнять следующие функции:

- Демультиплексирование шины адреса/данных (распределение ее на шину адреса и шину данных).
- Буферизацию шин (увеличение нагрузочной способности линий шин, обеспечение возможности их перехода в третье состояние).
- Формирование сигналов управления.

Выполнение первой функции осуществляется с помощью регистров-защелок, например, буферных регистров i8282, i8283. Обобщенная структурная схема регистра-защелки содержит восемь D-триггеров с выходными схемами, имеющими три состояния.

Для увеличения нагрузочной способности (вторая функция) двунаправленной шины данных используют 8-разрядные формирователи i8286 или i8287 . Формирователи i8286 не инвертируют данные, а i8287 – инвертируют.

Третья функция реализуется с помощью дополнительных логических элементов, формирующих сигналы шины управления из выходных сигналов МП. На шине управления должно действовать, как минимум четыре типа сигналов:

- чтение из памяти;
- запись в память;
- чтение с устройства ввода-вывода);
- запись в устройство ввода-вывода;

Шина управления используется для вывода сигналов и является однонаправленной.

Шина адреса является однонаправленной.

Шина данных является двунаправленной. В некоторых случаях данные генерируются МП и передаются от него к определенному устройству. Это устройство открывается с помощью заданного логического состояния линий адресной шины и получает данные с шины данных. В других случаях данные генерируются каким-то источником и передаются микропроцессору посредством шины данных. Хотя передача данных по шине может производиться в обоих направлениях, в каждый заданный момент времени она осуществляется лишь в одном направлении. Это означает, что для передачи данных в систему и их приема из системы МП переводится в соответствующий режим. Каждый периферийный модуль микропроцессорной системы имеет вход для приема сигнала "Выбор модуля" (CS)- В процессе работы микропроцессора с помощью этого сигнала одновременно может активизироваться только один из периферийных модулей. Это означает, что возможен обмен данными между выбранным модулем и процессорным модулем. Выходы

остальных модулей при этом остаются в высокоимпедансном состоянии (отключенном) и на работу микропроцессора не влияют.

6.2. Дополнить принципиальную электрическую схему модуля центрального процессора дополнительными периферийными модулями (в соответствии с полученным от преподавателя заданием)

a. ОЗУ объемом 2КВ расположено начиная с адреса 00000 (рекомендуется выполнить на базе статических микросхем памяти сери КР537 или К541). ПЗУ объемом 4КВ расположено начиная с адреса FF000 FFFFF (рекомендуется выполнить на базе ППЗУ (К573РФ2)
b. Имитатор внешнего устройства ввода -- восемь переключателей, подключаемых к МПС через порт В ППИ №2 (м/с К580ВВ55)
c. Имитатор внешнего устройство вывода -- восемь светодиодов, подключаемых к МПС через порт В ППИ №1 (м/с К580ВВ55)
d. Клавиатурная матрица 8×8 и дисплей из шести ячеек (семисегментных индикаторов) подключаемые к МПС через свободные порты ППИ №1 и №2. При разработке воспользоваться материалами лабораторной работы №8.6 из тома 3 «Микропроцессоры: средства отладки, лабораторный практикум и задачник».
e. Интерфейс последовательного ввода-вывода для связи с инструментальной ПЭВМ выполнить на БИС программируемого интерфейса связи в последовательном коде КР580ВВ51А. Предусмотреть задание стандартных скоростей передачи по последовательному порту (см. табл. 17) установкой соответствующих контактов переключателя S17 входящего в состав делителя частоты (см. рис. 13).
f. Контроллер прерываний на 8 запросов (рекомендуется выполнить на базе микросхемы К1810ВН59 (или К580 ВН59А)
g. Дешифраторы памяти ОЗУ, ПЗУ и внешних устройств выполнить, используя микросхемы серии К155, К555 (например К155ЛА2, К155ЛЛ1) и КР556РТ4 (ПЗУ объемом 256×4).

1. На основании анализа принципиальных электрических схем на рис. 16 и 17 составить карту прошивки ПЗУ на базе микросхемы КР556РТ4 (рис. 16 микросхема D35), реализующую возможность адресации портов внешних устройств в «Микролабе» в соответствии с картой памяти (табл. 10) и карту прошивок ПЗУ на базе м/с КР556РТ4, реализующая возможность адресации ОЗУ и ПЗУ объемом 4КВ.

Примечание 1. В соответствии с табл. 10 вместо БИС КР 580 ВВ79 в нашем случае подключается контроллер прерываний на 8 запросов на базе микросхемы К1810ВН59 (или К580 ВН59А).

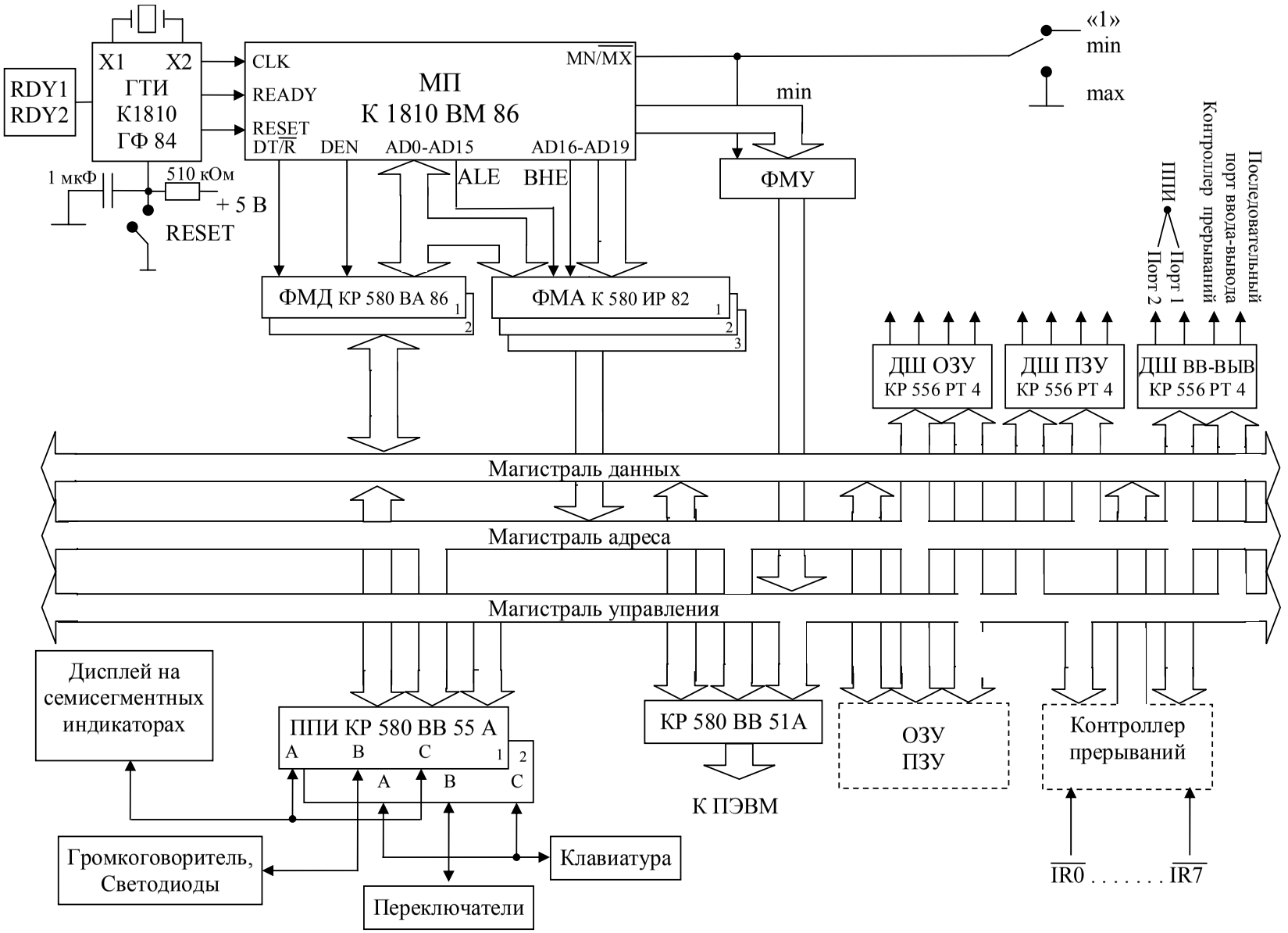


Рис. 20. Структура МПС на базе МП БИС КР 1810ВМ86

Содержание отчета

1. Точное наименование и цель работы.
2. Блок-схемы исследованных программ.
3. Таблицу сответствия между десятичными, двоичными и шестнадцалиричными значениями величен (от $15_{(10)}$ до $33_{(10)}$)
4. Структурная схема разрабатываемой МПС.
5. Принципиальная электрическая схема разработанной МПС.
6. Карта прошивки ПЗУ на базе м/с КР556РТ4 (рис. 16) для дешифратора портов внешних устройств в «Микролаб 1810» в соответствии с картой памяти (табл. 10)
7. Карта прошивки ПЗУ на базе м/с КР556РТ4, реализующая возможность адресации ОЗУ объемом 2КВ, расположенного начиная с адреса 00000.
8. Карта прошивки ПЗУ на базе м/с КР556РТ4, реализующая возможность адресации ПЗУ объемом 4КВ, расположенного начиная с адреса 00000.

Примечание. Отчет по лабораторной работе №1 выполняется по частям (на миллиметровой бумаге карандашом в соответствии с требованиями к условным графическим обозначениям интегральных микросхем) и представляется для проверки преподавателю по мере его оформления. В полном объеме отчет должен быть выполнен к окончанию лабораторного практикума

Лабораторная работа 2

Прохождение задач пользователя на ПЭВМ. Получение исходного, объектного и загрузочных модулей. Отладчики

Цель работы: изучить динамику развития программной модели МП фирмы Intel, динамик развития моделей микропроцессоров фирмы Intel, этапы прохождения задач пользователя на ПЭВМ, работу с отладчиками.

Задания для домашней подготовки

1. Ознакомиться с содержанием обучающих курсов
2. Изучить динамику развития программной модели МП фирмы Intel от i8080 до Pentium 4.

Примечание. При изучении использовать обучающий курс I386.exe, книги[4,9].

3. Ознакомиться с возможностями следующих групп программ для отладки, исследования и «вскрытия» исполняемых модулей:
   - отладчики реального режима (InSight, Meffistofel, Turbo Debugger (TD),AFD и др.);
   - отладчики защищенного режима (Soft-Ice, DeGlucker и др.);
   - автоматические дизассемблеры (Sourcer, Watcom Disassembler и др.);
   - интерактивные дизассемблеры (IDA, DisDoc и др.);
   - просмотровые программы с встроенным дизассемблированием и возможностью изменения кода (Hiew и др.);
4. Изучить режимы работы одного из отладчиков (AFD, TD или др.)

Задания для самостоятельной работы: изучить динамику развития моделей микропроцессоров фирмы Intel от i8080 до Pentium 4 (дата рождения, отличительные характеристики) (смотри сайты **www.Intel.ru**, **www.iXBT.com** и др.).

Порядок выполнения лабораторной работы

1. Сдать коллоквиум по теме «Микропроцессоры I80×86» (запустить последовательно предложенные преподавателем обучающие курсы – коллоквиум считается сданным, если по каждой теме получена оценка не ниже «хорошо» или «семь».
2. Изучить методы работы и ключи одного из трансляторов (MASM или TASM).
3. Ознакомиться с работой одного из отладчиков (debug, AFD ,TD).
4. Взять из папки « Образцы задач к лабораторной работе-2» исходный модуль программы решения задачи 9 из приложения 2. Получить объектный и загрузочные файлы (см. рис. 19). Выполнить загрузочный файл с использованием выбранного Вами отладчика. Результаты продемонстрировать преподавателю.Написать программу решения задачи 10 из набора базовых задач (см. приложение 2).
5. Получить у преподавателя задачу из набора вычислительных и логических задач (прил. 3), написать программу решения задачи. Выполнить

загрузочный файл с использованием отладчика. Результат продемонстрировать преподавателю.

Примечание. Обрабатываемые массивы должны задаваться в теле программы [3,4].

Содержание отчета

1. Точное наименование и цель работы.
2. Описание динамики развития программной модели МП фирмы Intel от i8080 до Pentium 4.
3. Описание шести новейших моделей микропроцессоров (включая многоядерные) от фирмы Intel по 2 модели для настольных ПК , ноутбуков и массовых серверов (дата рождения, отличительные характеристики) .
4. Описание ключей выбранного транслятора (MASM или TASM).
5. Описание окон и команд одного из отладчиков.
6. Листинги разработанных программ и описания используемых в них команд (форматы, методы адресации).
7. Сравнительная характеристика прграммных средств для исследования и вскрытия программ.

Лабораторная работа 3

Ознакомление с работой и методами программирования учебной микропроцессорной системы «Микролаб 1810»

Цель работы: изучить структуру МПС «Микролаб», этапы разработки программного обеспечения для целевой микропроцессорной системы «Микролаб» с использованием инструментальной ПЭВМ, ознакомиться с использованием в МПС программируемых интерфейсных компонентов.

Задания для домашней подготовки

1. Изучить краткое описание «Микролаб КМ 1810 ВМ 86», адресацию памяти и внешних устройств в «Микролаб 1810».
2. Повторить по методическому пособию «Организация интерфейсов микропроцессорных систем обработки информации на базе программируемых компонентов» построение параллельных и последовательных портов ввода-вывода МПС с использованием микросхем программируемого параллельного интерфейса (ППИ) КР580ВВ55 (Intel 8255) и универсального синхронно-асинхронного приемопередатчика К580ВВ51 (i8251).

Повторить описание параллельного и последовательного порта ввода-вывода в «Микролаб 1810» по методическому пособию «Аппаратно программные средства на базе микропроцессора КМ1810ВМ86»(часть 1- уроки 9, 10) [12].

3. Изучить описание контроллера клавиатуры и индикации КР580ВВ79 по методическому пособию «Аппаратно программные средства на базе микропроцессора КМ1810ВМ86»(часть 1 - уроки 8) [12]. Изучить использование контроллера клавиатуры КР580ВВ79 в «Микролаб 1810».

Примечание. Можно также соответствующую информацию найти на сайтах по цифровым интегральным микросхемам в Internet

4. Изучить структура лабораторного комплекса (см. рис. 9).
5. Изучить работу программы пересылки разработанного программного кода в целевую микропроцессорную систему «Микролаб» (программа SLINK).

Задания для самостоятельной работы: изучить динамику развития моделей микропроцессоров фирмы AMD от Am286 до Opteron (дата рождения, отличительные характеристики) (смотри сайты **www.amd.ru**, **www.iXBT.com** и др.). Особое внимание обратить на архитектурные особенности данного семейства МП – клонмейкера МП от Intel.

Порядок выполнения лабораторной работы

1. Получить задание у преподавателя, составить код управляющего слова и исследовать с помощью задающих воздействий, подаваемых с тумблерных регистров универсального стенда для исследования БИС, работу программируемого параллельного интерфейса К580ВВ55(i8255) и универсального синхронно-асинхронного приемопередатчика К580ВВ51 (i8251).

2. Написать программы решения задач 1 и 2 из набора базовых задач (прил. 2). Получить листинг программ (см. Описание ключей выбранного транслятора). Имея на экране ПК машинные коды, ввести в «Микролаб 1810» с клавиатуры и запустить на выполнение программы решения базовых задач 1, 2. При вводе программ помнить о том, что в памяти микропроцессорной системы на базе процессоров Intel многобайтные числа записываются в обратном порядке.

3. Построить блок-схемы программ, обратив особое внимание на задание режимов работы программируемого параллельного интерфейса К580ВВ55 (i8255).

4. Собрать лабораторный комплекс, описанный в разделе 2.3. Все подключения внешних устройств вести при выключенном ПК.

5. Получить у преподавателя пять задач из набора базовых задач (прил. 2), составить программу их решения на языке Assembler, выполнить и продемонстрировать результат преподавателю. Задачи должны разрабатываться с помощью инструментальной ПЭВМ и выполняться на Микролаб. Процедуру пересылки обеспечивает программа SLINK (см. раздел 3.2). Пересылаемый файл должен быть приведен к виду *.bin. Если после процедуры прохождения задачи на ПЭВМ у вас имеется файл вида *.com, то для преобразования к формату *.bin его достаточно переименовать. Если после процедуры прохождения задачи на ПЭВМ у вас имеется файл вида *.exe, то для преобразования к формату *.bin его достаточно обработать процедурой exe2bin и на выходе будет сформирован нужный файл *.bin.

Содержание отчета

1. Точное наименование и цель работы.
2. Структура и программная модель I 8255 (К580ВВ55). Управляющее слово, реализующее заданный преподавателем режим работы КР580ВВ55.
3. Структура и программная модель универсального синхронно-асинхронного приемопередатчика К580ВВ51 (i8251).
4. Структура и программная модель контроллера клавиатуры и индикации КР580ВВ79.
5. Структура и карта памяти учебной микро-ЭВМ «Микролаб 1810».
6. Листинги и блок-схемы программ, разработанных в п. 5, а также описания используемых в них команд (форматы, методы адресации).
7. Описание шести новейших моделей микропроцессоров (включая многоядерные) от фирмы AMD по 2 модели для настольных ПК , ноутбуков и массовых серверов (дата рождения, отличительные характеристики) .
8. Описание особенностей развития программной модели и системы команд МП фирмы AMD от Am286 до Opteron.

Лабораторная работа 4

Программирование стандартных портов внешних устройств ПЭВМ, обработкой прерываний в ПЭВМ

Цель работы: познакомиться с программированием стандартных портов персонального компьютера, обработкой прерываний в ПЭВМ.

Задания для домашней подготовки

1. Изучить адресацию стандартных портов в ПЭВМ на базе МП 80×86 (см. литературу [5,6]).
2. Изучить организацию прерываний в микропроцессорных системах на базе МП 80×86 (см. [2,3]))

Задания для самостоятельной работы: изучить динамики развития моделей микропроцессоров фирмы Cyrix-VIA (дата рождения, отличительные характеристики). Особое внимание обратить на архитектурные особенности данного семейства – клонмейкера МП от Intel (смотри сайт **www.iXBT.com** и др.).

Порядок выполнения лабораторной работы

1. Получить у преподавателя набор задач (например, 11, 12, 14, 15, 16,) из базовых задач для Микролаб и ПЭВМ (см. прил. 1). Написать программы на языке Assembler и продемонстрировать преподавателю полученные звуковые и световые эффекты на ПЭВМ.
2. Написать программу обработки прерываний при переполнении: Делимое находится в двух регистрах AX, DX; делитель равен двум; частное не умещается в одном регистре. Процедура DVROK должна перехватывать прерывание INT0 и выводить на экран соответствующее сообщение.

Содержание отчета

1. Точное наименование и цель работы.
2. Схемы подключения тестируемых внешних устройств к ПЭВМ.
3. Описание механизма обработки прерываний в МП I80×86.
4. Листинги разработанных программ.
5. Описание динамики развития и архитектурных особенностей шести новейших моделей микропроцессоров фирмы Cyrix-VIA (дата рождения, отличительные характеристики).

Лабораторная работа 5

Изучение интерфейса Centonics и последовательного интерфейса RS-232

Цель лабораторной работы: изучить интерфейс Centonics, ознакомиться с работой параллельных портов ПЭВМ; изучить программирование интерфейса RS-232 в составе лабораторного комплекса, приведенного на рис. 15.; ознакомиться с работой COM-портов ПЭВМ.

Задания для домашней подготовки

1. Ознакомиться с работой параллельного (LPT-порты) и последовательного интерфейсов (COM-порты) ПЭВМ (см. [11])
2. Построить программную модель LPT-порта [в режиме SPP (Standard Parallel Port)].
3. Составить таблицы описания выводов LPT-порта.
4. Изучить особенности возможных режимов работы LPT-портов ПЭВМ типа IBM PC AT (SPP, EPP, ECP). Ознакомиться с последовательным интерфейсом RS-232 (COM-порты ПЭВМ) (см.[11]).
5. Построить программную модель COM-порта.
6. Составить таблицы описания выводов COM-порта.
7. Изучить особенности возможных режимов работы COM-порта ПЭВМ (см. [11]).
8. Ознакомиться с программированием COM портом ПК под ОС Windows(см. раздел 3.2 данного пособия])

Задания для самостоятельной работы: изучить динамики развития моделей микропроцессоров фирмы Motorola от M6800до Power PC (дата рождения, архитектурные особенности) (смотри сайт **www.iXBT.com** и др.).

Порядок выполнения лабораторной работы

1. Подключить к ПЭВМ к порту LPT схему для определения правильного прохождения сигналов по шине данных в стандарте Centronics (см. рис. 15).
2. Написать на языке Assembler программу тестирования параллельного порта, которая обеспечивает режим бегущей единицы на выходных линиях LPT-порта.

Примечание 1. Параллельный порт переключить в BIOS на работу в стандартном SPP режиме.

Примечание 2. Минимальное время горения одной лампочки – 1 с.

3. Получить задание у преподавателя и исследовать с помощью задающих воздействий, подаваемых с тумблерных регистров универсального стенда для исследования БИС, работу и универсального синхронно-асинхронного приемопередатчика К580ВВ51 (i8251).
4. Собрать лабораторный комплекс в соответствии с рис. 15. Соединить учебную микро-ЭВМ «Микролаб» и ПЭВМ с использованием «нуль-модемного» (Null Modem) кабеля (тип монитора – клавиатурный).

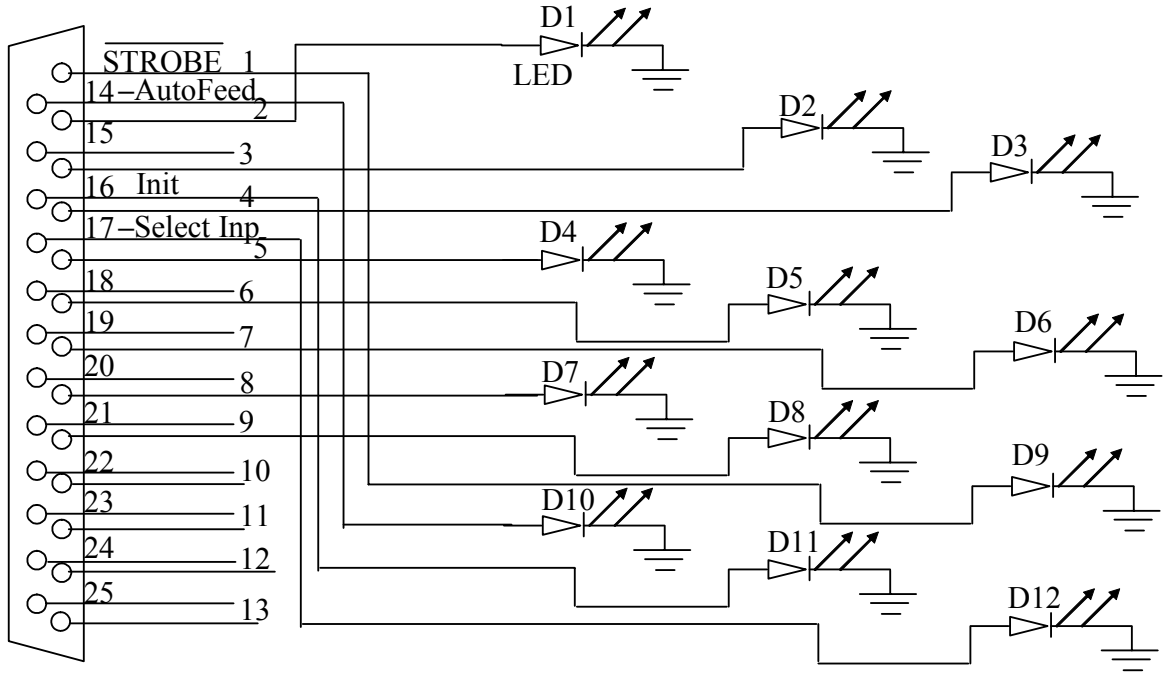


Рис. 21. Схема для определения правильного прохождения сигналов по шине данных в стандарте Centronics

5. Написать программы для ПЭВМ и «Микролаб», обеспечивающие передачу байта через COM-порт со стороны ПЭВМ, его прием учебной микро-ЭВМ, инвертирование и передачу обратно в ПЭВМ. ПЭВМ должна принять инвертированный байт назад, сложить его с переданным и результат вывести на экран ПЭВМ. Программу для ПЭВМ написать с использованием языка Паскаль (программа pc-com.pas), для «Микролаб» – языка Ассемблера (программа micro-com.asm). Формат передачи данных: длина символа 8 бит, стоповый бит, без бита четности, скорость 9600, коэффициент пересчета скорости для КР580ВВ51А равен 64.

Содержание отчета

1. Точное наименование и цель работы.
2. Программная и логическая модель LPT-порта компьютера.
3. Схема тестирования LPT-порта компьютера.
4. Программная модель COM-порта компьютера.
5. Структуру лабораторного комплекса.
6. Схема соединения компьютеров с использованием нуль-модемного кабеля.
7. Тексты программ, разработанных в ходе выполнения лабораторных работ.
8. Описание динамики развития шести новейших моделей микропроцессоров фирмы Motorola (дата рождения, архитектурные особенности).
9. Описание особенностей развития программной модели и системы команд МП фирмы Motorola от M6800 до G5.

Лабораторная работа 6

Решение логических и вычислительных задач на ПЭВМ с использованием процедур и макрокоманд языка Assembler. Программирование на языке Assembler в среде Windows

Задание для домашней подготовки

1. Изучить использование процедур и макрокоманд в языке Assembler.
2. Ознакомиться с программированием в среде Windows (см. данное методическое пособие и книгу В. Юрова «ASSEMBLER»).

Порядок выполнения лабораторной работы

1. Изучить и запустить на выполнение три тестовые примера, описанные в разделе 3.4. «Программирование на Assembler в среде Windows» данного методического пособия.
2. Запустить на выполнение тестовый пример ТЕСТ 1. Особое внимание обратить на вызов процедуры ReadWord (ввод с клавиатуры обрабатываемых данных) и работу с пакетом программ Print (см. главу 3) - вывод на экран результатов.
3. Получить у преподавателя две задачи из набора вычислительных и логических задач (приложение 2), выполнить их с привлечением процедуры ReadWord (ввод с клавиатуры обрабатываемых данных) и пакета программ Print (см. раздел 2.4) - вывод на экран результатов. Продемонстрировать результат преподавателю.

Примечание. Программы можно писать под современные версии Windows.

Содержание отчета

1. Листинги и блок-схемы разработанных программ.
2. Описание используемых процедур и правил их вызова.
3. Описание используемых библиотек макрокоманд и функций.

Лабораторная работа 7

Изучение структуры, сборка, оценка производительности персонального компьютера

Задание для домашней подготовки: ознакомиться с классификацией тестов оценки производительности персонального компьютера [9], программами **SiSoft Sandra** (см. сайт www.sisoftware.net) и Dr. Hardware (см. сайт www.dr-hardware.com).

Задания для самостоятельной работы: получить у преподавателя в соответствии с табл. 15 виртуальные деньги на покупку и виртуальную сборку трех компьютеров:

- Офисный ПК (BASE LEVEL COMPUTER)
- Домашний ПК (MIDDLE LEVEL COMPUTER)
- Сервер (HIGH-END LEVEL COMPUTER). Тип и характеристики сервера будут уточнены преподавателем в индивидуальном задании для каждого студента. Сервер, как правило, приобретается в виде готового изделия.

Примечание. Задания выполнить с привлечением ресурсов сети Internet.

Порядок выполнения лабораторной работы

1. Протестируйте производительность компьютеров в учебном классе пользуясь программой **SiSoft Sandra**.

Запустите программу SiSoft Sandra. Изучите интерфейс программы. Запустите на выполнение информационные модули (система в целом, материнская плата и т. д.) для выяснения конфигурации вашего компьютера. Выполните тесты производительности для процессора, мультимедийных функций и системной памяти. Обратите внимание на диаграмму сравнения параметров вашего процессора с процессорами других производителей. Учтите, что во время тестирования не рекомендуется передвигать курсор мыши и нажимать на клавиши.

Важным фактором, влияющим на производительность компьютера, является также эффективная работа его винчестера. Запустите тест дисков и протестируйте системный диск

таблица 15

ФИО студента			
Исходные финансы			
Наименование комплектующих	**BASE LEVEL COMPUTER**	**MIDDLE LEVEL**	**HIGH-END LEVEL COMPUTER**
Процессор (CPU)			
Cooler			
Материнская плата (Motherboard)			
Винчестер (HDD)			
Память (RAM)			
CD-ROM			

Дисковод (FDD)			
Видео-карта			
Звуковая карта			
Колонки			
Клавиатура			
Мышь+коврик			
Монитор (Monitor)			
Принтер			
Сканер			
Бесперебойник (UPS)			
Корпус			
Работа			
***ИТОГО**:*			

2. Протестируйте производительности компьютера в учебном классе, пользуясь программой «Dr Hardware». Создайте отчет о тестировании. Для этого воспользуйтесь Мастером создания рапортов. Пользуясь подсказками интерфейса, оставьте подключенными только требуемые модули. Сохраните отчет на жестком диске.

3. Протестировать свой домашний компьютер. Создайте отчет о тестировании.

4. Пользуясь полученными отчетами, сравните производительность компьютеров. Принимая во внимание отличия в конфигурации, прокомментируйте полученные результаты.

5. Составите описание своего домашнего ПК в соответствии со следующим планом:

- Полное название процессора (в старых или новых обозначениях).
- Ядро процессора.
- Частота процессора и процессорной шины.
- Объем кэш L1.
- Объем кэш L2.
- Модель Socketa.
- Модель системной (материнской) платы и тип чипсета на базе которого она выполнена.
- Тип MCH или северного моста.
- Тип ICH или южного моста.
- Возможные частоты внешней процессорной шины (FSB).
- Максимально возможный объем ОЗУ на плате.
- Реальный объем ОЗУ вашей плате.
- Возможные типы модулей памяти, поддерживаемые материнской платой.
- Типы модулей памяти в моей конфигурации.
- Тип интерфейса HDD.
- Объем жесткого диска.
- Тип и изготовитель видеокарты.
- Чипсет видеокарты.

- Объем видеопамяти.
- CD\R -CD/RW- DVD/R-DVD/RW
- Монитор.
- Количество и параметры периферийных интерфейсов ПК.
- Внешние устройства ПК.
- Мощность блока питания.
- Выполните расчет пропускной способности интерфейсов и устройств Вашего ПК: host-шина, кэш L2, шина PCI-E, шина AGP, шина памяти, шина PCI, интерфейс HDD, интерфейсы внешних устройств

Содержание отчета

1. Отчеты о тестировании ПК (в электронном виде).
2. Комментарии к полученным результатам.
3. Описание и структурную схему домашнего ПК.
4. Отчет о виртуальной сборке трех ПК. Отчет должен включать заполненную табл. 15 покупки трех ПК с указанием названия, стоимости и места покупки ПК. По каждому из применяемых комплектующих должна быть предоставлена полная информация о внешнем виде (картинка из Internet) и технические параметры. Отчет представляется в электронном виде. Образец отчета смотри на сайте с IP 10.135.1.61

5. КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ ПО ТЕМАМ

Тема 1.

Общие сведения из теории

1. Назовите основные характерные особенности реального и защищенного режимов работы МП.
2. Укажите максимально возможное количество дескрипторных таблиц, которые могут быть заданы.
3. Чем отличаются теневые регистры от программно-доступных?
4. Какую информацию содержат дескрипторные таблицы?
5. Укажите назначение регистра IDTR.
6. Какие команды необходимо выполнить перед переходом в защищенный режим работы?
7. Дайте характеристику существующих типов циклов шины.
8. Чем отличается обработка прерывания от обработки исключения?
9. Укажите назначения регистров, которые входят в программную модель 32-разрядного процессора.
10. Укажите назначения регистров управления CR, регистров тестирования TR.
11. Укажите назначения регистров отладки DR и системных адресных регистров GDTR, IDTR, TR, LDTR.
12. Укажите назначения и существующие типы дескрипторных таблиц.
13. Какое значение системного бита определяет тип дескриптора?
14. Дайте определение процесса свопинга и объясните, как он происходит.
15. Объясните принцип страничной организации памяти.
16. Объясните необходимость и принцип функционирования механизма защиты по привилегиям.
17. При каких условиях разрешается считывать данные определенного сегмента?
18. Какая информация помещается в вентилях вызовов?
19. Объясните особенности многозадачного режима работы ПК.
20. Укажите назначения буферов отложенной записи.
21. В каких случаях может нарушиться порядок обслуживания запросов на запись и чтение в буферах отложенной записи?
22. Укажите назначение OverDrive-процессоров.
23. Назовите назначения основных блоков структурной схемы процессора Pentium.
24. Какие микропроцессоры называются скалярными. Приведите примеры.
25. Какие микропроцессоры называются супескалярными. Приведите примеры.
26. Объясните работу блока предсказания адреса перехода на примере выполнения команды JC LABEL.
27. Какой эффект имеет разделение кэш-памяти на кэш-память команд и кэш-память данных?
28. Какие средства выявления ошибок имеет процессор Pentium?
29. В чем состоит принцип тестирования с помощью функциональной избыточности?
30. Какие функции тестирования имеет процессор Pentium?
31. Укажите назначение и возможности средств мониторинга производительности МП.

32. Укажите назначение и принципы выполнения команд MMX.
33. Укажите назначение и принцип выполнения команд SSE.
34. Как осуществляется переключение в режим сниженного энергопотребления и выход из него?
35. Сравните архитектуры 64-разрядных МП.
36. Объясните положительный эффект предикации.
37. Укажите особенности архитектуры EPIC.
38. Что такое кэш-память со сквозной записью и обратной записью?
39. Укажите различие между RISC- и CISC-процессорами.
40. Объясните различие между мультитредовой архитектурой и архитектурой с длинным командным словом.
41. Перечислите известные вам способы переименования ресурсов.
42. Приведите примеры различных реализаций механизмов предсказания ветвлений.
43. Укажите препятствия повышению эффективности суперскалярных микропроцессоров и процессоров с длинным командным словом.
44. Перечислите характерные особенности мультитредовой архитектуры.
45. В чем отличие мультитредовой архитектуры от традиционных мультипроцессорных систем?
46. Объясните механизм условного выполнения команд и его отличие от команд условного перехода.
47. Разъясните основные особенности архитектуры IA-64.
48. Перечислите основные типы архитектур универсальных микропроцессоров и характерные области их применения.
49. Укажите основных представителей микропроцессоров с архитектурой x86.
50. Что такое мультимедийное расширение системы команд? Приведите примеры реализаций мультимедийных расширений системы команд в микропроцессорах RISC- и CISC-архитектуры.
51. В чем смысл введения отдельных шин для работы процессора с кэш-памятью и основной памятью?
52. Перечислите составляющие технологии энергосбережения.
53. Укажите особенности организации кэш-памяти в микропроцессорах Intel шестого поколения.
54. Как в рамках архитектуры x86 сочетаются концепции RISC- и CISC-процессоров?
55. В чем состоят особенности реализации архитектуры x86 в конкурирующих изделиях фирм Intel, AMD, Cyrix? Сопоставьте эти семейства по основным структурным и функциональным характеристикам: реализация устройства выборки команд, управление ветвлениями, работа с кэш- памятью, диспетчеризация и исполнение команд.
56. Перечислите основные особенности микропроцессоров фирмы Transmeta.
57. Перечислите архитектурные особенности Alpha 21064, 21164, 21264. Прокомментируйте линию развития семейства микропроцессоров с архитектурой Alpha. Назовите состав и назначение основных блоков микропроцессоров.
58. В чем состоят характерные особенности архитектуры SPARC? Состав и назначение основных блоков?

59. Прокомментируйте развитие семейства процессоров с PA-архитектурой компании Hewlett-Packard. Структура и организация кэш-памяти. Состав исполнительных устройств. Механизм предсказания переходов.
60. Перечислите черты микропроцессоров семейств PowerPC и R-lxxxx как классических RISC-процессоров с разнесенной архитектурой.
61. Сопоставьте подходы различных производителей к организации кэш-памяти микропроцессоров. Укажите тенденции развития.
62. Назовите число возможных одновременно выполняемых операций в разных микропроцессорах. Что препятствует увеличению этого числа? Назовите длину конвейеров каждого микропроцессора.
63. Сравните механизмы реализации предсказания переходов в разных микропроцессорах. Назовите превалирующие тенденции развития.
64. Прокомментируйте развитие микропроцессоров отечественного производства

Тема 2.
Построение интерфейсов микропроцессорных систем с использованием непрограммируемых и программируемых интерфейсных компонентов. Подключение клавиатуры и памяти к микропроцессорной системе

1. Назовите составные части МПК.
2. Укажите назначение и составные части системной шины.
3. Назовите принципы передачи информации по шинам: адреса, данных, управления.
4. Назовите принципы построения МПС и охарактеризуйте их.
5. Объясните назначение входа управления третьим состоянием.
6. Дайте определение архитектуры МП.
7. Укажите различия между гарвардской и фон-неймановской архитектурами.
8. Какие функции выполняет устройство управления МП?
9. Что определяет содержимое счетчика команд? Как оно изменяется?
10. Чем отличается аккумулятор от других регистров МП?
11. Приведите основные характеристики и структурные схемы процессоров с фон-неймановской и гарвардской архитектурой.
12. Укажите назначение регистров МП с фон-неймановской архитектурой.
13. Охарактеризуйте функции устройства управления.
14. Назовите и охарактеризуйте существующие типы циклов шины.
15. Указать основные технические характеристики МП Intel 8086 и сравнить их с характеристиками МП Intel 8080.
16. Что такое минимальный и максимальный режимы работы МП Intel8086?
17. Каково назначение внешних выводов МП в минимальном режиме?
18. Каково назначение внешних выводов МП в максимальном режиме?
19. Опишите работу механизма прерываний в МП I80x86.
20. Можно ли в МП Intel 8086 одновременно (в одном такте) выставить адрес ячейки памяти и данные?
21. Опишите основной механизм сопряжения по времени работы МП и внешних устройств, имеющих меньшее быстродействие.

22. Каковы основные отличия работы МП в минимальном и максимальном режимах?
23. Объясните назначение шинного формирователя К1810ВВ86.
24. Почему к портам ввода-вывода следует обращаться как к чётным адресам памяти при подключении ВУ к линиям управления памяти?
25. В чем отличия статических и динамических ОЗУ?
26. Какие типы ПЗУ вы знаете?
27. Какой объем памяти имеется в учебной микроЭВМ Микролаб?
28. Почему слова в памяти целесообразно размещать по четным адресам?
29. Сколько 8-разрядных портов включает в себя БИС КР580ВВ55А.
30. Описать формат управляющего слова, задающего режим работы программируемого параллельного интерфейса (ППИ).
31. Описать основные режимы работы ППИ.
32. С помощью каких сигналов происходит обмен информацией между портами ввода-вывода и шиной данных.
33. Описать отличия последовательного порта ввода-вывода и параллельного порта.
34. Опишите формат слова выбора режима в микросхеме I8251.
35. Опишите работу I8251 в асинхронном режиме.
36. Опишите работу I8251 в синхронном режиме.
37. Постройте программную модель КР580ВВ51 (Intel 8251).
38. Сформируйте управляющее слово, настраивающее i8251 на работу в асинхронном режиме со скоростью передачи 1/64, длиной символа 8 бит данных, без разряда четности с 1 стоповым битом.
39. Сформируйте слово команды (слово приказа) контроллеру i8251 для разрешения приема/передачи.
40. Для каких целей используется команда сброса для i8251?
41. Дать определение порта ввода-вывода. Какая микросхема используется в Микролабе в качестве параллельных портов? В каких режимах она может работать?
42. Назначение 580ВВ55. Охарактеризовать. Назначение внешних выводов.
43. Для чего используется регистр управляющего слова в ППИ
44. Какие есть режимы работы каналов А В С для ВВ55?
45. Каким образом осуществляется установка разрядов буфера канала С.
46. Опишите особенности работы 580ВВ55 в режиме 0.
47. Опишите особенности работы 580ВВ55 в режиме 1.
48. Опишите особ работы 580ВВ55 в режиме 2.
49. Расскажите об использовании ВВ55 в Микролабе.
50. Для чего нужна ВВ55. Какие у нее режимы и как они устанавливаются.
51. Как программировать УСАПП на различные режимы работы.
52. Чем отличается асинхронный режим от синхронного режима работы.
53. Какими командами обращается МП к УСАПП.
54. Опишите сигналы управления модемом в БИС ВВ51.
55. Опишите программирование К580ВВ51.
56. Как определяет работу ВВ51 программа монитор в Микролаб-810
57. Объясните назначение входных и выходных сигналов схемы синхронизации.
58. Опишите функции БИС генератора i8284.
59. Опишите функции интерфейса центрального процессора с системной шиной.

60. Объясните принцип функционирования схемы буферного регистра.
61. Объясните принцип функционирования шинного формирователя.
62. Какие функции выполняет контроллер прерываний в МПС управления?
63. Какое число запросов прерываний может обслужить одна БИС ПКП?
64. Какого максимального количества запросов прерываний можно достичь в системе прерываний на базе БИС КР580ВН59?
65. Для чего предназначен режим специального маскирования?
66. Какие приоритетные схемы обработки запросов прерываний реализуются в БИС КР580ВН59?
67. Как изменяются приоритеты в схеме циклической обработки приоритетов?
68. В каких случаях целесообразно использовать режим фиксированных приоритетов?
69. В каких случаях целесообразно использовать режим циклических приоритетов?
70. Какой из входов IRQ имеет высший приоритет?
71. Как можно уменьшить приоритет входов IRQ?
72. Как можно отключить одну или несколько линий IRQ?
73. Когда могут записываться команды инициализации и управления?
74. Для чего необходимы сигналы INT и INTA?

Тема 3.

Прохождение задач пользователя на ПЭВМ. Получение исходного, объектного и загрузочных модулей. Отладчики

1. Почему слова в памяти целесообразно размещать по четным адресам?
2. Что представляет собой сегмент?
3. Сколько сегментов одновременно может быть доступно программе?
4. Какие дополнительные регистры существуют в микропроцессорах 80386 и выше?
5. Назовите возможные режимы работы процессоров 80386 и выше?
6. Составьте последовательность команд, заменяющую команду XLAT
7. С помощью каких команд организуется обмен между различными сегментами данных?
8. С помощью каких команд анализируются биты регистра флагов F?
9. В чем состоит различие в использовании флагов CF и OF?
10. В чем сходство и различие выполнения команд MUL и IMUL, а также DIV и IDIV?
11. Каковы основные особенности процедур коррекции результатов арифметических операций в МП ВМ86?
12. В чем заключается отличие команд сдвига в ВМ86 от ВМ80?
13. В чем состоит основное назначение команды SCAS?
14. Какие ограничения присущи командам передачи управления?
15. Какие команды предназначены для обеспечения работы МП i8086 в мультипроцессорном режиме?
16. Опишите структуру машинных инструкций микропроцессов серии Intel 80×86
17. Каково основное назначение директив языка ассемблера?
18. Каковы достоинства и недостатки языка ассемблера по сравнению с языками высокого уровня?

19. Приведите примеры мнемокодов команд с различными способами адресации .
20. Опишите назначение отладчики реального режима (InSight, Meffistofel, Turbo Debugger (TD),AFD и др.);
21. Опишите назначение отладчики защищенного режима (Soft-Ice, DeGlucker и др.);
22. Опишите назначение автоматических дизассемблеров (Sourcer, Watcom Disassembler и др.);
23. Опишите назначение интерактивные дизассемблеров (IDA, DisDoc и др.);
24. Опишите назначение просмотровых программ с встроенным дизассемблированием и возможностью изменения кода (Hiew и др.);
25. Перечислите основные команды выбранного Вами для выполнения лабораторной работы отладчика.

Тема 4.

Ознакомление с работой и методами программирования учебной микропроцессорной системы «Микролаб 1810»

1. Какой объем памяти имеется в учебной микроЭВМ Микролаб?
2. Почему слова в памяти целесообразно размещать по четным адресам?
3. Укажите назначение БИС программируемого параллельного интерфейса КР580ВВ55.
4. Опишите режимы работы программируемого параллельного интерфейса.
5. Назовите возможные комбинации включения портов БИС КР580ВВ55.
6. Какие порты параллельного интерфейса могут работать во всех возможных режимах?
7. Запишите управляющее слово для работы параллельного интерфейса (КР580ВВ55) в режиме 0 при настройке портов А и В на вывод, порта С – на вход.
8. Объясните принцип сброса/установки разрядов порта С БИС КР580ВВ55.
9. Укажите назначение БИС программируемого интерфейса клавиатуры и индикации.
10. Как программа монитора программирует БИС КР580ВВ79 в Микролаб?
11. Какое максимальное число клавиш можно соединить с БИС программируемого интерфейса клавиатуры и индикации КР580ВВ79?
12. Объясните особенности режима опроса матрицы клавиатуры БИС программируемого интерфейса клавиатуры и индикации КР580ВВ79?.
13. Объясните особенности режима опроса матрицы датчиков БИС программируемого интерфейса клавиатуры и индикации КР580ВВ79?.
14. Объясните особенности режима ввода по стробу БИС программируемого интерфейса клавиатуры и индикации КР580ВВ79?.
15. Опишите адресное пространство памяти в Микролаб.
16. Опишите адресное пространство устройства ввода-вывода в Микролаб.

Тема 5.

Программирование стандартных портов внешних устройств ПЭВМ, обработкой прерываний в ПЭВМ

1. Опишите адресацию портов ввода-вывода в процессорах семейства 80x86.
2. Как работают и какие методы адресации используют команды IN и OUT.
3. Приведите сравнительную характеристику видов обмена между МП и УВВ.
4. Какие существуют типы программного обмена?
5. Какие действия выполняет МП при переходе к подпрограмме обработки прерываний?
6. Назовите и охарактеризуйте типы прерываний МП i8086.
7. Охарактеризуйте понятия «прерывание» и «исключение».
8. Чем отличается обработка прерывания от обработки исключения?
9. Каким образом происходит запоминание содержимого аккумулятора, РОН, программного счетчика и флагов при обмене по прерыванию?
10. В каких случаях целесообразно применение прямого доступа к памяти?
11. Какие флаги изменяются при вызовах прерываний?
12. Организация прерываний в ПЭВМ на базе I80x86
13. Опишите основной механизм сопряжения по времени работы МП и внешних устройств, имеющих меньшее быстродействие.
14. Опишите работу обработчика прерываний в реальном режиме.

Тема 6.

Изучение интерфейса Centonics и последовательного интерфейса RS-232

1. Опишите структуру LPT-портов PC.
2. Опишите структуру COM-портов PC.
3. Опишите режимы работы современных LPT-портов компьютера.
4. Опишите скорости передачи информации через COM-порты современных компьютеров.
5. Постройте программную модель UART i8250.
6. Как формируется установка делителя частоты для i8250?
7. Сформируйте последовательность управляющих команд, устанавливающих UART в режим 9600 бит/с, длиной символа 8 бит, без разряда четности с 1-стоповым битом.
8. Какая команда в программе pc-com.pas запрещает прерывание от последовательного порта?
9. Какие команды в программе pc-com.pas организуют прием информации?
10. Какие команды в программе pc-com.pas производят передачу информации?
11. По какому адресу в памяти IBM PC находится базовый адрес последовательного порта COM2?
12. Построите программную модель КР580ВВ51 (Intel 8251).
13. Сформируйте управляющее слово, настраивающее i8251 на работу в асинхронном режиме со скоростью передачи 1/64, длиной символа 8 бит данных, без разряда четности с 1-стоповым битом.
14. Сформируйте слово команды (слово приказа) контроллеру i8251 для разрешения приема/передачи.

15. Какие команды в программе micro-com.asm организуют цикл записи/чтения буфера данных?
16. Для каких целей используется команда сброса для i8251?
17. Опишите параметры физических сигналов на выходе интерфейса RS-232.
18. Структура и назначение многофункциональные платы в/выв (multi I/O card).
19. Назначение и структура интерфейса Centronics/
20. Работа ПЭВМ с параллельными портами 1, 2, 3.
21. Назначение и структура интерфейса RS232/
22. Работа ПЭВМ с последовательными портами 1, 2, 3.
23. Отличительные особенности различных микросхем UART 8150/16450/16550.
24. Что вы знаете о интерфейсе USB1,2.

Тема 7.
Решение логических и вычислительных задач на ПЭВМ с использованием процедур и макрокоманд языка Assembler. Программирование на языке Assembler в среде Windows

1. Укажите существующие форматы данных МП i8086.
2. Какие сегментные регистры по умолчанию адресуют начало сегментов кодов, стека, данных?
3. Опишите формат оператора и директивы в ассемблере IBM PC.
4. Опишите использование меток в ассемблере IBM PC.
5. Структура программы на ассемблере IBM PC.
6. Директивы SEGMENT, ASSUME и PROC.
7. Каково назначение директивы ASSUME в языке MASM?
8. Типизация данных в ассемблере IBM PC.
9. Директивы определения данных.
10. Выражения в ассемблере IBM PC.
11. Какие действия выполняет МП при вызове процедуры типа FAR и NEAR?
12. Арифметические, логические и операции отношения в ассемблере IBM PC.
13. Операции, возвращающие значения и присваивающие атрибуты.
14. Процедуры, сегменты, модули. Директивы EXTERN и PUBLIC.
15. Особенности организации .COM программы.
16. Макрокоманды и библиотеки макроопределений в ассемблере IBM PC.
17. Библиотеки стандартных модулей.

Тема 8.
Изучение структуры, сборка, оценка производительности персонального компьютера

1. Что такое пиковая производительность?
2. Назовите основные виды тестов измерения производительности ПК .
3. Что такое тесты SPECxx?
4. Через какие интерфейсы может подключаться клавиатура к PC?
5. Для подключения каких устройств используют IDE-интерфейс?

6. Какие могут быть возможные причины того, что при первом включении в BIOS отсутствует информация о каком-либо IDE-устройстве?
7. Что необходимо сделать, чтоб осуществить загрузку компьютера с компакт-диска?
8. Какова нормальная температура работающего процессора? В чем может быть причина высокой температуры процессора?
9. Нужно ли настраивать BIOS? Если да, то как?
10. Что такое твикинг ОС? Как его правильно осуществить?
11. Назовите средства мониторинга состояния компьютера, доступные из BIOS.
12. Расскажите о режимах переключателей на винчестере, опишите состояние переключателей для случая подключения одного или нескольких винчестеров.
13. Опишите последовательность подключения процессора к материнской плате.
14. Опишите возможные варианты шлейфов для подключения винчестеров, их особенности.
15. Опишите возможные варианты жестких дисков, их типы и характеристики.
16. Опишите возможные варианты видеоконтроллеров, их типы и характеристики.
17. Опишите типы оперативной памяти, подключаемые к современным компьютерам.
18. Опишите форм-факторы современных материнской плат.
19. Опишите современные типы сокетов МП.
20. Опишите современную маркировку МП фирмы Intel и AMD.
21. Как осуществляется логическое форматирование жёсткого диска с помощью программы fdisk и PatitionMagic 8-0?
22. Опишите стандартное разрешение монитора CRT (рекомендуемое разрешение для стандартных диагоналей).

ПРИЛОЖЕНИЯ

1. ПЕРЕЧЕНЬ ОСНОВНЫХ КОМАНД ЯЗЫКА АССЕМБЛЕРА МП Intel 8086

Мнемоника	Название команды	Алгоритм работы	Флаги
1	2	3	4
ADC dst, src	Сложение с использова-нием бита CF	dst:=(dst)+(src)+(CF)	AF,CF,OF PF,ZF,SF
ADD d dst, src st, src	Сложение	dst:=(dst)+(src)	AF,CF,OF PF,ZF,SF
AND dst, src	Логическое умножение	dst:=(dst)^(src)	CF,OF=0, SF,ZF,PF
CALL dst	Вызов процедур	dst:=(ip)+(cer):(смещ	
CBW	Преобразование байта в слово		
CLC	Сброс бита CF	(CF)=0	CF
CLD	Сброс бита DF	(DF)=0	DF
CLI	Сброс бита IF	(IF)=0	IF
CMC	Инверсия бита CF	если (CF)=0, то (CF)=1	CF
CMP dst, src	Сравнение двух операндов	(dst)-(src)	AF,CF,OF PF,ZF,SF
CMPS dst, src (CMPSB,CMPSW)	Сравнение строки байт или слов	(dst)-(src)	AF,CF,OF PF,ZF,SF
CWD	Преобразование слова в двойное слово		
DEC dst	Уменьшение операнда на единицу	dst:=(dst)-1	AF,OF,PF SF,ZF,CF
DIV SRC	Деление без знака	ac:=guot((ext:ac)/(src) ext:=rem((ext:ac) /(src))	
HLT	Останов процессора		
IDIV src	Деление целых чисел со знаком	ac:=guit((ext:ac) /(src)) ext:=rem((ext:ac) /(src))	AF,CF,OF SF,ZF,PF
IMUL src	Умножение	dst:=(ac)*(src)	CF, OF
IN ac, port	Ввод байта, слова	src=port	
INC dst	Увеличение операнда на единицу	dst:=(dst)+1	AF,OF,PF SF,ZF,CF
INT type	Прерывание	(sp)=sp+2 ((sp)+1:(sp))=F	IF, TF
IRET	Выход из процедур	(ip)=((sp)+1:(sp) sp)=(sp)+2 (cs)=((sp)+1:(sp) (sp)=(sp)+2 F=((sp)+1:(sp)	AF,OF,PF SF,ZF,IF TF,CF,DF
JA или JNBE	Перейти, если выше/ не ниже или равно		CF\ /ZF
JAE или JNB	Перейти, если выше или равно /не ниже		CF=0
1	2	3	4
JB или JNAE	Перейти, если ниже/ не выше или равно		CF=1
JBE или JNA	Перейти, если ниже или равно /не выше		CF\ /ZF=1
JCXZ	Перейти по нулевому со-держимому CX		(CX)=0
JE или JZ	Перейти, если равно /нуль		ZF=1
JG или JNLE	Перейти, если больше/ не меньше или равно		(SF+OF)\ / =0
JLE или JNG	Перейти, если меньше или равно/не больше		(SF+OF)\ / ZF
JGE или JNL	Перейти, если больше или равно/не меньше		SF+OF=0

JL или JNGE	Перейти, если меньше/ не больше или равно		SF+OF=1
JMP метка	Безусловный переход		
JNE или JNZ	Перейти, если не равно/ не нуль		ZF=0
JNO	Перейти, если нет переполнения		OF=0
JNP или JPO	Перейти, если нет пари-тета/паритет нечётный		PF=0
JNS	Перейти, если нет знака		SF=0
JO	Перейти, если есть переполнение		OF=1
JP или JPE	Перейти, если есть паритет		PF=1
JS	Перейти, если есть знак		SF=1
LANF	Передача младшего байта регистров флагов в регистр AH	(AH)<-((SF):(ZF): (AF):(PF):(CF))	
LDS reg, src	Загрузка адреса в DX, а смещение в reg	(reg)<-(src) (DS)<-(src+2)	
LEA reg, src	Загрузка исполнитель-ного адреса	(reg)<-(src)	
LES reg, src	Загрузка адреса в ES, а смещение в reg	(reg)<-(src) (ES)<-(src+2)	
LOCk	Для мультипроцессор-ных систем		
LODS src (LODSB,LODSW) LOOPZ,LOOPE	Загрузка строки байт или слова Цикл, если z=1,исх.=/0 и выход из цикла, если бит ZE=0 или CX=0	ac:=(src)	
LOOPNZ или LOOPNE	Цикл, если сброшен бит ZF=0,CX=/0. Выход из цикла CX=0 или ZF=1		
1	2	3	4
MOV dst, src	Пересылка	dst:=(src)	
MOVS dst, src (MOVSB,MOVSW	Пересылка строк байт или слов	dst:=(src)	
MUL src	Умножение беззнаковое	ext:ac:=(ac)*(SRC)	CF, OF
NEG dst	Изменение знака	dst:=0-(dst)	AF,CF,OF PF,SF,ZF
NOP	Команда пустой операции		
NOT src	Логическое отрицание или инверсия	src:=(src)	
OR dst, src	Логическое сложение	dst:=(dst)\ /(src)	CF,OF,PF SF,ZF,AF
OUT port, ac	Вывод байта или слова в порт		
POP	Загрузить слово из стека	(dst)=((sp)+1|(sp (sp)=(sp)+2	
POPF	Загрузка регистра флагов из стека	F=((sp)+1:(sp)) (sp)=(sp)+2	CF,OF,AF SF,ZF,TF, IF,DF
PUSH	Записать слово в стек	(sp)=(sp)-2 ((sp)+1:(sp)=(src	
PUSHF	Записать регистр флагов в стек	(sp)=(sp)-2 ((sp)+1:(sp)=F	
RCL dst, count	Сдвиг циклический влево через бит CF		CF,OF
RCR dst, count	Сдвиг циклический вправо через CF		CF,OF
REP (REPZ, REPE, REPNZ, REPNE)	Повторение строковой операции		Зависят от стро-ковой операции
RET	Выход из процедур		
ROL dst, count	Сдвиг циклический влево		CF,OF
ROR dst, count	Сдвиг циклический вправо		CF,OF
SHR dst, count	Логический сдвиг вправо		CF,OF,PF, SF,ZF
SUB dst, src	Вычитание	dst:=(dst)-(src)	AF,CF,OF

			SF,ZF,PF
SAR dst, count	Сдвиг арифметический вправо		CF,OF,PF, SF,ZF
SBB dst, src	Вычитание с заёмом	dst:=(dst)-(src)-(CF)	AF,CF,OF PF,SF,ZF
SCAS dst (SCASB, SCASW)	Команда сканирования	(ac)-(dst)	AF,CF,OF PF,SF,ZF
TEST dst, src	Логическое сравнение	(dst)/\(src)	CF,OF,PF, SF,ZF,AF
STC	Установить бит CF	(CF)=1	CF
1	2	3	4
STD	Установить бит DF	(DF)=1	DF
STI	Установить бит IF	(IF)=1	IF
STOS dst (STOSB,STOSW)	Запоминание строки байт или слов	dst:=(ac)	
WAIT	Перевод процессора в ожидание		
XLAT	Замена содержимого AL на значение из таблицы	AL:=((BX)+(AL))	
XOR dst, src	Сложение по модулю 2	dst:=(dst)+(src)	CF,OF,PF, SF,ZF

2. БАЗОВЫЕ ЗАДАЧИ ДЛЯ МИКРОЛАБ И ПЭВМ

Задача 1

Написать программу, обеспечивающую принятие числа из порта ввода с адресом FFFB и перезапись информации в порт вывода с адресом FFFA без обработки. Предварительно осуществить настройку интерфейсной микросхемы К580ВВ55 на работу в соответствующих режимах.

Задача 2

Принять число из порта ввода с адресом FFFB, поразрядно инвертировать. Результаты выдать в порт вывода с адресом FFFA. Предварительно осуществить настройку интерфейсной микросхемы К580ВВ55 на работу в соответствующих режимах. Решение задачи 1 с использованием транслятора MASM.

Задача 3

Принять число из порта ввода с адресом FFFB, сдвинуть на 1 разряд влево. Результаты выдать в порт вывода с адресом FFFA. Предварительно осуществить настройку интерфейсной микросхемы К580ВВ55 на работу в соответствующих режимах.

Задача 4

Подсчитать число битов, установленных в «1» в порту FFFB, и результат выдать в двоичном коде в порт с адресом FFFA. Предварительно осуществить настройку интерфейсных микросхем КР580ВВ55 на работу в соответствующих режимах.

Задача 5

Ввести число из порта с адресом FFFBh, подсчитать число бит, установленных в «1», результат выдать в порт вывода FFFAh, зажигая число светодиодов, равное числу установленных бит.

Примечание. Осуществить настройку интерфейсных микросхем КР580ВВ55 на работу в соответствующих режимах.

Задача 6

Написать программу бегущих огней на светодиодах (порт с адресом FFFA). Произвести инициализацию портов.

Задача 7

Написать программу, обеспечивающую получение числа из порта ввода (FFFB) и визуальное отображение числа в шестнадцатеричном коде на индикаторах дисплея. Осуществлять предварительную настройку микросхем КР580ВВ55 и КР580ВВ79 на работу в соответствующих режимах.

Задача 8

Написать программу, имитирующую бросание игральной кости. Программа работает следующим образом: нажатием любой клавиши (кроме «Сброс» и «Прер.») можно заставить «кость» начать «катиться». Еще одно нажатие любой клавиши, кроме указанных, останавливает «кость», а на дисплее в самой левой позиции появляется ее значение, т. е. цифра от 1 до 6. Вывод значений на дисплей и опрос клавиатуры осуществляется самой программой.

Задача 9

Написать программу, переставляющую элементы массива soure (10,20,30,40) в массив dest в обратном порядке.

Задача 10

С помощью команды XLAT заменить элементы массива MAS (десять цифр от 0 до 9) на элементы массива TABL (первые десять букв английского алфавита) .

Задача 11

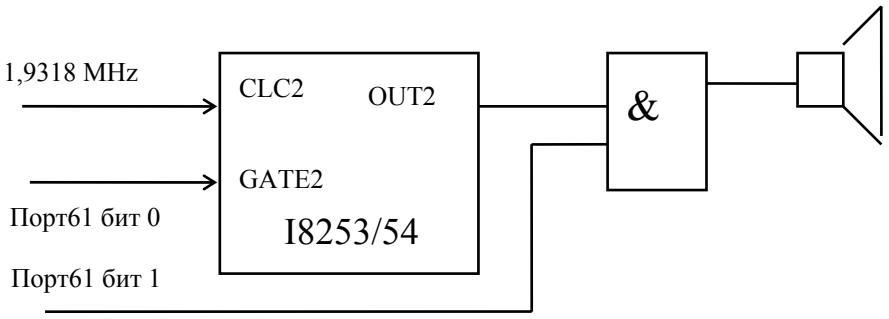
Написать программу вывода сообщения на дисплей, используя прерывание DOS int 21h.

Задача 12

Написать программу, обеспечивающую получение символа с клавиатуры ПЭВМ и отображение его на дисплее.

Задача 13

Написать программу организации режима «бегущих огней» на светодиодах, подключенных ко всем выходным линиям порта принтера (порт с адресом 378 (LPT1) выполнен в стандарте (Centronix)). Произвести инициализацию портов.



Задача 14

Написать программу звучания динамика. Канал управления звуком приведен на рисунке.

Задача 15

Написать программу, проверяющую связь с FDD путем включения светодиода FDD.

Задача 16

Написать программу, включающую режим работы дисплея 40×25, используя функции драйвера экрана, доступные по прерыванию INT 10H.

Примечание. Стандартная функция 00H устанавливает режим работы экрана. Режим 00H - символьный 40x25 с подавлением цвета.

Задача 17

Написать программу обработки прерываний при переполнении. Делимое находится в двух регистрах AX, DX; делитель равен двум; частное не умещается в одном регистре. Процедура DVROK должна перехватывать прерывание INT0 и выводить на экран соответствующее сообщение.

3. КОНТРОЛЬНЫЕ ЗАДАЧИ ДЛЯ ВЫПОЛНЕНИЯ НА ПЭВМ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА АССЕМБЛЕР

Задача 1

В матрице *C (M ,N)* найти значение элемента $C(x_0, y_0) = max\ C\ (x, y)$. Индексы x_0, y_0 вывести на печать. $M = 3$, $N = 3$

Задача 2

В матрице *C (M, N)* обнулить все диагональные элементы. $M = 3$, $N = 3$.

Задача 3

Найти произведение матрицы *A (N, N)* на матрицу *B (N, N)*. $N = 3$.

Задача 4

В матрице *C (M, N)* все отрицательные элементы умножить на 5. $M = 3$, $N = 3$.

Задача 5

В матрице *C (M, N)* поменять знак у положительных элементов. $M = 3$, $N = 3$.

Задача 6

Найти сумму всех элементов матрицы *C (M, N)*. $M = 3$, $N = 3$.

Задача 7

Найти сумму всех диагональных элементов матрицы *C (M, N)*. $M = 3$, $N = 3$.

Задача 8

Из каждого элемента матрицы *C (M, N)* вычесть значение минимального элемента в его столбце. $M = 3$, $N = 3$.

Задача 9

В матрице *C (M, N)* найти сумму элементов всех четных строк. $M = 3$, $N = 3$.

Задача 10

Найти сумму элементов всех нечетных столбцов матрицы *C (M, N)*. $M = 3$, $N = 3$.

Задача 11

Умножить матрицу *A (N, N)* на константу *E*. $N = 3$, $Q = 3$

Задача 12

Для матрицы *C (M, N)* найти вектор *A (M)*, каждый *i*-й элемент которого равен сумме элементов *i*-го столбца. $M = 3$, $N = 3$.

Задача 13

В матрице *C (N, N)* уменьшить значение каждого элемента на величину диагонального элемента соответствующей строки. $N = 3$.

Задание 14

Транспортировать матрицу *C (M, N)*. $M = 3$, $N = 3$.

Задача 15

Для матрицы *C (M, N)* вывести на печать индексы всех отрицательных элементов. $M = 3$, $N = 3$.

Задача 16

В матрице *C (M, N)* значения всех элементов, меньших 1 заменить на обратные. $M = 5$, $N = 7$.

Задача 17

Все положительные элементы матрицы *C (M, N)* разделить на константу *E*, все отрицательные – умножить на эту величину *E*. $M = 3$, $N = 3$, $E = 4$.

Задача 18

В матрице *C (M, N)* заменить все отрицательные элементы на нуль. $M = 3$, $N = 3$.

Задача 19

В матрице *C (M, N)* сосчитать количество отрицательных элементов. $M = 3$, $N = 3$.

Задача 20

В матрице *C (M, N)* вывести на печать индексы всех элементов, меньших 20. $M = 3$, $N = 3$.

Задача 21

Вычесть из каждого элемента матрицы *C (M, N)* максимальный элемент соответствующего столбца. $M = 3$, $N = 3$.

Задание 22

Определить разность наибольшего и наименьшего элементов матрицы *A (M, N)*. $M = 3$, $N = 3$.

Задача 23

Построить матрицу *A (M, N)*. Подсчитать количество нулевых элементов. $M = 3$, $N = 3$.

Задача 24

Определить количество нечетных чисел в массиве чисел *A (M, N)*. $M = 3$, $N = 3$

Задача 25

Для матрицы *C (M, N)* найти произведение элементов всех четных строк. $M = 3$, $N = 3$.

Задача 26

Множество точек в трехмерном пространстве задано своими координатами *X, Y, Z*. Подсчитать количество точек из множества *N*, которые принадлежат полупространству

$$X+Y+Z=10,\ N=3.$$

Задача 27

Множество точек в трехмерном пространстве задано своими координатами *X (N), Y (N), Z (N)*. Подсчитать количество точек которые лежат внутри сферы

$$X^2 + Y^2 + Z^2 = 10,\ N = 3.$$

Задача 28

Найти наименьшее *N*, при котором $\sum_{k=1}^{N} \frac{1}{K} \succ A$, $A = 4$

Задача 29

Найти сумму *M* членов арифметической прогрессии 7, 10, 13, 16, 19, 22, … двумя способами. $M = 6$.

Задача 30

Найти среднее арифметическое массива из *N* элементов. $N = 10$.

Задача 31

Организовать массив данных, каждый *i*-й элемент которого равен сумме *i*-х элементов двух других массивов данных из *N* элементов каждый. *N* = 5.

Задача 32

Организовать массив данных, каждый *i*-й элемент которого равен разности *i*-х элементов двух других массивов данных из *N* элементов каждый. *N* = 3.

Задача 33

Найти произведение всех элементов массива данных из *N* элементов. *N* = 5.

4. ТЕКСТ ПРОГРАММЫ, ИМИТИРУЮЩЕЙ БРОСАНИЕ ИГРАЛЬНОЙ КОСТИ

```
                        Установка знач. сегментных регистров
        MOV CX, CS      ; значение регистра сегмента данных
        MOV DS, CX      ; равно значению регистра сегмента кода
                        Контроллер КР 580ВВ79 - в исходное сост.:
        MOV DX, OFFEAH  ; команда программного сброса
        MOV AL, OD3H    ; и сброса регистра состояния
        OUT DX          ; контроллера КР580ВВ79
                        Ожидание нажатия клавиши, т.е. запуска
                        «катящейся кости»:
READKEY:
        MOV DX, OFFEAH  ; адрес порта команд КР 580ВВ79
        IN DX           ; считывание регистра состояния
        AND AL, OFH     ; счётчик ОМ-ОЗУ=0 (клавиша нажата)?
        JZ READKEY+3    ; если нет, продолжать ожидание
        CALL READDATA   ;считать код клавиши, чтобы освободить
                        ОМ-ОЗУ
ZERO:                   Запуск:
        MOV BX, 0000H   ; начальное значение счётчика «кости»
START:
        INC BX          ; увеличить счётчик на 1
        CMP BL, 07H     ; если счётчик равен 7
        JZ ZERO         ; на сброс в 0
        MOV DI, BX      ; преобразование значения счётчика в 7-
                        сегментный
        MOV CL CDTBL [DI-1] ; код для индикации в поле регистра DI
        MOV DX, OFFEAH  ; команда «запись в ОЗУ
        MOV AL, 087H    дисплея» в 7-ю позицию
        OUT DX
        MOV DX, OFFE8H  ; вывод значения счётчика «кости»
        MOV AL, CL      ; на дисплей (в порт КР 580ВВ79)
        OUT DX
                        Ожидание нажатия клавиши (остановки
                        «катящейся кости»
        MOV DX, OFFEAH  ; считывание регистра состояния
        IN DX           ; контроллера КР580ВВ79
        AND AL OFH      ; счётчик ОМ-ОЗУ=0 (клавиша нажата)?
        JZ START        ; если нет, продолжить просчёт значений
        CALL READATA    ; считать код клавиши, чтобы освободить
                        ОМ-ОЗУ
        JMP READKEY     ; на ожидание запуска
                        Процедура считывания кода клавиши из
                        ОМ-ОЗУ КР580ВВ79:
READATA:
        MOV DX, OFFEAH  ; команда контроллера
        MOV AL, 040H    ; КР580ВВ79 «считывание данных»
        OUT DX
        MOV DX, OFFE8H  ; считать код клавиши AL
        IN DX           ; результат в регистре AL
        RET             ; возврат

                        Таблица преобразования чисел от 1 до 6
                        в код для 7-сегментной индикации
        DB 06H                  ; 7-сегментный код «1»
```

```
                DB 05BH         ; 7-сегментный код «2»
                DB 04FH         ; 7-сегментный код «3»
                DB 066H         ; 7-сегментный код «4»
                DB 06DH         ; 7-сегментный код «5»
                DB 07DH         ; 7-сегментный код «6»
```

5. ТЕСТОВАЯ ПРОГРАММА ДЛЯ МИКРОЛАБ

```
0000                            .model tiny
0000                            .code
                                org 100h
0100  8C C8           Start:    mov   ax,cs          ; установка сегментных регистров
0102  8E D8                     mov   ds,ax
0104  BA FFEA                   mov   dx,0FFEAh      ; сброс КР      580 ВВ 79
0107  B0 D3                     mov   al,0D3h
0109  EE                       out   dx,al
010A  B9 0096                   mov   cx,150         ; задержка
010D  E2 FE                     loop  $
010F  BB 013Cr                  mov   bx,offset Tbl  ; адрес таблицы перекодировки в BX
0112  BA FFFB                   mov   dx,0FFFBh      ; адрес порта ввода
0115  EC                       in    al,dx          ; чтение порта ввода
0116  E8 0007         OutData: call  Convert        ; вывод кода на дисплей
0119  EC                ReadAgn:in    al,dx          ; чтение порта ввода
011A  3A E0                     cmp   ah,al          ; если значение не изменилось, то
011C  74 FB                     je    ReadAgn        ; читать снова
011E  EB F6                     jmp   OutData
0120  52                Convert: push  dx            ; процедура перекодировки
                                                     содержимого
0121  8A E0                     mov   ah,al          ; регистра AL в семисегментный код и
0123  BA FFEA                   mov   dx,0FFEAh      ; вывод его на индикатор
0126  B0 90                     mov   al,90h
0128  EE                       out   dx,al
0129  BA FFE8                   mov   dx,0FFE8h
012C  8A C4                     mov   al,ah
012E  24 0F                     and   al,0Fh
0130  D7                       xlat
0131  EE                       out   dx,al
0132  8A C4                     mov   al,ah
0134  B1 04                     mov   cl,4
0136  D2 E8                     shr   al,cl
0138  D7                       xlat
0139  EE                       out   dx,al
013A  5A                       pop   dx
013B  C3                       ret
013C  3F                Tbl    db 3Fh               ; семисегментный код "0"
013D  06                       db    06h            ; семисегментный код "1"
013E  5B                       db    5Bh            ; семисегментный код "2"
013F  4F                       db    4Fh            ; семисегментный код "3"
0140  66                       db    66h            ; семисегментный код "4"
0141  6D                       db    6Dh            ; семисегментный код "5"
0142  7D                       db    7Dh            ; семисегментный код "6"
0143  27                       db    27h            ; семисегментный код "7"
0144  7F                       db    7Fh            ; семисегментный код "8"
0145  6F                       db    6Fh            ; семисегментный код "9"
0146  77                       db    77h            ; семисегментный код "A"
0147  7C                       db    7Ch            ; семисегментный код "B"
0148  39                       db    39h            ; семисегментный код "C"
0149  5E                       db    5Eh            ; семисегментный код "D"
014A  79                       db    79h            ; семисегментный код "E"
014B  71                       db    71h            ; семисегментный код "F"
                                end Start
```

ЛИТЕРАТУРА

1. Микропроцессоры. Архитектура и проектирование микроЭВМ. Организация вычислительных процессов. т. 1. Под ред. Л. Н. Преснухина. Минск, 1987.
2. Микропроцессорный комплект К1810: Структура, программирование, применение. Справочная книга под ред. Ю.М. Казаринова. М., 1990.
3. *Митницкий В. Я.* Архитектура IBM PC и язык Ассемблера. М., 2000.
4. *Юров В.* ASSEMBLER. СПб., 2001.
5. *Майко Г. В.* Assembler для IBM PC. М., 1999.
6. *Скэнлон Л.* Персональные ЭВМ IBM PC/XT. М., 1989.
7. *Нортон П., Соухэ Д.* Язык Ассемблера для IBM. М., 1992.
8. *Чекатков А. А.* Использование Turbo Assembler при разработке программ. Киев, 1994.
9. *Шалатонин И. А.* Микропроцессоры и ПЭВМ. Мн., 2004.
10. *Пустоваров В. И.* Ассемблер: программирование и анализ корректности машинных программ. Киев, 2000.
11. *Шалатонин И. А.* Изучение интерфейсов микропроцессорных систем. Мн., 2003.
12. *Смолин В. Я. и др.* Аппаратно-программные средства устройств на базе микропроцессора КМ1810ВМ86. Методическое пособие для начинающих. ч. 1. ЦООНТИ «ЭКОС», 1989.
13. *В.В.Долгий ,Шалатонин И. А.* Программирование микропроцессорных систем . Мн., 2004.

ISBN 978-985-485-783-1